SELF-ADAPTIVE AUTHORISATION INFRASTRUCTURES

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT
IN THE SUBJECT OF COMPUTER SCIENCE
FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY.

By
Christopher Michael Bailey
March 2015

Abstract

Traditional approaches in access control rely on immutable criteria in which to decide and award access. These approaches are limited, notably when handling changes in an organisation's protected resources, resulting in the inability to accommodate the dynamic aspects of risk at runtime. An example of such risk is a user abusing their privileged access to perform insider attacks.

This thesis proposes self-adaptive authorisation, an approach that enables dynamic access control. A framework for developing self-adaptive authorisation is defined, where autonomic controllers are deployed within legacy based authorisation infrastructures to enable the runtime management of access control. Essential to the approach is the use of models and model driven engineering (MDE).

Models enable a controller to abstract from the authorisation infrastructure it seeks to control, reason about state, and provide assurances over change to access. For example, a modelled state of access may represent an active access control policy. Given the diverse nature in implementations of authorisation infrastructures, MDE enables the creation and transformation of such models, whereby assets (e.g., policies) can be automatically generated and deployed at runtime.

A prototype of the framework was developed, whereby management of access control is focused on the mitigation of abuse of access rights. The prototype implements a feedback loop to monitor an authorisation infrastructure in terms of modelling the state of access control and user behaviour, analyse potential solutions for handling malicious behaviour, and act upon the infrastructure to control future access control decisions.

The framework was evaluated against mitigation of simulated insider attacks, involving the abuse of access rights governed by access control methodologies. In addition, to investigate the framework's approach in a diverse and unpredictable environment, a live experiment was conducted. This evaluated the mitigation of abuse performed by real users as well as demonstrating the consequence of self-adaptation through observation of user response.

Acknowledgements

I would like to express my thanks and gratitude to my supervisor, Rogério de Lemos. Through all the challenges, hard decisions, and deadlines, he has always been there to encourage me, and offer his support, insightful knowledge, and friendship. Without his guidance this thesis would not have happened.

Many thanks to the members of the thesis panel, Eerke Boiten and Julio Hernandez-Castro, for their useful feedback and advice. In particular, my thanks goes to David Chadwick, who has supported me greatly in my understanding of authorisation infrastructures, as well as providing invaluable support throughout the Ph.D.

A special thanks goes to my fiancée Charlotte Pilbeam. Words cannot express how grateful I am for her love, support, and the sacrifices she has made on my behalf. I'd also like to express my thanks to my parents, my brother, and my grandparents, for all the support they have given me, their patience, and belief in me when times were hard. My thanks also goes to my future mother-in-law and father-in-law for their support, and providing myself and Charlotte with a roof over our heads, enabling our future.

I would also like to thank Lionel Montrieux, Michel Wermelinger, and Yijun Yu for their collaboration and advice on model verification. Carlos Eduardo da Silva for his advice and help in understanding model driven engineering. Kristy Siu for her advice and help in understanding identity management. And, Sally Fincher and Andy King for their advice and professional support. Not forgetting my friends Andy Mannering, Phil Stevens, Davin McCall, Matthew Mole, Dan Knox, and Michael Berry, in their support and advice throughout the Ph.D. I'd also like to thank all of the anonymous participants who voluntarily took part in a crucial experiment, enabling the evaluation of this thesis.

Finally, my thanks goes to the Engineering and Physical Sciences Research Council in supporting me with a Ph.D. research grant, and all of the supporting staff within the School of Computing, University of Kent.

Contents

List of Tables

List of Figures

List of Publications

The following publications are all original in contribution and relevant to this thesis, where the author of this thesis has been the main contributor.

1. Bailey, C., Chadwick, D. W. and de Lemos, R. (2014). Self-adaptive federated authorization infrastructures. *Journal of Computer and System Sciences*, 80(5), pp. 935-952, Special Issue on Dependable and Secure Computing (**Chapter 3 and Chapter 5**)

2. Bailey, C., Montrieux, L., de Lemos, R., Yu, Y. and Wermelinger, M. (2014). Run-time generation, transformation, and verification of access control models for self-protection. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, New York, NY, USA: ACM, SEAMS 2014, pp. 135-144 (**Chapter 4**)

3. Bailey, C., Chadwick, D. W., de Lemos, R. and Siu, K. W. (2013). Enabling the autonomic management of federated identity providers. In *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet - Volume 7943*, Berlin, Heidelberg: Springer-Verlag, AIMS'13, pp. 100-111 (**Chapter 5 and Appendix B**)

4. Bailey, C. (2012). Application of self-adaptive techniques to federated authorization models. In *Proceedings of the 34th International Conference on Software Engineering*, Piscataway, NJ, USA: IEEE Press, ICSE '12, pp. 1495-1498, Doctoral Symposium (**Chapter 3**)

5. Bailey, C., Chadwick, D. W. and de Lemos, R. (2011). Self-adaptive authorization framework for policy based RBAC / ABAC models. In *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, Washington, DC, USA: IEEE Computer Society, DASC '11, pp. 37-44 (**Chapter 3**)

Chapter 1

Introduction

A critical concern for organisations surrounds the assurances of confidentiality, integrity, and availability of their computer based resources. To provide such assurances, organisations utilise access control to protect against unauthorised access. Current approaches, such as Role Based and Attribute Based Access Control (RBAC and ABAC) [58, 97], rely on the assessment of immutable criteria before awarding access.

Whilst RBAC and ABAC approaches are capable of protecting from unauthorised access, they do not take into account the dynamic aspects of risk at runtime, due to uncertainty in user behaviour. A number of approaches in improving access control have been proposed in response to such concerns. These include dynamic access control methods that consider risk [83], trust [12], and usage [109], as part of the criteria for accessing resources, whereby they seek to enable a fine granularity of control. However, such approaches introduce complexity into the definition of access control, potentially restricting access unnecessarily, or introducing errors. Moreover, to be effective, these approaches must accommodate every possible abuse at time of defining the criteria for access.

Regardless of adopting a fine grained approach to access control, abuse of access is still possible. Any form of access, no matter how restrictive, presents the risk of attacks due to uncertainty in user behaviour. To accommodate for this risk, organisations employ a range of methods [92] to monitor and audit access within their systems and resources. Traditionally, human administrators are relied upon to actively identify and drive changes in access control in response to detected abuse, natural organisational change, or identified errors in the criteria for access. However, as evident by recent scandals [10, 20, 141], much more could be done.

Implementations of access control (e.g., authorisation infrastructures [32]), must be capable in handling the dynamic aspect of risk at runtime, driven by the uncertainty in user behaviour. It is therefore necessary for such systems to actively observe how access rights are being used, in order to infer whether the current criteria and assignment of access are enabling a user to conduct malicious activity.

1.1 Research Problem

Authorisation infrastructures [32] consist of integrating independent systems for implementing access control. These include systems to facilitate storage of user access rights; user authentication; the issuing, release, and validation of user access rights; decision for access; and enforcement of access control decisions. As such, no one system exhibits a complete view of the configuration of access.

It is therefore challenging for human administrators to maintain a true awareness of the configuration of access, particularly within a runtime environment. With no complete view of access, obtaining assurances [57] against changes made to mitigate the abuse of access is limited. This potentially enables erroneous changes that cause a greater impact to the organisation over identified abuse.

In addition, and as evident in case studies of historic insider attacks [27], the use of human administrators alone is inefficient in mitigating abuse in a timely manner. Improving on access control methodologies is one solution, yet such approaches [12, 66, 83, 109] are unable to actively mitigate abuse, since they are constrained to a static definition of the criteria for access control at runtime. Given these problems, this thesis considers the following research question.

> *How can access control be managed to handle unpredictable changes in user behaviour at runtime?*

Enabling the management of access control presents several challenges. First, authorisation infrastructures that implement access control contain a multitude of systems in which an access control methodology is achieved. There are many diverse implementations that can exist within an authorisation infrastructure [81, 112, 114], and as such, a challenge concerns how a single solution can integrate, observe, and adapt such a variety of systems at runtime.

Second, an authorisation infrastructure may be owned by a single organisation (i.e., centralised), or multiple organisations (i.e., federated [93, 140]), where access to resources is assigned by multiple third party organisations. As a result, observing and obtaining a unified perception of the configuration of access is challenging, due to the need to operate across multiple management domains.

Third, given the nature of change against the criteria and assignment of access, a certain degree of assurance is required. As with the need to verify the definition of an access control model against requirements at development time [57], it also becomes necessary to verify any changes in the definition of access control at runtime.

1.2 Research Approach

Given the current limitations in access control, the approach presented in this thesis proposes a *framework that enables the automated management of access control through self-adaptation*.

The framework observes and controls authorisation infrastructures at runtime, via the automated adaptation of access control policies and user privileges. The framework defines a means for adapting the criteria and assignment of access depending on the presence of insider threat (related to the abuse of access control). The framework has been implemented as a prototype consisting of an autonomic controller, configurable within RBAC and ABAC authorisation infrastructures.

The approach is demonstrated as a reactive means to handling insider threat. This is an important contrast to how existing approaches aim to handle malicious behaviour in access control. Rather than deploying restrictive access control policies that attempt to preempt malicious behaviour, this approach positions the automated adaptation of access control in response to malicious behaviour.

The approach relies on self-adaptation [121] and model driven engineering [16] as a means to achieve automated management of authorisation infrastructures. Applying self-adaptive techniques to authorisation infrastructures enables the infrastructure to observe, reason, and act on its own configuration of access control. Through the use of a feedback loop [22], it is possible to employ a clear separation of concerns between the decision for access, and decision for a management change, therefore reducing the complexity in the criteria for access that dynamic access control approaches introduce.

Model driven engineering (MDE) [16] enables a self-adaptive system to abstract and generate models of its own system and environment state, such as the runtime state of access in terms of the configuration of access control. Models are important in enabling the reasoning of environment and system state, as well as the adaptation of system state. For instance, model transformation [126] enables the conversion of one model (e.g., a particular implementation of an RBAC access control policy) to another (e.g., a homogeneous model of RBAC). Through model transformation, a self-adaptive solution is therefore able to observe and adapt diverse implementations of access control.

Finally, in order to evaluate the approach, the framework is deployed within a centralised (single organisation) RBAC authorisation infrastructure, and a federated (multi-organisation) ABAC authorisation infrastructure. The framework is then evaluated in terms of simulating case studies of insider threat, as well as evaluated within a live experiment using gamification [60]. The latter demonstrates a novel approach to evaluating self-adaptive systems under diverse and unpredictable change.

1.3 Thesis Contributions

The goal of this thesis is to design and evaluate a framework for the runtime management of access control. In pursuit of this goal, this thesis presents several contributions.

1. The definition of a framework to enable the automated management of authorisation infrastructures through self-adaptation, capable in mitigating the abuse of access by adapting access control policies and user privileges at runtime.

2. An approach to generating and transforming a model of the state of access, causally connected to the configuration of access control within distributed and complex systems at runtime.

3. A means to automating the management of federated identity providers [93], in order to mitigate the abuse of access given the existence of multiple management domains.

4. A novel approach to evaluating self-adaptive systems through gamification [60], enabling the generation of data representative of change (e.g., abuse of access) that is diverse and unpredictable.

1.4 Thesis Structure

The rest of this thesis is structured as follows. Chapter 2 presents a literature review aimed at scoping the thesis's research problem and reviewing related work. The literature review discusses the following topics: traditional methods in access control, insider threat, recent approaches in access control with respect to insider threat, self-adaptive software systems, and self-protecting software systems.

Chapter 3 proposes the Self-Adaptive Authorisation Framework (SAAF), a framework for enabling the automated management of authorisation infrastructures. It presents the conceptual aspects of the approach, notably the role of autonomic controllers in observing and adapting authorisation infrastructures. The approach is also classified with respect to a recent taxonomy of self-protecting systems [150], leading to a discussion of the approach and related work.

Chapter 4 describes the implementation of a SAAF autonomic controller prototype, deployable within RBAC and ABAC authorisation infrastructures. A major focus of the chapter is on the role of models at runtime, and defining a means in which the controller can be integrated with diverse implementations of authorisation infrastructures. A preliminary experiment then demonstrates mitigation of a historic case of insider threat, simulated within an RBAC authorisation infrastructure.

Chapter 5 presents a formal definition of change within authorisation infrastructures. The formal approach is used to drive the simulation of a fictitious insider attack within a federated ABAC authorisation infrastructure. Several experiments are discussed in this chapter, evaluating the prototype's ability to mitigate insider attacks given limitations in the observation and control of federated authorisation infrastructures.

Chapter 6 seeks to complement the previous evaluation by deploying SAAF within a real user environment. In this chapter, an online game has been developed for emulating an organisational resource, protected by a centralised RBAC authorisation infrastructure. The prototype controller is deployed and evaluated in detecting and mitigating malicious behaviour conducted by players of the game, at real time, whereby the prototype is faced with diverse and unpredictable change.

Lastly, Chapter 7 presents the conclusions of the thesis by stating the achievements, limitations, and identification of future work.

Chapter 2

Literature Review

2.1 Introduction

Modern organisations are facing increasing security threats, where their intellectual property, systems, and resources are at risk to internal and external attacks. This had led to the use of a wide range of security controls in order to provide assurances in the fundamental aspects of security: the CIA triad of confidentiality, integrity, and availability [115].

Confidentiality refers to the prevention of unauthorised access to information and systems (e.g., a top secret document is not disclosed to the public). Integrity aims to assure information and systems remain accurate, and behave as expected (e.g., software has not been tampered with to act maliciously). Lastly, availability seeks to assure that information and systems remain available when needed (e.g., an authorised employee is not hindered in accessing a critical system).

Various controls exist for organisations to maintain assurances in confidentiality, integrity, and availability. These range from administrative procedures (e.g., definition of security policies) to technical solutions, such as physical controls (e.g., electronic door locks and alarms), authentication and access control systems, network security, application security, and encryption. As such, an organisation may employ several layers of control in order to prevent or mitigate security threats. For instance, an organisation may utilise firewalls to prevent access to their network from wide area networks, and deploy intrusion detection systems [120] to identify external attacks. Additionally, an organisation may install anti-virus tools to maintain the integrity of installed applications, and access control systems to prevent unauthorised access to confidential resources.

This research focuses on access control, with the intent on improving the management of access control in light of a particular type of security threat: insider threat. Specifically, the goal is to extend access control in order to handle prevalent insider threats that modern organisations are victim to, by introducing a greater degree of automation.

Access control, where an organisation seeks to govern user access to electronic resources (e.g., systems, hardware, and data), refers to how a user is identified, the privileges a user has, the constraints of access a user is subject to, and how access is decided. It is considered the enabling technology for users to gain access to resources, in which a user can exhibit insider threat. Insider threat refers to the potential of a user within an organisation to cause damage against that organisation, in terms of theft, sabotage, and fraud. Users who abuse their privileges to carry out insider threat can be mitigated through active management of access control (i.e., removal or assignment of access). However, current solutions in access control are limited in automating such a process, relying on manual practices to identify and respond to such threat.

A solution to automating the management of access control is self-adaptation. Self-adaptation, where a system is capable of observing and adapting itself, is considered due to the ability for a system to reason about its own state, identify non-conventional states, and adapt its state to protect, manage, or repair. Self-adaptation can be applied to existing approaches in access control to improve upon the handling of insider threat, through active adaptation of access control rules and user privileges. This enables a direct response to an ever changing and dynamic environment (i.e., the organisation, its users, and its resources).

In this chapter, a literature review explores the topics of access control and self-adaptation, along with their relation to the mitigation of insider threat. It is structured as follows. Section 2.2 provides a core overview of access control. Common access control models within industry are highlighted, as well as emerging models attaining increased interest in the security research area. Section 2.3 introduces insider threat, highlighting different types of attacks, and current approaches used for detection and mitigation. In Section 2.4, an overview of self-adaptation and self-adaptive software systems is presented, discussing prevalent self-adaptive reference models, challenges in realising self-adaptation, and an example self-adaptive solution. Section 2.5 discusses relevant work on self-protection and their application to mitigating insider threat. Lastly, in Section 2.6, a summary of the literature review is given.

2.2 Access Control

Access control is the government and restriction of access to a resource [130], where a resource could be anything from a software system (web application, database) to an electronic device (electronic door lock, mobile phone). Through well structured access control policies (rules), assignment of user access rights, and precise separation of duties, an organisation garners a certain level of protection from unwanted access. By definition, access control mitigates the potential damage from malicious insiders by limiting a user's scope of access.

Access control embodies two concepts: *identities* and *permissions*. An identity is a digital representation of a subject (a user), where a subject could be a human being, a system, or even a process [12]. An *identity* contains information about the subject, particularly relevant for authentication [110] where a subject must identify themselves (e.g., entering in a username and password, or use of biometrics [117]). Most importantly, an identity contains a set of subject access rights (i.e., privileges). Access rights represent a subject's right of access to a resource, used in conformance to a set of *permissions*. Permissions define the access right(s) required in order to obtain access to a resource. Once a subject invokes the required access right(s) to a resource, the subject is said to be authorised.

Access control models seek to classify and define how access is governed, in terms of how permissions are expressed, who can define permissions, and what an access right looks like. The Mandatory Access Control [75] (MAC) model enables subjects with a set of access rights to access resources in conformance to centrally specified permissions (i.e., defined by security administrators). In contrast, the Discretionary Access Control [75] (DAC) model enables subjects in a similar sense to MAC to access resources, yet, permissions can be specified by the subjects themselves in relation to the resources that they own. Additionally, the Bell-LaPadula model (BLP) [11] awards access based on labelled classifications, such as *Top Secret* or *Public*, and a subject's level of security clearance.

Arguably, the most adopted [102] access control model in industry is Role Based Access Control (RBAC) [97]. RBAC introduced the notion of roles, where a role is assigned a set of permissions that enable access to a resource. Finally, Attribute Based Access Control (ABAC) [152] presents a more generic view of the RBAC, where instead of roles, attributes (e.g., $\langle gender, male \rangle$) are used in order to collate and assign permissions.

The rest of this section addresses RBAC and ABAC. This includes the discussion of the eXtensible access control markup language (XACML) [101]; the role of authorisation infrastructures in implementing access control; and the notion of federation, where access control is implemented across multiple organisations.

2.2.1 Role Based Access Control

The Role Based Access Control model is the culmination of works by Ferraiolo et al. [49] and Sandhu et al. [123] that led to a National Institute of Standards and Technology (NIST) RBAC standard [97]. The RBAC standard is defined by three layers, each layer extending the layer prior with additional features. The layered approach provides flexibility for organisations to adopt the standard, whereby not all functionality may be necessary. These layers are referred to as *Core*, *Hierarchical*, and *Constrained*.

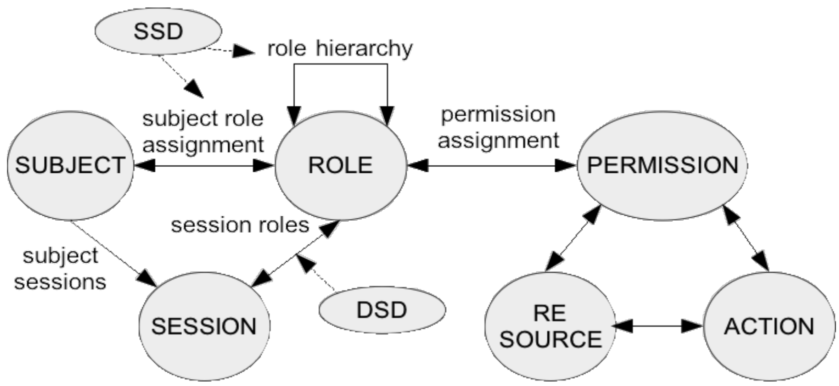


Figure 2.1: Role Based Access Control Model [97]

RBAC *Core* (Figure 2.1) defines the minimum requirements to conform to the RBAC standard. It defines the fundamental elements that must exist within an RBAC implemented access control system, namely, *subjects* (identities), *roles*, *resources*, *actions*, *permissions*, and *sessions*. *Subjects* are assigned a set of *roles*, where a *role* may define a job function within an organisation (e.g., operations manager). *Roles* are assigned *permissions*, where each *permission* details the ability for a *subject* to execute an *action* (e.g., print) on a *resource* (e.g., printer). A *subject's session* captures a set of *roles* that the *subject* has currently activated.

RBAC *Hierarchical* extends RBAC *Core* by introducing the ability for *roles* to inherit *permissions* of another *role* (role hierarchy, Figure 2.1). The RBAC standard defines inheritance by stating that a $role_1$ inherits from $role_2$ only if $role_1$ contains the same *permissions* as $role_2$.

RBAC *Constrained* extends RBAC *Core* and RBAC *Hierarchical* by introducing constraints in regards to *subjects* and *roles*. Constraints are used to prevent a *subject* from abusing business functions within their position of power. For example, a constraint may exist that prevents a *subject* who is a *student* from marking their own work, as a member of *staff*. The RBAC standard defines two types of constraints: Static Separation of Duties (*SSD*), and Dynamic Separation of Duties (*DSD*), as labelled in Figure 2.1. *SSD* refers to the restriction of subject role assignment, meaning if a *subject* owns $role_1$, they may not own $role_2$. *DSD* refers to the restriction of which *roles* a *subject* may activate within a *session*. This means, if a *subject* has activated $role_1$ but also has been assigned $role_2$, access to permissions assigned to $role_2$ are denied.

There are many proposals that extend RBAC, highlighting not only RBAC's popularity in industry, but also in the research area. Kalam et al. extend RBAC to include the notion of organisations in Organisation Based Access Control (OrBAC) [69]. Introducing *organisations* enables the specification of RBAC rules relevant to an organisation, where there are many sources of authority (SOAs) sharing access. Similar work by Demchenko et al also address the problems caused by multiple sources of authority, proposing Role Based Access Control for Distributed Multidomain Applications (RBAC-DM) [44]. Demchenko highlights limitations of RBAC in collaborative environments (containing multiple SOAs), and addresses them via the use of multi-domain authorisation sessions (where an RBAC *session* can span across several organisations). Lastly, Bertino et al.'s GEO-RBAC [13] introduces the notion of *location*, where a subject's geographical location influences the activation of a subject's assigned roles. GEO-RBAC addresses the need for spacial aware access control, where subjects may only access a resource depending on their location.

2.2.2 Attribute Based Access Control

Attribute Based Access Control (ABAC) is a recent development in access control, garnering increased attention from the research community. There are a number of proposals [58], critiques [122], and implementations [32, 93, 101]. ABAC can be considered a natural progression from the RBAC model, whereby instead of *permissions* assigned to *roles*, *permissions* are assigned to *attributes* of a subject. An attribute is defined as a tuple $\langle AttributeType, AttributeValue \rangle$, where *AttributeType* is either unique to an organisation or something commonly owned by identities (e.g., email address, gender, etc.), and *AttributeValue* represents the value of the attribute. For example, an attribute for a subject's identity with first name *John* is described as: $\langle \mathit{firstName}, \mathit{John} \rangle$.

ABAC implementations and proposals have put forward additional criteria for access control, as opposed to simply replacing the notion of roles in RBAC with attributes. Notably, environment conditions are considered in order to provide additional context to a subject's request for access. For example, a subject requesting access outside of normal office hours should not be granted access, despite having the necessary attributes to gain authorisation to the resource. The inclusion of environment conditions has the ability to expand the criteria necessary to award access, and further protect an organisation's resources from attacks (e.g., credential stealing attacks [128], by blacklisting IP addresses).

Sandu states that the leap from RBAC roles to the use of attributes offer a number of benefits [122]. This highlights the fact that ABAC unifies many access control models, such as attributes used for *roles* (RBAC), *location* (GEO-RBAC), and *security labels* (Bell-LaPadula). However, the resulting benefits of ABAC come with complexity. Organisations now have to be more specific when utilising ABAC, as access rights could be represented as anything a subject, resource, or environment might own. This has the potential to lead to conflicting access control policies, or increased challenges when managing access, due to no clear representation of an access right.

2.2.3 Extensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML) standard [101] is used to define a plethora of access control policies, primarily enabling the specification of ABAC and RBAC policies [100]. The purpose of XACML is to provide an extensible format that organisations can use for specifying access control rules, as well as a format in which access requests and decisions can be conveyed. The extensible characteristics of XACML enables integration between independent access control systems, whereby common terminology is used. In addition to the markup language, XACML promotes separation of access control from resources (i.e., a resource relies on an external service to perform access control decisions). Separating access control decisions allows for centralised management, removing redundancy in access control rules that are typically specified and interpreted locally at each resource.

XACML describes a set of conceptual components necessary to implement separation of access control from resources (Table 2.1). These components are the enabling factors of access, whereby in real systems [48, 148] that implement such components, access control can easily be monitored, and managed, in a centralised manner. This removes the need for each of an organisation's resources to implement access control, and for administrators to replicate access control rules in a redundant fashion.

Component	Description
Policy administration point (PAP)	The source of authority / system that issues access control policies
Policy decision point (PDP)	Evaluates access requests against policies to provide access decisions
Policy enforcement point (PEP)	Makes access requests and enforces access decisions
Policy information point (PIP)	Contains subject identity information (attributes)

Table 2.1: XACML components

2.2.4 Authorisation Infrastructures

An authorisation infrastructure [32] is a loose term for a collection of services and mechanisms that implement an access control model. There are a number of varying terms for authorisation infrastructures, such as the ones defined by authentication and authorisation infrastructures (AAIs) [81], XACML's reference authorisation architecture [101], and privilege management infrastructures [31]. Chadwick et al. [32] define an authorisation infrastructure as follows:

Definition 1 (Authorisation Infrastructure [32]) *"Authorisation infrastructures manage privileges and render access control decisions, allowing applications to adjust their behaviour according to the privileges allocated to users."*

Key to this definition and other variations is the existence of independent services that facilitate the management of subject access rights (*identity services*), and access control decisions (*authorisation services*). Examples of *authorisation services* include the axiomatics policy server [4], PERMIS standalone authorisation service [114] (both of which utilise the XACML standard to define access control policies), and the community authorisation service (CAS) [112]. Examples of an *identity service* include directory services, such as the Lightweight Directory Access Protocol [76] (LDAP). Other forms of identity services include credential issuing services (such as SimpleSAMLphp [131] and the Shibboleth identity provider [93]). These types of identity services not only maintain a subject's access rights (privileges), but can be configured to decide what access rights can be issued and released to given services across multiple domains.

General Model for Authorisation Infrastructures

To abstract from varying implementations of authorisation infrastructures, Figure 2.2 defines a general model of an authorisation infrastructure. Based on existing implementations [32, 81, 101], *identity services* may authenticate subjects, and maintain, assign and release a subject's access rights (i.e., privileges) to *authorisation services* based on policies (e.g., Shibboleth's attribute release policy [93]). *Authorisation services* may validate and evaluate a subject's access rights against a set of policies (e.g., PERMIS's access control and credential validation policies [114]).

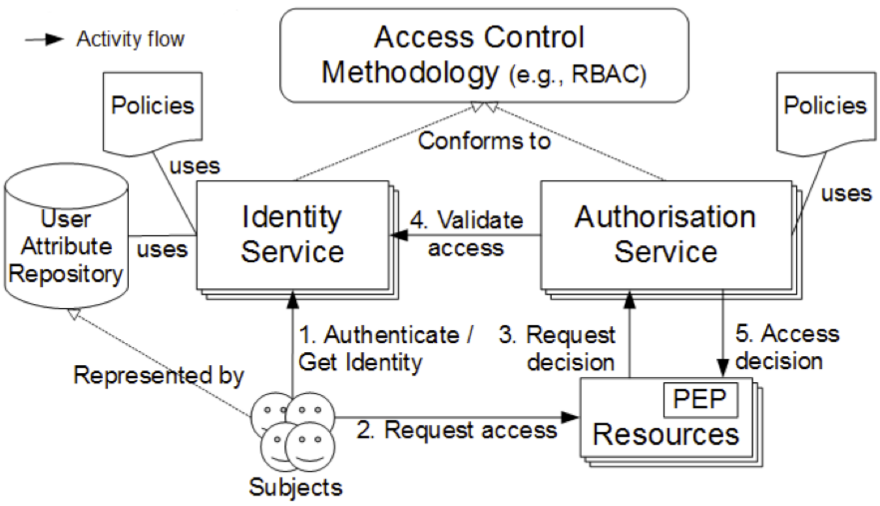


Figure 2.2: General authorisation infrastructure model

The combination of both *identity services* and *authorisation services* results in the implementation and conformance to an access control methodology (e.g., RBAC [97]). Policies within *authorisation services* are used to define the constraints of access (i.e., RBAC role permission assignments), whereas policies and subject attribute repositories (e.g., LDAP [76]) within *identity services* contain or define what subjects have in terms of assigned privileges.

Associated with policies and access rights is the notion of *source of authority* (SOA) and *issuer* [32]. An SOA is the owner of a resource that establishes the rules of access (as policies) to their resources. An *issuer* is the identity service or person that issued a subject a set of access rights, which are either stored in an attribute repository as unsigned or signed attributes [64], or are generated at time of request [99].

The key facet of authorisation infrastructures, highlighted by the general model, is the use of services to implement and provide access control, external to an organisation's resources. This implies a separation of duties between provisioning of functionality within resources, and the assessment of right to access [32, 81, 101]. Specifically, access control is implemented by services that provide the following functionality:

1. Authentication and disclosure of subject information, such as access rights and subject identifiers (*identity services*)

2. Evaluation of access rights against access control rules, and decision of access (*authorisation services*)

3. Coordination of access requests and enforcement of access control decisions (*policy enforcement points - PEPs*)

With reference to the flow of communication in Figure 2.2: to obtain authorisation, subjects (users) authenticate (1) with a given *identity service* that maintains a set of access rights for each subject. The authenticated subject can then request (2) access to a particular resource. The resource's policy enforcement point (PEP) communicates with an *authorisation service* (3), which can first validate (4) the subject's set of access rights (i.e., ensure they are legitimate), and then decide upon access. The *authorisation service* sends a response back to the *PEP* with a message indicating if authorisation should be granted or denied (5).

PERMIS Authorisation Infrastructure

An example of an authorisation infrastructure is PERMIS [32]. PERMIS is a modular based authorisation infrastructure that implements an RBAC / ABAC model. Figure 2.3 portrays one configuration of a PERMIS authorisation infrastructure, tailored for use within a single organisation. Specifically, the figure details a PERMIS standalone authorisation service interacting with a resource's policy enforcement point (PEP) and an identity service, to govern access.

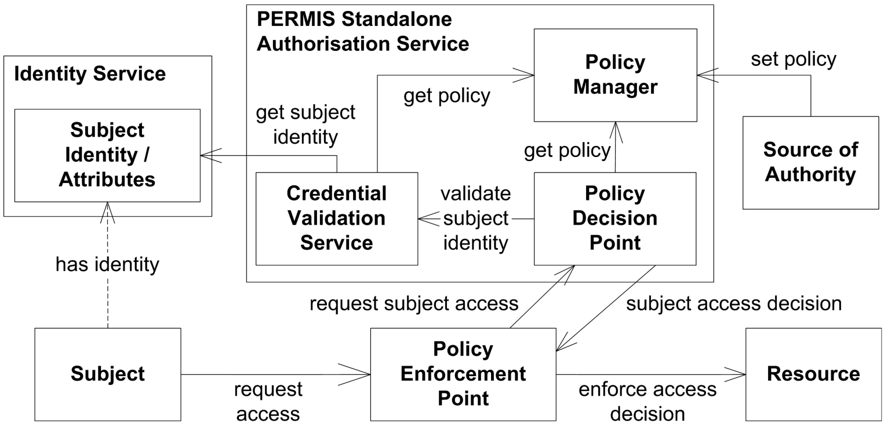


Figure 2.3: A simple example of a PERMIS authorisation infrastructure

The authorisation service's role is to analyse requests made through a resource's PEP, validate them and assess whether or not access should be granted. Access control decisions are governed by policies defined by sources of authority (SOAs). Policies are written and stored as XML files using either PERMIS's own proprietary schema, or the XACMLv2 schema. A policy contains a set of rules conforming to RBAC / ABAC, such as permission assignments, environment constraints, and role hierarchies. Subject access rights are specified either as plain text attributes or in the form of an X.509 Attribute Certificate (AC) [64], often referred to as a credential. A credential is a digitally signed access right, signed by the administrator who issued the access right to the subject, and stored within an identity service (e.g., LDAP).

Components within the PERMIS authorisation service interpret policies and subject credentials in order to generate access control decisions via its Policy Decision Point (PDP). The credential validation service (CVS) is a novel concept of PERMIS, whereby the authorisation service has the ability to validate a subject's set of credentials to ensure whoever issued the credential, is a trusted issuer. The PDP assesses the subject's set of valid attributes (once validated) to check if the subject meets all the requirements for access (defined within a PERMIS authorisation policy), thus granting / denying access.

2.2.5 Federated Identity Management

Federated identity management (or federated access) is the sharing of access across multiple management domains (organisations). Various access control models are suitable for federated access control [44, 58, 69, 139], demonstrated by several implementations [32, 93, 131]. The International Telecommunications Union (ITU) defines federated identity management [63] as follows.

Definition 2 (Federated Identity Management [63]) *"A set of functions and capabilities (e.g., administration, management and maintenance, discovery, communication exchanges, correlation and binding, policy enforcement, authentication and assertions) used for: assurance of identity information (e.g., identifiers, credentials, attributes); assurance of the identity of an entity (e.g., users / subscribers, groups); and enabling business and security applications."*

Basically, federated identity management is the management of identities across multiple organisational domains. Subjects are capable of using their identities / privileges to access resources owned by organisations other than their own (a service provider) so long as these are issued by a trusted third party organisation (an identity provider). Each service provider maintains their own set of access requirements to their resources, and identity providers maintain their subject's set of access rights (as identities) that can be used to access such resources.

Figure 2.4 shows a high level overview of a federated environment, containing a service provider (SP) and several identity providers (IdPs). Subject identities managed by an identity provider can be assigned a set of attributes that are stored within an identity service (e.g., simpleSAMLphp [131]). Subjects can use their attributes to gain access to a service provider's resources given that the service provider trusts the identity provider. To control the release of attributes, some identity providers may define attribute release policies [93] to prevent certain types of information from being released to service providers. When an identity provider releases a subject's set of attributes, it is typically in the form of a signed certificate [64] or assertion [99], indicating what attributes the subject owns, who assigned the attributes, and how long the attributes are valid for. To control access, each service provider within the federation manages their own set of permissions defined in access control policies. The service provider ultimately decides upon access through the use of authorisation services (which provide access control decisions local to the organisation).

A problem that arises from federation of access control, is that service provider organisations will likely face large and unknown user bases. Coupled with the fact that identity providers do not have to release personal or meaningful identifiable information (e.g., e-mail address, name, etc.) about their subjects (often opting to

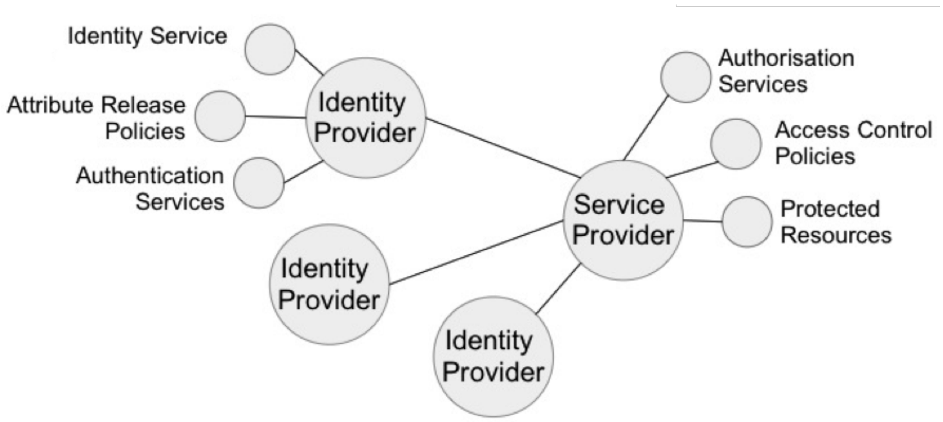


Figure 2.4: Conceptual view of federated / collaborative access

use a uniquely generated persistent (PID) or transient ID (TID) instead [131]), a service provider may find it challenging to manage access at a federated level. This requires identity providers to cooperate with service providers, where currently no efficient medium exists to facilitate management change between federated organisations.

2.3 Insider Threat

Insider threat refers to an organisation's risk of attack by their own users or employees. It is fast becoming a prominent topic that organisations need to address, as highlighted by recent scandals in the media [10, 20, 143]. The CERT Guide to Insider Threats (Cappelli et al.) [27] defines malicious insider threats as the following.

Definition 3 (Insider Threat [27]) *"A malicious insider threat is a current or former employee, contractor, or business partner who has or had authorised access to an organisation's network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organisation's information or information systems."*

A common characteristic of insider threat is that malicious insiders utilise their knowledge of their organisation's systems, and their assigned access rights, to enable attacks. This places a malicious insider in a fortuitous position, whereby the insider (as an authorised user) can cause far greater damage than an external attacker, simply due to their access rights [28]. Authentication, access control, and authorisation infrastructures provide critical security measures in enabling confidentiality, integrity, and availability to an organisation's resources. However, unless additional measures are put into place, malicious insiders can abuse these security measures, as the scope of such systems assume that if a user has authenticated, and has the required access rights, access to resources should be given.

There has been a significant amount of research in methods to detecting insider threat. These range from the use of honeypots [132] that store fake information to entice and catch out malicious insiders, to characterising insider attacks [98] based on analysis of historical attacks, as well as the development of detection tools [61] that enable the identification of abnormal activity within an organisation. Whilst there are a number of novel techniques [14, 37] that enable the detection of insider threat, there is little research that utilises such techniques within an automated solution [30, 136]. Many existing approaches require analysis by human agents to identify and execute resultant actions to mitigating attacks, and few tackle the potential abuse of access.

The rest of this section discusses the classification of insider threat, as well as current methods of insider threat, techniques in detecting and mitigating insider attacks, and related advancements in access control.

2.3.1 Classifying and Characterising Insider Threat

Capelli et al. [27] classify three forms of insider threat: *sabotage*, where malicious users attempt to damage or corrupt organisational resources, *theft*, where resources are stolen and distributed, and *fraud*, where activity is covered up or information is used to commit crimes, such as falsifying money transfers.

Characterising attacks within these classifications has led to a variety of models of insider threat, such as CERT's Management and Education of the Risk of Insider Threat (MERIT) models [26]. A set of MERIT models exists for characterising *sabotage* attacks [90], and *theft* attacks [89]. These MERIT models outline the characteristics pertaining to different classification of attacks, identifying stark differences in social and technological factors that have led to an attack occurring. For example, *sabotage* attacks were more technically sophisticated in comparison to *theft*, suggesting that insiders committing *sabotage* were advanced computer users. A contrasting example looks at the sociological factors behind an attack, such as *fraud* attacks were often committed by low-level employees over long periods of time until caught. By contrast, data *theft* is often committed by highly educated personnel (e.g., scientists, engineers) who steal information prior to resignation [27, 141].

As opposed to a characterisation model per classification of threat, Nurse et al.'s approach opts for a unifying model [79, 98]. Their approach is capable of describing all classifications of threats, presenting a usable model to facilitate detection and mitigation, to what is already a challenging problem. With reference to this model, insider attacks are clearly characterised by *catalysts*, *actor characteristics*, *attack characteristics*, and *organisation characteristics*.

Catalysts and *actor characteristics* refer to behavioural and psychological events pertaining to the malicious insider in question. For example, an actor characteristic is the insider's attitude to work (i.e., loyalty, uncaring), their personality (i.e., introvert, or share traits within the dark triad of behaviour [65]), previous historical behaviour (i.e., breaking policy, physical altercations, rule violations) and psychological state (i.e., are they disgruntled about past events). Nurse et al. claim that a mix of social and technical factors precipitates the motivation to carry out an attack, and a major tipping point to this are *catalyst* events. For example, redundancy, demotion, or moving to a competitor.

Many of these factors are difficult to identify from a technological standpoint, especially in forming an assessment of psychological state that leads to the execution of an attack. However, previous research utilises these factors in an attempt to predict the likelihood of insider threat. For example Kandias et al. [71] monitor a user's technological activity and couple this with psychometric testing to indicate stress. Whilst the approach is novel, and shows promise in terms of outlining focused monitoring strategies, psychometric testing is not always a practical or time-relevant activity to carry out. That is to say, psychometric testing can give a general awareness of an attacker's personality, but does not account for recent events that impact an attacker's mood or motivation.

Attack characteristics and *organisation characteristics* refer to technological factors. For example, an attack characteristic refers to the type of attack the insider carries out (e.g., abuse access rights, deletion of backups), which are linked to the classification of attack. Within the attack itself, there exists a set of attack steps that are indicative of the actions performed against *organisation characteristics*. For example, an attack step will target a particular organisation resource, via exploiting a vulnerability (i.e., within the resource itself, security configurations, or lack of security measures). These characteristics provide concrete evidence of an insider attack, and are essential when considering a reactive approach to detecting malicious insiders.

Lastly, various methods exist in which to describe insider attacks. Attack trees [91, 118] seek to model high level attacks that define a set of attack paths to achieve a desired goal (e.g., discover the scope of an access control policy). This approach is useful in defining high level attacks, such as network intrusions, yet is limited in the modelling of abuse of access (due to the diversity of user behaviour).

Rather than define a set of attack steps that model abuse of access, process models [17, 47] can be used to identify where abuse could occur. This can guide the deployment of relevant mitigative solutions in protecting resources, but also enable the detection of attacks. For example, Bishop et al. [17] have demonstrated the use of process models in identifying insider sabotage attacks within a system workflow. Similar approaches have also been shown to work at runtime in detecting the sabotage of executing software, whereby software infected with malware has been automatically repaired based on modelling execution norms [113].

2.3.2 Methods of Insider Threat

There are a variety of methods of insider threat that organisations are vulnerable to, some of which are identified by the 2013 Vormetric Insider Threat Report [104]. The report identifies that the abuse of privileged user access rights was seen as the forefront method of concern for organisations, which remains a focal point of this research. However, it is worth considering other methods of insider threat in order to scope what this research does not intend to address.

Theft of physical devices is a major concern for organisations, typically aligned with data theft. For example, there have been cases where a laptop or USB device has been used as a vessel to transport confidential documents outside of an organisation [54]. It is arguable that as devices become smaller, and the popularity of tablet devices rises, the ability for employees to steal, or even lose them outside of the organisation, increases.

Another method of insider threat concerns application vulnerabilities. An example of this method is an employee installing a seemingly innocent application on their workstation. The application has not been approved by the organisation's user platforms team and as such is seen as an insider threat once installed. An insider threat exists as the application may contain malicious code that compromises the integrity of the organisation's network, or carry out credential stealing attacks [128] to allow an external attacker to gain entry. This activity is often referred to as accidental or unintentional insider threat [138], where through direct or indirect action an employee creates a risk despite no intentional malice. Organisations attempt to mitigate such risks through implementing a change management control procedure [68], which guides any changes to an organisation's network, systems, and software.

A second example related to application vulnerabilities is the risk from disgruntled employees leaving back door access via in-house developed applications. This example is often used to commit sabotage [27], where an employee has created a security hole within an application to by-pass security controls (such as firewalls and access control systems). This begs the question can any application within an organisation be trusted, and is it possible to increase the trust in applications as non-malicious?

Clearly there are many approaches to address when mitigating insider threat, indicating there can be no single solution. It is therefore necessary to adopt a layered approach to provide a high coverage in mitigation of insider threat. For example, physical security is needed to mitigate theft of physical devices, change management to prevent unsolicited applications from being installed, and intrusion detection to cover the possibility of back-door vulnerabilities left by disgruntled employees. In addition, incorporating recent approaches in understanding data flow and execution in deployed applications [17, 113] can greatly improve upon the trust organisations have in their applications.

2.3.3 Detection and Mitigation

Insider threat presents a unique scope of attacks which are arguably more challenging to detect when compared to external attacks, such as network intrusion or deployment of malware. These conventional attacks are obvious in terms of 'attack steps' performed, making it less of a challenge to detect. Whilst such attacks are entirely possible of a malicious insider, they are not representative of the attacks that many organisations consider to be most vulnerable from, being the abuse of privileged access rights by the employees of an organisation [104].

As such, detection and mitigation approaches are still within their infancy in terms of application in industry. Detecting attacks, such as the abuse of access, is traditionally done through auditing of logs (e.g., access control logs, server logs, etc). This continues to be a prevalent method in detection [54]. A problem with traditional methods is that they are highly reliant on human analysis, where detected attacks may go unchallenged, or responded to in an incorrect or untimely manner. To improve upon this, a number of automated solutions exist within the scope of detection of insider attacks, in a runtime environment.

Intrusion detection systems (IDSs) were introduced as a means of identifying attacks, as well as flaws within an organisation's deployed security systems. The concept of an IDS is well known, with a number of approaches available [43]. Primarily, IDSs are concerned with identifying and notifying administrators of attacks and ongoing attacks. However, they also provide the basis to detecting malicious insiders. IDSs are useful in identifying insider attacks, but often are seen as separate to existing security systems (i.e., independent). Here, administrators must act on alerts from the IDS by instigating responses via other security systems (e.g., access control systems) to halt or prevent identified attacks.

IDSs can be categorised by signature / pattern based or anomaly based detection. Signature based detection utilises information about historical attacks (e.g., a repeated pattern observed in network traffic). For example, SNORT [120], is capable of detecting the rate at which a user accesses a node on a network, or a sequence of packets sent across a network. anomaly based detection makes use of thresholds / rules that classify activity based on a set of heuristics to identify anomalies. PAYL [144] takes a statistical approach to measure activity against a pre-defined norm, triggering alerts when the measure exceeds a given threshold. A signature based approach provides a stronger indication of an attack, as observed signatures of an attack can be matched precisely. However, the approach is limited in detecting unknown attacks, and is only as accurate as the set of known signatures available. By contrast, anomalous based detection is capable of identifying new and anomalous behaviour (that deviates from the norm). Whilst this is beneficial in detecting unknown attacks, detection is not as accurate, resulting with a greater likelihood of false positives.

A significant step to detecting the abuse of access is to consider user behaviour at an application level. Bertino et al. [14] discuss their anomaly based IDS approach based on mining database trace logs and attributing behaviour (usage) of the (RBAC protected) database to a user's role in an organisation. Doing so enables the profiling of normal behaviour per organisational role, in order to identify anomalous behaviour within each role profile. Rather than utilise pre-defined rules to classify behaviour, Bertino et al. make use of a machine learning classifier [46] to identify normal and anomalous behaviour. Here the characteristics of an activity are compared to a perception of normal behaviour for the role. A machine learning approach reduces the need for a set of complex detection rules, or fine grained security measures to mitigate insider threat. However, such an approach requires the organisation to have a clear understanding of different attack profiles associated to roles, to train classification of behaviour [86]. This in itself can be just as complex as defining a set of detection rules.

2.3.4 Mitigating Insider Threat through Access Control

As discussed previously, the potential for the abuse of privileged access rights is seen as a major vulnerability for organisations in being susceptible to insider threat [104]. This thesis positions that whilst access control seeks to provide assurances in confidentiality and availability of an organisation's resources, it also presents vulnerabilities given the fact that users can abuse their access. Traditional approaches can no longer be seen as enough to assure confidentiality of resources, as no assumption is made about how a user is utilising their access. It is therefore important to consider user behaviour alongside the traditional assignment of user privileges, prior to awarding access.

To this end, a number of recent methods extend access control away from the traditional mechanisms presented by RBAC and instead enable dynamic qualities. These dynamic access control approaches [18, 19, 66, 109] incorporate mechanisms from detection based systems, as well as expand upon the criteria of access to decide upon access.

Usage Control (UCON) [109] builds upon traditional access control models whereby obligations and conditions are used to assess a subject's usage of a resource, as part of an access decision. A novel aspect of UCON is its ability to capture a subject's state within a resource, and use this as a contributing factor within the access decision. Whilst the UCON model is sophisticated in identifying and managing a subject's usage, it only allows for a transient solution to managing insider threat. For example, a subject could invalidate usage requirements for a particular resource, but go on to access other resources despite being seen as a threat. In addition, the UCON approach to access control has the potential to become complex, with usage rules woven with traditional access control rules on a per resource basis. Lastly, whilst UCON extends access control to constrain access, its view of usage is scoped only to user access of protected resources. To build an accurate picture of insider threat, additional dimensions should be considered, such as social factors and contextual information (i.e., whether the subject is accessing the resource from an untrustworthy connection).

A step forward from usage is the inclusion of trust and reputation when generating an access control decision, via a *Trust Policy Decision Point* [19]. Here, a weighting of trust is calculated based on the usage or feedback from resources, providing additional context to a subject's usage. Serrano et al. [127] explore trust management to achieve access control. Within trust management, subjects and protected resources are given a level of trust, calculated from dimensions, such as past behaviour of the subject, the access rights they already own, the issuer of access rights, and feedback from other subjects / resource owners.

In similar works by Bistarelli et al. [18], a formal framework for trust policy negotiation is proposed. In contrast to Serrano et al., access is awarded through the reasoning of access control policies and a trust level generated from a subject's given set of credentials. An interesting aspect of this work is that Bistarelli et al. state that not all subjects will know all the required credentials for access. Therefore they propose an additional control that notifies the subject of the required credentials, providing a subject is deemed trustworthy enough. This adds an extra level of security to prevent unnecessary revealing of requirements for access, as knowledge of which could be abused by a malicious subject.

Both works attempt to ensure a better accuracy of access (and mitigating insider threat) by assessment of a level of trust / reputation of subjects. Potential applications of such methods are more favourable towards federated environments, where Serrano et al. suggest Paypal as an application domain (using trust in terms of authorising payments). However, the methods proposed demonstrate a singular authorisation technique, lacking integration with current standards (RBAC) and existing access control systems. In addition, neither address the problem caused by the occurrence of an insider attack, where the approach may fail to mitigate an insider attack. If a subject abuses their credentials you may expect, from the viewpoint of a system administrator, the credentials or access rights of a subject are removed entirely. Trust management is limited in this case, as static rules are defined that result in transient limitations of access.

Lastly, similar dynamic approaches specialise in expressing access control rules with a set of temporal constraints [66]. In this instance, access control policies contain a set of branch like rules, relevant to a set of system and environment states. Given a state that conforms to a temporal constraint or one that exhibits a particular event, access control mechanisms are constrained to a branch of relevant access control rules. This approach to enabling dynamic access control (along with the aforementioned) is defined as *dynamic policies*.

The dynamic policy approach, whilst capable in preventing access based on foreseen threats, has several limitations. Here, it is necessary for dynamic policies to contain a comprehensive set of access control rules to accommodate for all potential risk of abuse at runtime. The approach is also vulnerable in the sense that any prevention of access is bounded to a particular state, meaning that it is open to potential subterfuge (i.e., in perception of trust of a subject) and that prevention of access is transitive.

2.4 Self-Adaptation

Self-adaptive software systems (SASs) are characterised by the system's ability to reason and adapt its state automatically at runtime. The theory of self-adaptive software stems from control theory [103], where controllers observe a system, and based on observations, modify input to the system in accordance to a set of goals (e.g., ensure a system operates within an optimum range). A plethora of examples exist that extend control theory to demonstrate innovative methods that enable the classification, development, deployment, and evolution of self-adaptive systems in complex modern environments, from research roadmaps [34], to reference models [73]. de Lemos et al. define self-adaptive systems [42] as the following:

Definition 4 (Self-Adaptive Systems [42]) *"Systems that are able to modify their behaviour and / or structure in response to their perception of the environment and the system itself, and their goals."*

Key to this definition is the notion of *perception* and *adjustment* [34]. The former refers to the ability of a system to capture and analyse its own system state, and state of the environment it resides in, emphasising 'self'. The latter refers to the ability of a system to adjust its own execution or configuration, emphasising 'adaptation'. In terms of *perception*, a self-adaptive system will aim to identify *non-conventional* operation states [24], a state that warrants adaptation (e.g., a high load on a web server). *Adjustment* aims to bring a self-adaptive system out of a *non-conventional* operation state, and into a *conventional* operation state [24].

The rest of this section discusses prevalent reference models that are used to develop self-adaptive systems, and how self-adaptive systems are classified. This is followed by discussion of two key issues facing self-adaptive systems, being, the notion of 'when' to adapt (triggering adaptation), and 'what' and 'how' to adapt (decision theory). Finally, a solution to architectural-based self-adaptation (the Rainbow framework [53]) is discussed as an example of a re-usable framework for self-adaptive systems.

2.4.1 Reference Models for Self-Adaptive Systems

To guide the development of self-adaptive systems, several reference models [2, 73, 77] have been defined within the research area [42]. These reference models differ in terms of approach, yet all rely on the use of feedback loops [22, 55].

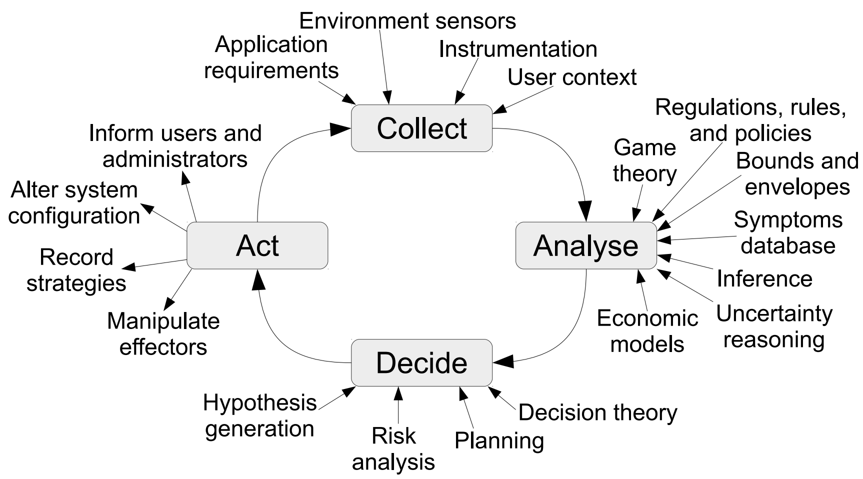


Figure 2.5: CADA generic feedback loop [22]

Dobson et al. [45] propose a generic feedback loop (extended by Brun et al. [22]), referred to as CADA - *Collect*, *Analyse*, *Decide*, *Act*. The *Collect* activity contains the collection of application requirements that govern the extent of adaptation, and collection of information to generate an awareness of system and environment state. Collected data is filtered and stored in a useable format, such as a model, in order to maintain an awareness of current and historical system states. Given a state of the system, the *Analyse* activity attempts to identify if there is need for adaptation. The *Decision* activity performs analysis of the solution domain, and planning. Once a solution is selected, planning identifies how the solution should be realised. Finally, the *Act* activity defines the realisation of an adaptation against a system, as well as informing users and administrators, and logging of adaptation. For each activity, Brun et al. propose a number of questions which should be considered when implementing feedback loops [22]. In general, the robustness and reliability of each phase of the feedback loop should be scrutinised. For example, in *Collect* and *Analyse*, it is necessary to understand the scope of information to collect, and how accurate information is. Whilst CADA provides a flexible view of key activities within a feedback loop, it negates specific information on process flow and architecture. As such, it is important to review current self-adaptive reference models.

An early reference model [107] for self-adaptation, by Oreizy et al, presented a methodology for developing self-adaptive software systems. The model considers a two tiered approach, in which a software system has an adaptation management lifecycle, and evolution management lifecycle. The evolution management lifecycle can be likened to an enabling architecture that can observe, model and enact change against implementation (i.e., the system). The adaptation management lifecycle can be likened to that of a controller, which utilises evolution management to analyse states of the system, and decide upon change. For example, adaptation management evaluates changes in the architectural model (a model of the target system). Changes trigger the need for adaptation, which is planned and deployed as a set of change descriptions to the architectural model. Evolution management reflects these changes to the architectural model against the target system, synchronising the real system with the changed model. A key part of Oreizy et al.'s reference model is the notion of consistent modelling of the target system. This ensures the self-adaptive system's perception of 'self' is up-to-date.

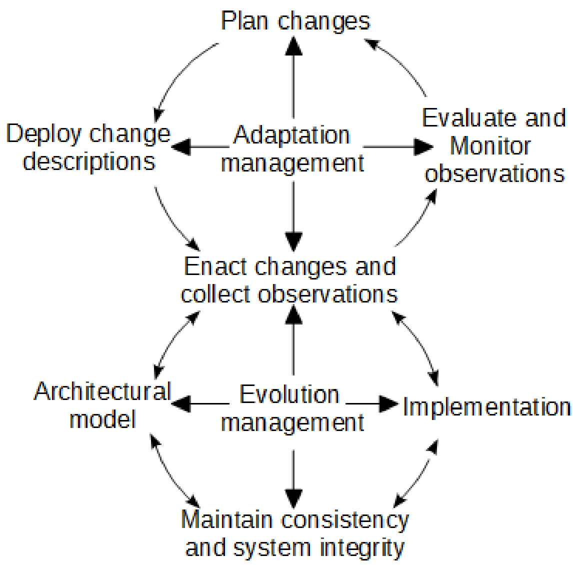


Figure 2.6: Oreizy et al. self-adaptive reference model [107]

The *Model, Analyse, Plan, Execute - Knowledge (MAPE-K) reference model*, as defined by Kephart et al. [73] follows a more traditional representation of a feedback loop (Figure 2.7). In this reference model, an autonomic manager which embodies a feedback loop observes and adapts a manageable element (i.e., a target system). Kephart et al.'s approach identifies an architecture for feedback loops within autonomic systems. It has a heavy reliance on *knowledge* gained through observation, and execution of loop activities. *Knowledge* is key to the MAPE-K reference model, and represents any information that enables the provision of self-adaptation, critical to guiding accurate adaptation and identification of non-conventional system states. The MAPE-K reference model is appealing due to its simplicity and extendable nature. For instance, an autonomic manager that incorporates the MAPE-K feedback loop could also become a managed element. The significance of this approach is that autonomic managers can be adapted, allowing for evolution of autonomic elements, and adaptation across distributed systems.

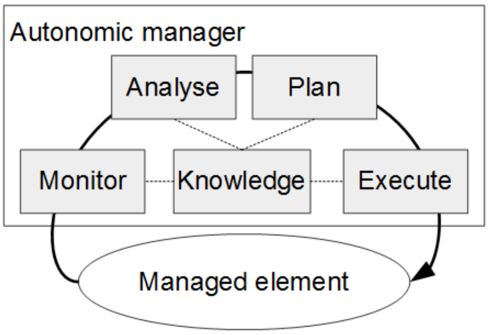


Figure 2.7: MAPE-K reference model for autonomic computing [73]

Kramer and Magee [77] propose a *three-layer reference model* (Figure 2.8) in which feedback loops enable adaptation, as well as evolution of a target system. *Component control* contains a set of components that enable the functionality of the target system. Similar to the MAPE-K reference model, this layer refers to the managed element, and sensors and actuators. *Change management* enables adaptation of components within the target system, in accordance to a set of plans. For example, should a component fail, the change management layer would observe the component's state, identify the relevant plan to resolve the failed state, and instantiate a new component. *Goal management* generates new plans in accordance to unaccounted states. In terms of the MAPE-K reference model, this is similar to employing an autonomic manager to managing a low-level autonomic manager (as a managed element). A novel concept of the three-layer reference model is the existence of feedback within each layer. Should feedback loops fail to enable the component control layer to operate as intended, the problem is escalated to the next layer for resolution.

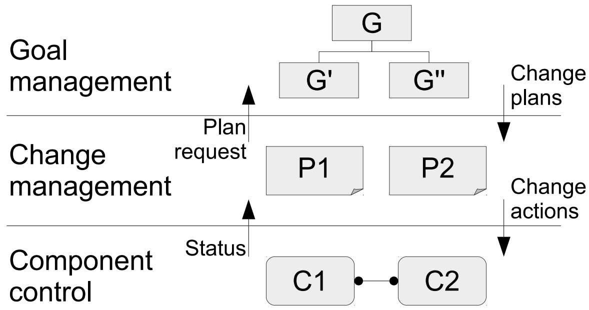


Figure 2.8: Three layer reference model for self-managing systems [77]

Andersson et al. describe a *reflective reference model* [2], focusing on abstraction of the target system (Figure 2.9), and adaptation through the use of models. The reference model is inspired by computational reflection [82], where the concept has been repurposed for self-adaptive systems. In the self-adaptive case, reflection is performed on systems as a whole (Figure 2.9). Andersson et al. state that in order for a system to adapt itself, the system must maintain a representation of itself, similar to the knowledge aspect of the MAPE-K reference model, yet with greater emphasis on defining state. To demonstrate this, the reflective reference model specifies a system with a meta- and base-level. The base-level refers to the executing system, where computation is guided by a domain model (configuration and rules), which represents the domain (real world requirements). The meta-level refers to an abstraction of the executing system (the meta-model), and enables adaptation through meta-computation (i.e., analysis and decisions that lead to adaptation). Adaptations are realised through the reflection of modelled state, as shown by the causal connection between the meta-model and base-level sub-system. The benefits of utilising a reflective approach include the ability to maintain a closed representation of the target system. The causality between the meta-model and base-level sub-system ensures that the meta-model is current, synchronising any changes that occur within the sub-system.

Finally, Weyns et al. position the FOrmal Reference Model for self-adaptation (FORMS) [146] that builds upon Andersson et al.'s reflective reference model. In general, a FORMS self-adaptive system exists within an *environment* (synonymous to any system's environment: users, external systems, etc.), and has *base-level* and *reflective* sub-systems. FORMS is defined using Z notation [133] (a standardised formal specification language). The benefit of using Z notation

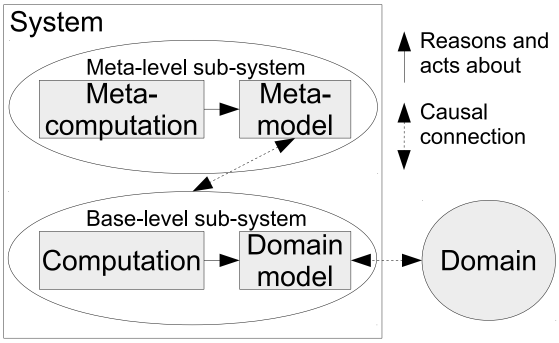


Figure 2.9: Reflective reference model for self-adaptive systems [2]

is that key dimensions of the self-adaptive system can be explicitly formalised, capturing common properties that one would expect to exist in many types of self-adaptive systems. For example, Weyns et al. formalise the self-adaptive system's environment as a collection of *processes* and *attributes*. A *process* refers to something that performs computation, yet cannot be controlled by the self-adaptive system. The result of a *process* leads to a change in *attribute* values, such as a printer's 'print' *process* that lowers the printer's 'ink' *attribute* level. Weyns et al.'s reference model is novel, due to its approach in formalising aspects and principals of a self-adaptive system. Whilst other reference models have focused on the flow of events, FORMS precisely identifies what to consider when developing a self-adaptive system.

2.4.2 Classifying Self-Adaptive Systems

Self-adaptive systems are said to contain self-* properties, which can be used as a means for general classification. In this research, four self-* properties are considered, as defined by the IBM autonomic computing initiative [56]: *self-configuration* (adapting a system's configuration of software entities), *self-healing* (detection and recovery from system faults), *self-optimisation* (managing performance of a system), and *self-protection* (detecting and recovering, or mitigating from malicious activity). There are many other references to self-* properties [73, 119], yet it is generally considered [121] that IBM's initiative provides the de-facto classifications.

In addition to classifications through self-* properties, a number of works have discussed characteristics of self-adaptive systems [1, 22, 121]. Salehie et al. characterised [121] self-adaptive systems in terms of *object to adapt* (where adaptation takes place), *realisation issues* (how adaptation is decided upon and applied), *temporal characteristics* (when adaptation occurs), and *interaction concerns* (with other systems and human operators). Alternatively, Andersson et al.'s approach [1] extends a viewpoint of characterising traditional systems, for self-adaptive systems. Here, a set of characteristics are described, beginning with *goals* (the objectives of a self-adaptive system), *change* (what triggers adaptation), *mechanisms* (how adaptation is achieved), and *effect* (the impact of adaptation).

An example characteristic regarding 'how' self-adaptation is achieved is the organisation [1, 121] of the self-adaptive system. A top-down approach, also referred to as centralised [149], is a self-adaptive system that opts for a singular controller that controls multiple system components. In contrast, a bottom-up approach opts for multiple controllers, also referred to as decentralised [149], across multiple system components that operate together to achieve self-adaptation.

Another example, relating to the 'mechanisms' of self-adaptation, is how decision making is achieved [121]. Salehie et al. state that self-adaptive systems either employ static or dynamic decision making. Static decision making refers to a process that is hard coded into the self-adaptive system, and cannot be changed during runtime. Dynamic decision making refers to a process that can change at runtime, either by the system itself, or by an external party (i.e., a human operator). In addition to these two categories, Yuan et al. [149] characterise these further by identifying theoretical foundation, such as whether decision making is based on logic, heuristics, utility [74], or machine learning [86].

Regarding what type of adaptation can occur, self-adaptive systems are characterised in terms of parametric and / or structural adaptation [1]. Parametric refers to the change of system parameters that impact the behaviour of the executing system (e.g., increasing the time interval of a sensor reading). Structural refers to the change of composition of the system (e.g., activating servers within a video streaming system).

Finally, several approaches specialise in characterising certain types of self-adaptive systems. For example, Weyns et al. [147] provide a reference model that can be used to characterise distributed self-adaptive systems. In addition, Yuan et al. [149] provide a taxonomy that can be used to classify self-protecting software systems.

2.4.3 Triggering Adaptation

Triggering adaptation refers to when a self-adaptive system is put into a non-conventional state, as a consequence of system and / or environment change. It relates to Andersson et al.'s *change* characterisation, and is of particular interest as it concerns the mechanisms that define non-conventional states (i.e., 'when' to adapt). A popular approach is *Event - Condition - Action* (ECA), as exemplified by the Rainbow [53] framework, whereby conditions (that express facets of non-conventional states) are used to evaluate consequences of changes in the system (events) that trigger adaptation (action).

Specifying and evaluating conditions of non-conventional states are exemplified in various approaches. Schneider [124] use formal proof to specify conditions that enforce security policies. Conditions are evaluated against a running system to confirm if security policies are properly enforced, and halting execution should an event indicate potential constraint violations. Rainbow [53] makes use of Stitch [36], an architecture-based adaptation language, which defines heuristic based conditions (containing quality dimensions and thresholds) that trigger the need for adaptation on a system's architecture. Bertino et al. [14] employ a data mining and machine learning approach to classify whether behaviour is normal or is malicious within the use of an RBAC configured database. The condition in this case is a machine learning classifier that classifies a user's behaviour (i.e., a collection of events invoked by a user) as malicious.

What constitutes conditions of a non-conventional operation state is often relevant to the given self-adaptive system. In the context of malicious behaviour, for a given organisation, any number of conditions can represent malicious behaviour, and thus trigger adaptation. Whilst the approaches discussed can be utilised to identify malicious behaviour, a unifying mechanism is needed to trigger adaptation.

One solution is Baracaldo and Joshi's [9] approach that exemplifies the use of a unifying mechanism to produce a level of trust. A level of trust is calculated through a multitude of conditions being met, where Baracaldo and Joshi cite several solutions, including Bertino et al.'s machine learning approach. A set of condition violations can act as input to the calculation of a user's level of trust, such as the inference of unauthorised access, or requesting access from an untrusted connection. A calculated weighting, as positioned by Baracaldo and Joshi is useful, given that a change to this weighting can trigger adaptation (as a result of multiple constraint violations), but also provide an indication to the extent of adaptation required (e.g., high impact strategies for persistent malicious users), and a measure of 'when' to adapt.

2.4.4 Decision Theory

Deciding what and how to resolve non-conventional operation states is a problem facing self-adaptive systems where there exists a multitude of strategies. It relates to Salehie et al.'s *realisation issues* characterisation. For example, an e-mail service notices abnormal login activity from an unrecognised IP address in a foreign country. The e-mail service may choose to temporarily disable login ability from all but previously trusted IP addresses, instead of removing complete access to the targeted account. Both solutions will prevent the abnormal behaviour from continuing, but given the state of the system, a self-adaptive manager must decide upon which strategy is most appropriate.

Utility theory [50] aims at identifying an optimum solution in the field of economics, and was applied by Kephard et al. [74] to self-adaptation for decision making. In their approach, utility functions are used to map multiple dimensions of system states against desirable values. A controller then executes the utility function to select adaptations enactable against the system. The goal of a utility function is to calculate the trade offs between multiple quality dimensions of a particular system state, whereby dimensions are prioritised by preferences specified by the deploying organisation.

An implementation of utility in self-adaptation is Stitch [36], a language for architecture-based self-adaptation. The Stitch language enables the specification of utility as a feature of its *Strategy Definition Language*, considering the notion of *quality dimensions*, *utility preferences*, and *impact vectors*. *Quality dimensions* define utility functions mapped to a property of a given system state (e.g., a scale from 0 to 1 that captures a weight of 'happiness' for average response time). *Utility preferences* enable the ability to prioritise particular quality dimensions over one another (e.g., response time has greater importance than system cost). Finally, *impact vectors* describe the cost of actions (adaptations) on each quality dimension.

One of the challenges involving the specification of utility preferences is the quantification of the characteristics of given states. Self-adaptive systems that are targeted towards system performance have a plethora of state characteristics that are transferable to utility (i.e., power consumption of a server, bandwidth usage, latency, processing power). However, quantifying qualitative aspects of a system state, such as ones for the response to malicious behaviour can be problematic. One option is mapping qualitative aspects of a system state (e.g., the criticality of a system component) to a monetary cost, via cost sensitive modelling.

Cost sensitive modelling refers to the assessment of monetized trade offs. A relevant example of this work is the framework proposed by Strasburg et al. [136], where cost sensitive modelling is used to automatically select responses to intrusion detection. Here, a self-adaptive action is synonymous to a 'response' (e.g., counter attack the source address) as a result of detecting an intrusion. Each response is assigned a cost, which is calculated through the cost of performing the response (e.g., processor usage), plus a level of impact of response on the system (negative effect of response on the system), minus the response goodness (intrusions that response can address). The response cost can then be used to rank responses to a given intrusion.

2.4.5 Rainbow Framework

The Rainbow framework [53] is a solution to architecture-based self-adaptation. It is described as an example of a reusable framework for self-adaptive systems that provides relevant learning outcomes to the research contributed by this thesis. Architecture based self-adaptation is the structural adaptation of systems, such as the removal or addition of system components. Adaptation occurs as a result of changes to resources, user requirements, and system faults, characterised in terms of event - condition - action (ECA). Rainbow uses architectural models of the system at runtime, modelling a target system as a group of related components, connections, and properties. The state of the architectural model is constantly updated through the use of probes (sensors) within the target system, and assessed against a set of constraints to identify non-conventional operational states. Upon identifying a non-conventional operation state, the Rainbow framework adapts the target system through structural changes (via effectors), to achieve conventional states.

Rainbow promotes the use of external controllers to enable self-adaptation of entire systems. This promotion of separation of duties (between system and adaptation) enables the integration with legacy based systems with little to no modification of existing system code.

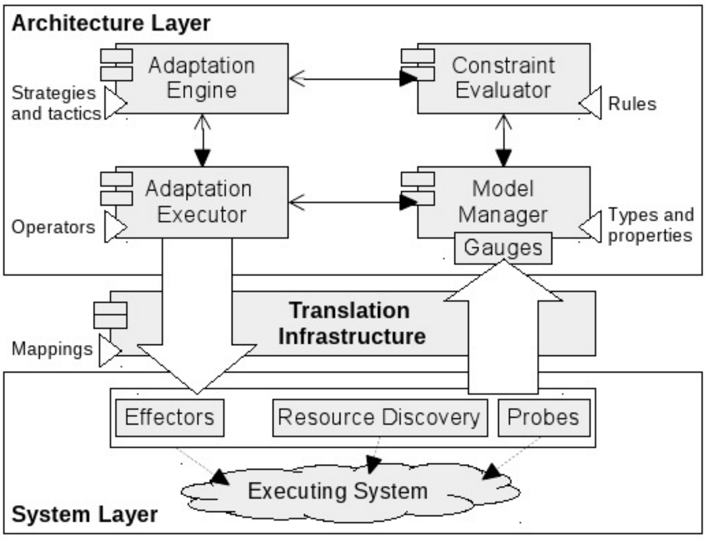


Figure 2.10: Rainbow framework [53]

Rainbow achieves its goals through its three-layer approach presented in Figure 2.10. The *system layer* refers to the target system to be controlled, and complimentary services, such as probes (that allow observation of the current state of the system), effectors (that allow for control over the system), and resource discovery (that allows for the discovery of new types of resources available to the system, i.e., new components).

The *architecture layer* enables the self-* properties of the target system, and is comparable to an autonomic manager in the MAPE-K reference model. In this layer, a feedback loop is executed. Here, the runtime system is modelled as an architectural model. It uses the input of gauges that have aggregated data from probes in the target system to identify current state (i.e., current throughput, latency of system components). The modelled architecture is then checked against constraints (e.g., violations in allowable latency of a system component), which then triggers the enactment of adaptation via the adaptation engine.

Lastly, the *translation infrastructure* is key to enabling portability to Rainbow, as each target system may have its own bespoke view of the abstract concepts Rainbow models within the *architecture layer*. For this reason, the *translation infrastructure* is capable of using a mapping that converts from system specific types, into the abstract types that Rainbow understands. This allows for unrestricted deployment within many diverse systems.

2.5 Self-Protection

Many of the approaches that can be applied to the mitigation of abuse of access, such as the ones discussed in Section 2.3.4, offer some form of dynamic or adaptive behaviour. Some of these approaches can be classified as a self-protective system (a specialisation of self-adaptive systems), where self-adaptation is used to mitigate malicious activity. As such, it is important to distinguish what is meant by dynamic / adaptive (i.e., protective), and what is meant by self-adaptive (i.e., self-protective), in the context of security.

The difference between the two is exemplified clearly by Yuan et al. [149], where they provide an example of an authentication algorithm that periodically changes its pass key, as adaptive. This is in contrast to a system that periodically changes the authentication algorithm it uses at runtime, as self-adaptive. The former is adaptive since no change occurs within the system itself, such as is the case with dynamic access control policies. The latter is self-adaptive, as the system changes itself based on perception of its own state (i.e., changes its execution). Yuan et al. define self-protecting systems as follows:

Definition 5 (Self-Protecting System [149]) *"Self-protecting software systems are a class of autonomic systems capable of detecting and mitigating security threats at runtime."*

There are various self-protective solutions that seek to detect and mitigate malicious behaviour. Self-protection in access control is one such solution, yet few approaches exist that are able to concretely address self-adaptation with a view to mitigate the abuse of access. Other solutions include self-protection at the network layer (e.g., intrusion response systems [96, 134, 136]), as well as self-protection in terms of architecture [151] (e.g., available servers, components). These approaches are exemplified and discussed in the following.

2.5.1 Intrusion Response

Intrusion response systems (IRSs) are an established method of identifying and responding to malicious activity, although they do not explicitly class themselve's as self-adaptive systems. A sub class of intrusion response systems, known as automated intrusion response systems (AIRS) [96], can be considered to be self-protecting systems as they share activities that map closely to the CADA feedback loop. Additionally, there is evidence of similar characterisations, such as differences between static [96] and dynamic decision making [136] (referred to as non-adaptive, and adaptive decision making), temporal characteristics (i.e., reactive and proactive), and decision theory.

Mu et al. [96] define a static decision model applicable to automated intrusion response; it is based on a hierarchical planning of responses (adaptations), which are assigned to risk thresholds. The target system (the network that the IRS protects) exhibits a calculated level of risk at runtime, which warrants a necessary response when intrusions are detected, in order to repel the detected attack. Examples of adaptations include adding rules to firewalls that block IP addresses to particular ports, or removing connectivity to a host. An interesting aspect of Mu et al.'s approach is the discussion of applicable goals [30] within IRS systems: *Analyse attacks* seek to perform responses to further evaluate the attack identified. *Capture the attack* aims to put a halt to the attack by blocking entry to a network for a particular IP. Finally, *maximise confidentiality* proposes to tighten security (e.g., through stricter firewall rules).

Other works take intrusion response further, specifically in reference to the decision making methodologies, which introduce the notion of adaptive decision making. Stakhanova et al. [134] propose a dynamic decision making model in achieving the selection of response to an identified attack. Here, the approach analyses system state post execution of a response, to assess the effectiveness. The success or failure of a response is then factored in to future decisions.

The advantage of an IRS is the ability to respond to unauthorised and external attacks, where many adaptations involve changes to architecture (structural) or firewall rules (parametric). Whilst an IRS is well positioned to mitigating external malicious behaviour, it is limited in mitigating attacks related to the abuse of access. These internal attacks offer different traits to that of external attackers, where arguably, there is a greater challenge in detecting a malicious authorised user, over that of an unauthorised user. In addition, IRS solutions lack specific mechanisms to perform relevant adaptation (i.e., generation and deployment of user privileges and complex policies) in the context of access control.

2.5.2 Architectural-Based Self-Protection

Architectural-based self-protection (ABSP) [151] presents a general solution to detection and mitigation of security threats, via runtime structural adaptation.

ASBP utilises an architectural model of the running system used to identify the extent of impact of identified attacks. Once attacks or security threats have been assessed, a self-adaptive architectural manager (Rainbow [53]) is utilised to perform adaptations to mitigate the attack. One example adaptation the approach offers is to throttle network connections to a server, in order to disrupt ongoing attacks. Another example is the deployment of *application guards* where a protective wrapper is deployed around architectural components (e.g., a web server). These provide mitigation measures that improve upon the integrity of architectural components (e.g., the encryption of session IDs susceptible to hijacking).

Another similar example of self-protection is one proposed by Morin et al., which takes a novel approach in managing access control, through the use of architectural adaptation [94]. An access control policy, defined by a security expert, is used to generate an architectural model (which reflects the properties of the access control policy), and synchronised against the running system. Morin et al.'s approach shows the effective deployment of access control across an entire system, where unlike a top down approach proposed by XACML, there is no centralised point of failure. A limitation in this approach is that this form of architectural adaptation is expensive, requiring all resources that need access control to be engineered in a particular manner, lowering the usefulness of the approach in industry. In addition, it lacks the ability to automatically evolve and reflect changes to access control, once malicious behaviour has occurred. However, the technique poses a novel and viable means of realising a change to access control, once such a change has been formulated.

In terms of applying an architectural based approach to detecting and mitigating insider attacks, this is a complimentary but incomplete solution. Whilst architectural based self-protection can deploy additional security components (e.g., the application guard - to aid in threat detection), it is unable to employ fine grained parametric adaptation in relation to access control. Moreover, a perception of the state of access is critical to performing necessary changes to mitigate malicious behaviour, which is challenging through the use of architectural models alone.

2.5.3 Self-Protection in Access Control

There are a variety of methods that seek to realise self-protection within access control. One example in tackling insider threat, is through risk-adaptive access control [83], as exemplified by Baracaldo and Joshi [9], Kandala et al. [70], and

Cheng et al. [35]. Risk adaptive access control (RADac) seeks to determine (at runtime) a risk to protected resources, calculated by internal and external factors (e.g., a national terrorist threat level). Based on the given risk, the level of access for a subject dynamically changes to reflect the risk on resource(s). For example, a government data system is under a distributed denial of service attack. Coupled with a national security threat level, a subject that previously could read, write, and delete from the government data system, can now only read against the system. This approach can be considered 'self-protective' as a system's perception of its environment (risk models) leads to a change (adaptation) in the way access is decided upon (albeit representative of dynamic policies, as opposed to the adaptation of policies).

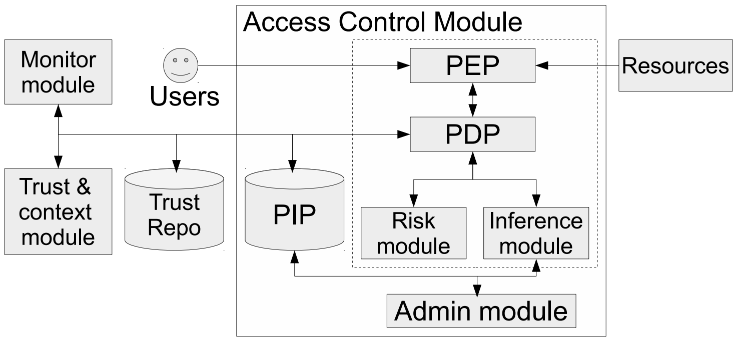


Figure 2.11: Adaptive risk management and access control architecture [9]

In particular, Baracaldo's and Joshi's framework (Figure 2.11) combines qualities of RADac and TrustPDPs [19] to mitigate insider threats. This approach extends components standardised by XACML (Table 2.1), so that access is governed on user access rights and trust (where a trust value is calculated from user activity). Access control policies are woven with trust thresholds, stating a level of trust required when a user requests a permission. The policy decision point (PDP) utilises trust data stored within the trust repository to decide upon access, whereby a risk module confirms if the user has the necessary level of trust. Risk in this context is associated to roles and permissions, calculated by a cost to the organisation multiplied by the probability of occurrence. Whilst a cost value is a viable measure an organisation could provide, a viable measure of probability is far more unlikely. For example, figures of probability (such as 0.1%) are defined without any evidence in being a meaningful value. In terms of adaptation, should a user's trust be identified as too low, the privileges of the user become limited.

Baracaldo claim that this demonstrates that the framework is capable in adapting to insider threats, but it is not clear if a user could potentially improve upon their trust (thus continuing an attack), or if the user's access rights are concretely removed.

Pasquale et al. define SecuriTAS [111] as a form of self-protection via access control. SecuriTAS aims to enable dynamic decisions in awarding access, based on a perceived state of the system (and environment). It furthers the concepts in RADac to include a notion of utility, whereby given a perceived state of the system, the optimum set of security controls are used. This is achieved through the use of a MAPE-K controller, which updates and analyses a set of models (defining system objectives and vulnerabilities, threats to the system, and importance of resources in terms of a cost value) at runtime. A novel aspect of this work is that it is positioned towards physical security, whereby a resource (e.g., a computer terminal) is stored within an office. SecuriTAS may change the conditions of access to the office based on the presence of high cost resources, or the presence of highly authorised staff. Pasquale et al. provide an example of a professor detected within an office, reducing the perceived level of threat against a set of physical resources. If the professor is present, a user with the role of student can access the office. However, if the professor is not present, the user is unable to access.

An advantage of SecuriTAS is that it is pre-emptive in mitigating potential insider attacks, by employing its own adaptive approach to access control. This is an added benefit in comparison to static access control methods (e.g., RBAC), due to the validity of potential threats increasing the level of access required for a resource. This is similar to a government organisation strengthening access given a significant national security threat (e.g., from terrorism). However, whilst this reduces the ability of a malicious user to perform an attack, it may be seen as restrictive. For instance, it may prevent legitimate users from gaining access who have never and may never perform malicious behaviour. Lastly, malicious behaviour can still occur (as in any access control model), as accessing users could abuse their access rights where no threat was previously foreseen (e.g., the professor abusing his access within the aforementioned example). In order to address malicious behaviour that does occur, a fine grained reactive response is required as a result of observing malicious acts.

2.6 Summary

In summary, this chapter has presented an overview of core background and state of the art in some topics that are relevant to this thesis, namely: *access control*, *insider threat*, *self-adaptation*, and lastly *self-protection*. Motivation for this research has been provided in terms of discussing access control and its relevance in protecting organisational resources. Insider threat has been discussed in relation to the detection and mitigation of abuse of access, whereby authorised users, with access to protected resources, perform malicious behaviour. The discussion follows a viewpoint that handling insider threat is intrinsic to access control, and how access control is managed. Through active management of access control, insider threat can be mitigated by way of changing the conditions of access, as demonstrated through solutions that seek to incorporate the notion of user behaviour when granting access.

The notion of self-adaptive systems is discussed, providing a review of recent reference models, classification dimensions, and challenges that face self-adaptive systems. This was further refined into focusing on self-protective systems, a specialisation of self-adaptive systems appropriate to the detection and mitigation of malicious behaviour. Finally, in terms of applying self-adaptive techniques to handling insider threat, this research follows the definition of a self-protective system, whereby self-protection is achieved via the parametric self-adaptation of access control.

As a first step towards applying self-adaptation to handling insider threat, the following chapter proposes the Self-Adaptive Authorisation Framework (SAAF). The framework identifies how self-adaptation is defined in terms automating the management of access control, discussing the notion of autonomic controllers (that perform adaptation), models (critical in enabling analysis of 'self'), the target system (i.e., the authorisation infrastructure), and the environment. A framework is required as existing solutions are limited in achieving self-adaptation of access control, specifically, where access control is implemented via authorisation infrastructures. The framework is discussed at a conceptual level, in order to highlight the challenges, benefits, and limitations of self-adaptation towards handling insider threat.

Chapter 3

Self-Adaptive Authorisation Framework

3.1 Introduction

The detection and mitigation of insider threat, attributed to the abuse of access in an organisation's resources, is often not at the forefront of concern. Traditionally, organisations rely on human administrators to identify the need for change to mitigate malicious behaviour, such as through the analysis of audit trails [62]. Assuming users will act appropriately within their access rights is an increasingly risky assumption to make, especially relevant as organisations have begun to work together and federate access to their resources. This only serves to make the identification and enactment of changes to access control a difficult task.

The immediacy in identifying and responding to abuse is necessary to mitigating and handling insider threat. Whilst solutions [14, 79, 144] exist to detect malicious user activity, few automated mechanisms exist that both identify, and actively respond. An automated response is essential to realising a timely solution to what could be a persistent high severity attack, which can ultimately be solved through the introduction of self-adaptation.

As such, the contribution of this chapter is the proposal of the Self-Adaptive Authorisation Framework (SAAF); a framework to enable the automated management of authorisation infrastructures (i.e., access control).

This framework is exemplified in terms of the automated detection and mitigation of insider attacks. It enhances authorisation infrastructures through provisioning an extra layer of security that achieves self-protection of an organisation's resources. The novelty of the proposed approach is that the framework presents a means to enable the benefits of dynamic access control in existing authorisation infrastructures, whilst promoting separation of concerns between the decision for access, and the decision for a management change. In addition, the framework is extendable to different access control models and diverse implementations of authorisation infrastructures.

The framework is related to dynamic access control [9, 12, 66, 109], which shares similar goals in managing access. However, the framework adopts a distinctively different approach, based on self-adaptation, where there are several overarching benefits. For example, it promotes concrete (persistent) changes to authorisation infrastructures to mitigate and further prevent attacks, as opposed to transient solutions when deciding upon access.

The rest of this chapter is organised as follows. In Section 3.2 the objectives, motivation, and conceptual design of the Self-Adaptive Authorisation Framework (SAAF) are described. Section 3.3 outlines SAAF's target system and environment, notably what can be observed and controlled. Section 3.4 describes the conceptual stages of SAAF's adaptation process, in detecting and analysing malicious behaviour to mitigate abuse of access. Section 3.5 classifies SAAF's approach in terms of a self-protecting system to enable comparison of SAAF with related work. Section 3.6 identifies the evaluation strategy in which to determine the feasibility of SAAF. Section 3.7 compares SAAF with related work, outlining the benefits and limitations. Finally, in Section 3.8, a summary of the chapter is provided.

3.2 Self-Adaptive Authorisation Framework

The Self-Adaptive Authorisation Framework (SAAF) defines an approach in which self-adaptation can be realised in authorisation infrastructures. It is designed to integrate with existing technologies that enable access control, such as PERMIS [32], Shibboleth [93], and XACML [101], as opposed to designing new technologies that have been developed with self-adaptation in mind. The reason for this is to ensure clear separation of concerns (between access control and self-adaptation), re-use of popular (and proven) technologies that are prevalent in modern organisations, and to allow organisations to integrate self-adaptation with their existing systems. To this end, the major characteristic in this thesis's approach is not to create a new dynamic access control methodology, but to make existing proven access control methodologies self-adaptable at runtime.

The following describes the objectives of the framework, a discussion on why a self-adaptive approach is taken to achieve these objectives, and then a description of the framework's conceptual design.

3.2.1 Objectives

The objective of SAAF is to enable the autonomous monitoring and analysis of state of an authorisation infrastructure, make judgements on the behaviour exhibited through *target system* and *environment* change, and adapt the authorisation infrastructure to self-protect against insider attacks. Moreover, SAAF seeks to achieve the following:

- The automated management of authorisation infrastructures, exemplified through detection and mitigation of insider attacks;
- The achievement of automation through complete separation of the decision of access (access control), and the management of access control (adaptation);
- The specification of a reusable process that can mitigate abuse of access rights (i.e., insider attacks) through the runtime parametric adaptation of an authorisation infrastructure;
- A reusable approach to enable automated management of access control in existing legacy based technologies, complementing existing standards.

SAAF's objectives are aligned with the confidentiality and integrity properties of the CIA triad [115]. SAAF maintains confidentiality as it seeks to prevent access to protected resources once a user has exhibited malicious behaviour. Essentially, a user who becomes malicious can no longer guarantee their activity is legitimate, therefore it must be assumed that any further access to resources breaks confidentiality. In addition, SAAF maintains integrity as it seeks to prevent future access to resources once a user has been identified as malicious. This prevents a malicious user from further compromising or sabotaging resources yet to be attacked.

3.2.2 Motivation for Self-Adaptation

A general problem facing authorisation is that once the criteria for access have been defined (i.e., through access control policies and issuing of access rights), the process of awarding access remains largely static [58, 97]. Authorisation infrastructures that seek to govern access to subjects follow the basis that if a subject meets the criteria for access, then the subject should be authorised. Few approaches address the fact that despite a subject fulfilling all the necessary requirements for access, they may abuse their pending authorisation.

In July 2010, it is alleged that a US army intelligence analyst downloaded a quarter of a million classified documents from the US Department of Defence [20]. The analyst claims to have had unprecedented access to classified networks for over a period of 8 months, allowing the theft of documents and the ultimate release to the general public. It can be said that the analyst used their legitimate access to abuse their position and perform an insider attack. Assuming an access control system or authorisation infrastructure was in place to protect these resources (classified documents), a traditional access control methodology would not have been enough in identifying or responding to the analyst's abuse.

As discussed in the literature review, within the last decade a push has been made towards dynamic access control methodologies. These approaches build upon awarding access based on traditional static criteria, by introducing a perception of usage [109], risk [9], threat [111], and trust [12, 127], as part of the criteria for access. Taking into account these perceptions, authorisation awarded to a subject can fluctuate based on the actions of the subject, or events within the environment (e.g., national security level being raised in threat of terrorism). These approaches seek to restrain access in light of abuse scenarios, such as the data theft example previously described. However, they face several limitations, which are addressed by this thesis's self-adaptive approach.

Separation of Concerns

Dynamic approaches to access control [9, 70, 127] consider mutable input in deciding access governed by immutable definitions for access (e.g., a subject's historic usage and access rights evaluated against an access control policy). The dynamic aspect of these approaches is akin to a human administrator temporarily preventing subject access, despite the subject having the necessary access rights. In essence, not only is there a requirement to specify authorisation in terms of an organisational viewpoint (i.e., roles, attributes, permissions), but there is a requirement to weave a perception of malicious behaviour. The downside to this is the resulting complexity in defining criteria for access, and an increase in the likelihood of redundant or conflicting access control rules.

In contrast, self-adaptation promotes separation between functionality of a target system (i.e., the process in awarding authorisation), and the functionality that achieves self-adaptation (i.e., the dynamic aspect in deciding whether or not a subject meets the criteria for access). Such an approach enables the clear separation between definition of criteria for access, and the definition of perception of behaviour, whereby authorisation and behaviour are related, but exist independently to each other. This has the benefit in allowing a broader definition of behaviour outside the scope of authorisation, whilst removing potential conflicts in which organisational roles (for example) should have access to which resource.

Persistence in Control

Dynamic access control is typical in governing access based on transient solutions. For example, in trust management [127] a subject is awarded access based on their given activity, reputation, and a perception of trust at the time of request. Each time the subject requests access, the approach in place re-evaluates the perception of activity and trust to decide upon whether or not access should be given. It is arguable that if a subject has presented a low trust in the past, due to abuse of access, re-evaluating this perception of trust per request is redundant (as the subject has already shown themselves to be malicious). In addition, by making a temporary decision in preventing access, these approaches are at risk of subterfuge. A subject may change their behaviour to reflect an authorisation infrastructure's perception of trust, in order to improve their trust in which to resume access.

Self-adaptation promotes control in this sense, but is capable of both transient and persistent solutions. Transient solutions, as demonstrated by dynamic access control approaches, may relate to the adaptation of the runtime execution of an authorisation service, such as the required criteria of access at time of access request. This creates an overhead in generating an access control decision. For example, in risk-based approaches [35], an access decision must consider a calculation of risk, subject behaviour, and the subject access rights at time of request.

Alternatively, a persistent solution refers to adaptation of authorisation assets that implement both the criteria of access (i.e., access control policies), and the conformance to access (i.e., subject access rights). Adaptation via this approach removes uncertainty from future access decisions, meaning that once a subject has been prevented in access (by removal of access rights), access is permanently removed. In addition, a persistent solution reduces the possibility of abuse where access can be awarded by multiple systems. For example, in a federated authorisation infrastructure, a subject may interact with many access control systems. Assuming each access control system implements a dynamic access control methodology, each one maintains its own perception of trust, resulting in vulnerabilities where a subject may be rejected by one system, but gain access via another.

Re-use of Existing Standards and Technologies

Solutions via dynamic access control purport their own access control methodologies, requiring bespoke implementations to achieve access control [9]. From an organisational perspective, migration to such approaches require a dramatic change in governing access. This includes the deployment of bespoke access control systems, and moving away from existing standards [97].

Self-adaptation allows for an approach whereby existing standards and technologies can become dynamic. It does not require an organisation to change their method of access control, or deploy new systems to utilise a bespoke methodology. For example, with a solution utilising the MAPE-K reference model [73], an organisation's authorisation infrastructure becomes a manageable element. A controller can be deployed to manage such an element given a set of goals (i.e., mitigate malicious behaviour). A benefit of this is a controller can abide by the authorisation infrastructure's implemented access control methodology.

Extendibility

Lastly, dynamic access control approaches are limited solely towards performing adaptive decisions on access at time of request. Whilst these approaches offer transient solutions to the problem of misuse of access rights, they are not extendible to handle wider aspects of the management of access control (by design). It is here that a self-adaptive approach can demonstrate its flexibility and extendibility.

For example, self-adaptation in this thesis's approach promotes the persistent adaptation of authorisation assets to mitigate malicious behaviour. A solution to enabling this requires the definition and implementation of mechanisms to achieve manipulation of such assets, from interfacing with a target system, to observation and decision making. Given the existence of these mechanisms, a self-adaptive approach can be extended to focus on other aspects of management of access control, beyond mitigating malicious behaviour. This includes the potential for automated policy repair in authorisation services, creation of access rights for new subjects, and evolution to natural change in organisations (e.g., specification of access for newly deployed resources, and removal of access rights from redundant employees).

3.2.3 SAAF Conceptual Design

The Self-Adaptive Authorisation Framework (SAAF) considers three key concepts of a self-adaptive system, being: the *controller* (i.e., to enable self-adaptation), the *target system* (i.e., an authorisation infrastructure), and the *environment* (i.e., everything outside of the authorisation infrastructure).

Figure 3.1 portrays a conceptual model to achieve self-adaptation within authorisation infrastructures, highlighting the separation of concerns between the *target system* and *controller*, and what exists within the *environment*.

The *target system* consists of an authorisation infrastructure, all of its subsystems, and authorisation assets. Subsystems exist as self-contained services that fulfil the role of identity services and authorisation services in order to achieve access control. An authorisation asset is characterised as either the digital representations of a subject's identity, including a representation of the subject's assigned access, or the specification of criteria for access (e.g., an access control policy). Authorisation assets are viewed as the parameters of services within the authorisation infrastructure. The emergent property of the combination of such services and assets is the conformance to an access control methodology (e.g., RBAC).

The *environment* contains everything that can interact with the authorisation infrastructure. This includes the users (subjects) of the authorisation infrastructure, protected resources, and any external systems that infer behavioural characteristics in regards to users and protected resources (e.g., firewalls, intrusion detection systems).

The *SAAF controller* is implemented using a feedback loop based on the Monitor-Analyse-Plan-Execute-Knowledge (MAPE-K) reference model [73]. Monitoring processes data from probes that report system and environment

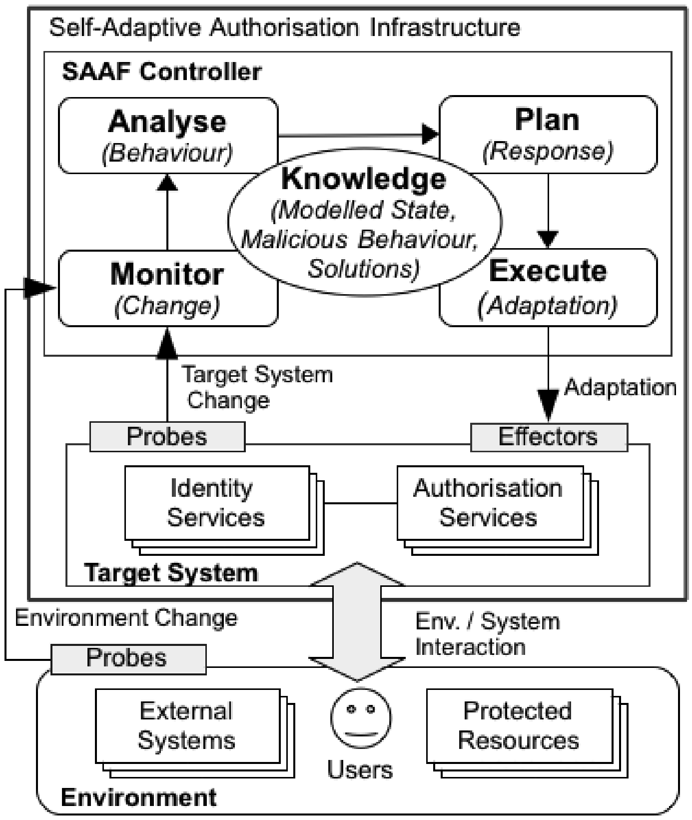


Figure 3.1: SAAF conceptual design

change (to capture operational state). Analysis assesses state to identify the presence of malicious behaviour, leading to the identification of solutions capable in mitigation. Planning selects an appropriate solution and generates a step-by-step plan in which a solution can be enacted against the authorisation infrastructure. Execution realises a plan against the *target system* whereby operations are instructed through system effectors (that perform change). This results in the adaptation of authorisation assets (the runtime parameters) within the authorisation infrastructure. Ultimately, control is achieved through manipulation of these assets, in order to change how access is awarded for future requests.

Each of the four stages require knowledge, whereby knowledge includes a model of access (the criteria and assignment of access), a model of behaviour (present and historical subject behaviour), perception of malicious behaviour (rules that define the conditions for malicious behaviour), and enactable solutions (for mitigation).

Lastly, a given implementation of the *SAAF controller* is bound to a single access control methodology. For example, a controller can adapt an ABAC [58] based authorisation infrastructures, but is not capable of monitoring or controlling a MAC [75] based authorisation infrastructure. This is due to the intrinsic nature of identifying the need for adaptation within a particular access control methodology, and identifying how to respond to such need (i.e., each methodology exhibits different processes as to how access is achieved and can be controlled). The desired result is that a *SAAF controller* bound to a specific access control methodology should be capable in controlling any authorisation infrastructure that conforms to that methodology.

3.2.4 Scope of Attacks

Figure 3.2 identifies the scope of attacks possible within a self-adaptive authorisation infrastructure. The figure conveys a feedback loop depicting a subject requesting r and gaining access d to a protected resource, as well as a feedback loop depicting a controller observing probe information p and performing adaptations via effectors e. In addition, potential attacks ($A1$ to $A12$) are envisioned in regards to the two feedback loops.

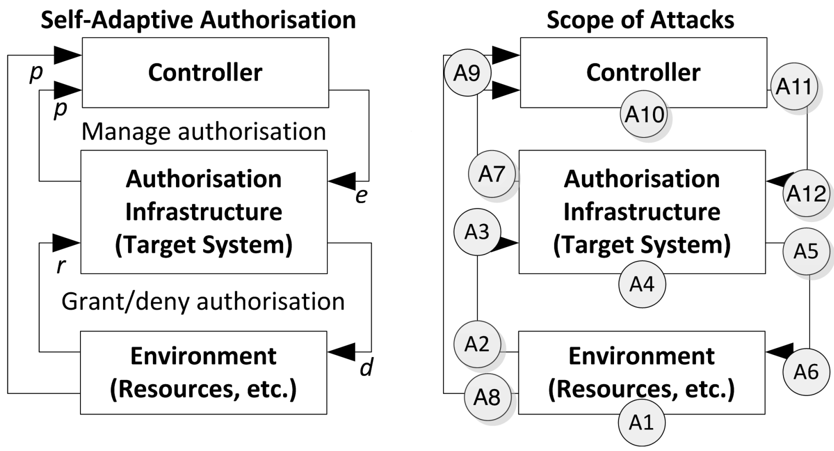


Figure 3.2: Attack points on self-adaptive authorisation

The possibility for attacks already exist within the traditional authorisation infrastructure. Here, a subject can abuse their access in relation to resources $A1$; provide false information $A2$; disrupt the running of the authorisation infrastructure $A3$; attack the authorisation infrastructure itself $A4$ (e.g., obtain additional access); spoof authorisation decisions $A5$; or attempt to disrupt resources $A6$.

SAAF is only concerned with mitigating attacks against an organisation's resources ($A1$). The scope of mitigation is constrained to automatically halting abuse and protecting against reoccurrence, related to data theft, sabotage, and fraud. However, it does not seek to repair the damage of attacks once identified, such as the restoration of a database sabotaged by a malicious subject.

Few works reflect on the potential attacks that self-adaptation may enable [149]. However, it has been considered by Cardenas et. al. in the area of cyber-physical systems [29], which shares similarities to self-adaptive software systems. As such, it is recognised that the introduction of a controller has the potential to enable attacks, including, falsifying probe data $A7$ and $A8$ in order to hide malicious activity; disrupting the service of a controller $A9$ to prevent detection and adaptation; gaining control of the controller $A10$; spoofing adaptation $A11$ to gain proxy control of an authorisation infrastructure; or disabling the service of effectors $A12$ thus preventing automated control. Whilst these forms of attack present interesting paradigms, this is considered as an important aspect of future work.

3.3 Target System and Environment

The scope of SAAF's *target system* is aligned with the general authorisation infrastructure model presented in Chapter 2.2.4. With reference to this general model and SAAF's environment, this section identifies the assumptions on an authorisation infrastructure, and concerns relating to observation and control.

Assumptions on Authorisation Infrastructures

In order for an instantiation of the target system (i.e., an authorisation infrastructure) to be made self-adaptable, the following assumptions are adhered to:

- Services of the authorisation infrastructure are capable of generating logs of its actions to capture subject activity, e.g., successful access requests and authentications that are identifiable to the requesting subject's digital identity;

- Services of the authorisation infrastructure have components that enable observation and control of authorisation assets (authorisation policies and subject privileges), and observation of activity surrounding authorisation assets (changes in policies and assignment of privileges, and access requests and decisions);

- For authorisation assets to be made adaptable, source of authorities (SOAs) must place trust in automated adaptation, but if this is not possible then,
    - SOAs should be capable of accepting notifications about changes to authorisation assets, and are willing to fulfil adaptation requests;
- Identity services provide some means of identifying a subject by their digital identity, either through personal information, a unique persistent ID, or a unique transient ID that the identity service can map to the subject.

Observation and Control of the Target System

The extent of observation and control of an authorisation infrastructure is constrained by the access control methodology implemented, and availability of probes and effectors. It is further constrained by the existence of multiple management domains, where the authorisation infrastructure is distributed and owned by many organisations, such is the case in federated access [93]. With this in mind, the following addresses a holistic view (i.e., no restriction) of what can be observed and controlled within the *target system*.

The *target system* has two unmanageable, but observable assets. These are the activity logs maintained by identity services (capturing authentications and requests for release of access rights) and authorisation services (capturing access requests and decisions). Observation of such assets, notably, activity relating to the request and decision of access, is key to identifying malicious patterns of access, as it can indicate behavioural anomalies (e.g., requesting access from insecure locations, high volume of requests, non-conformance to business processes, etc.).

In regards to the control, there are four manageable (and thus observable) authorisation assets. These are the attributes of a subject, credentials (the release of an access right), policies of an identity service, and policies of an authorisation service. Through the adaptation of such asset types, the SAAF controller is capable of the following:

1. **Control of subject access right assignments**

Asset:	Subject Attribute
Service:	Identity Service
Objective:	To remove, lower, or increase a subject's level of access
Process:	Change in an identity provider's attribute repository in relation to a subject entry

Consequence: Permanently modifies a subject's assigned access right, therefore changing the scope of access a subject has in accessing resources

2. **Control of subject authentication**
   Asset: Subject Attribute
   Service: Identity Service
   Objective: To prevent or allow a subject to authenticate
   Process: Change in an identity provider's attribute repository in relation to a subject entry
   Consequence: Permanently modifies a subject's ability to authenticate with an identity service; resulting in preventing or enabling access to resources that do not require authorisation (e.g., workstations)

3. **Control of attribute / credential release**
   Asset: Policy
   Service: Identity Service
   Objective: To prevent, limit, or increase the scope of what attributes can be released on behalf of a subject as credentials
   Process: Deployment of new policy in identity service
   Consequence: Modifies what attributes can be released to resources; preventing, limiting, increasing credentials a subject can use

4. **Control over revocation of credentials**
   Asset: Subject Credential
   Service: Identity Service
   Objective: To prevent use of an active subject credential
   Process: Change in an identity provider revoking credentials
   Consequence: Revokes a credential (in terms of a released access right) in use at a resource. Subjects who utilise long and short term credentials can repeatedly re-use their credentials to request access, despite changes in their assigned attributes within their identity service's attribute repository. The scope of this applies to either authenticated sessions in which a subject has a credential valid for such session, or if the subject is issued with a long term digital certificate for use throughout many authenticated sessions

5. Control over criteria for access

Asset:	Policy
Service:	Authorisation Service
Objective:	To change the criteria of access within the bounds of the access control methodology
Process:	Deployment of new policy at authorisation service
Consequence:	Changes the requirements for access, such as the required access rights / credentials to utilise a protected resource

Each approach to control has differing consequences for an organisation. For instance a policy change in an authorisation service will likely impact a large number of subjects, whereas changing a subject's set of access rights only impacts the individual subject.

Observation of the Environment

Unlike the *target system*, which can be observed and controlled by a *SAAF controller*, the *environment* (i.e., users, protected resources, and external systems), can only be observed. The *environment* can contain a wide and complex set of observable types of change that could infer subject behaviour. This is potentially limitless given what resources can exist, and what external systems may be present within an organisation. Observation of the *environment* is critical, as it is capable of providing context to subjects who request and gain access to resources, as well as their activity beyond the scope of resources. For example, it is important to observe both the subject's request to access a database, and the subject's behaviour once authorised in accessing a database.

Enabling Observation: Probes

Given SAAF's *target system* and *environment*, there are a finite set of probe types available to a *SAAF controller* to observe both system and environment change. Table 3.1 defines a set of probe types applicable to SAAF's *target system* and *environment*. Probes can be characterised in relation to an owner, being either an authorisation service, identity service, or domain (i.e., the deploying organisation). Probes can also be characterised by the type of changes that they observe.

System change typically refers to a change in the state of access within the *target system*. For example, the 'AS Criteria' probe type is a probe that belongs to an authorisation service, capable of notifying a *SAAF controller* of changes in

Probe Type	Owner	Observed Change
AS Criteria	Auth. Service	System change to criteria for access (e.g., as defined in access control policies)
AS Access	Auth. Service	System & environment change in terms of access requests and decisions
IS Criteria	Identity Service	System change to criteria for the release of subject credentials (e.g., credential issuing policies)
IS Subject	Identity Service	System change to assignment of subject access rights (e.g., assignment of attributes in subject attribute repository)
IS Release	Identity Service	System & environment change referring to the request and release of subject access rights as credentials
Resource	Domain	Environment change detailing a subject's activity within an authorised session for a protected resource
Environment	Domain	Environment change detailing generic subject activity beyond protected resources (e.g., in external systems)

Table 3.1: Target system and environment probe types

the criteria for access. It delivers the change to the controller, where this may be an activation or deactivation of an authorisation service's access control policy. Another example is the 'IS Criteria' probe type, capturing changes regarding what credentials an identity service can release, or the 'IS Subject' probe type, capturing changes to a subject's assigned access rights. Without such notification of system change, a *SAAF controller* is limited in gaining a perception of what the current criteria for access is, or which subject is permitted to access which resource.

Environment change refers to a change in subject behaviour within the *environment*. For example, the 'AS Access' probe type provides a view of subject activity in terms of subject access requests, including activated access rights and granted / denied decisions. Another example is the 'Resource' probe type, capturing environment changes in regards to an authorised subject's activity within a specific resource. Without instantiations of these probe types it is impossible to gain a perception of subject behaviour, which is necessary in identifying attacks.

Enabling Control: Effectors

In order to facilitate control over an authorisation infrastructure, the *SAAF controller* requires a set of *target system* effectors. Effectors are components that perform simple operations against the *target system*, such as deploy access control policy, or revoke credential. It is expected that an effector receives the necessary (adapted) authorisation assets as parameters to such operations.

Effector Type	Owner	Description
AS Criteria	Auth. Service	System change in criteria of access (e.g., deployment of new access control policy)
IS Criteria	Identity Service	System change in criteria for what an identity service can release in terms of credentials for a subject
IS Subject	Identity Service	System change in the assignment of access to a specific subject
IS Credential	Identity Service	System change in the validity of an issued subject credential

Table 3.2: Target system effector types

Table 3.2 details effector types necessary to achieve control over an authorisation infrastructure. These can be characterised in terms of adaptation of an authorisation service, or adaptation of an identity service.

Adaptation of an authorisation service ultimately defines the criteria necessary to gain access to a resource, as denoted by the 'AS Criteria' effector type. In contrast, a greater extent of control is available in regards to adaptation of identity services, where control over what access rights are issued, released, and are active, can be achieved.

3.4 Adaptation Process

The following describes the conceptual aspects of each of the four stages of the SAAF controller's feedback loop. Each stage is broken down into a set of activities, which rely on a set of models (i.e., knowledge) to facilitate their objectives.

3.4.1 Models (Knowledge)

Models are essential to SAAF, as they provide the perception of 'self' in terms of the state of the authorisation infrastructure, and the environment. SAAF promotes the following three models to support its feedback loop.

Access Control Model

The *access control model* (ACM) exists to support the feedback loop with a modelled perception of the state of access control. It is essential for the reasoning of the runtime state of access, which ensures appropriate, consistent, and verified adaptation prior to the enactment of any change against the *target system*.

In essence, the ACM is a homogeneous model that conforms to a given access control methodology (e.g., RBAC). It is causally connected (Figure 3.3) to the *target system's* deployed access control policies and subject privileges. As such, the homogeneous model should be capable of modelling the state of access control exhibited in many diverse implementations of a given approach to access control. For example, a PERMIS policy [31] and XACML [100] policy both conform to RBAC, yet exhibit diverse differences in terms of format, structure, and terminology. It is therefore important for an autonomic controller that seeks to manage RBAC to be capable of abstracting from these complexities.

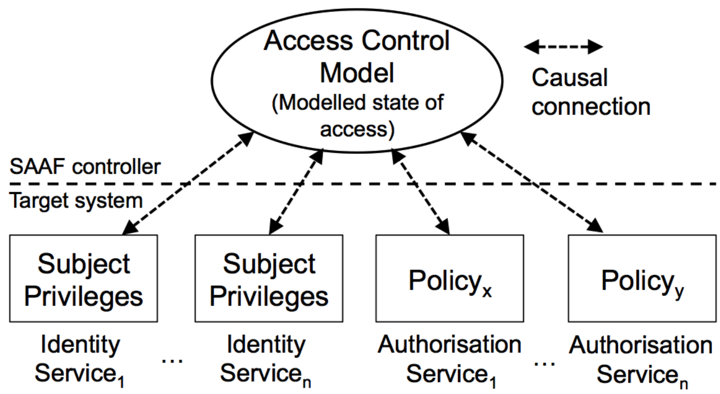


Figure 3.3: Causal connection of access control model to target system

In addition, the ACM is holistic, whereby aspects of access control distributed across services within a *target system* are combined. This is essential to identifying emergent *relationships* as a consequence of the integration of services within an authorisation infrastructure. For example, several access control policies contained within an authorisation service describe constraints over access to protected resources. It is important to combine these constraints with assigned subject privileges contained within identity services, as to identify *relationships* between subjects, their privileges, and accessible resources.

Finally, given the ACM's causal relationship with the target system, adaptation of the ACM should generate potential new states of access within the target system. As such, meta-information must be maintained to specify ownership (i.e., the service(s) an asset exists in), which is essential to derive the relevant access control policies or user privileges for a given service within the *target system*.

Behaviour Model

The *behaviour model* exists to provide a perception of subject activity in relation to the modelled state of access. It is essential to identifying and recording current and historic violations in subject behaviour, as to attain a view of malicious behaviour against the organisation over time.

Primarily, the behaviour model seeks to maintain a set of mutable properties for each relationship within the *access control model* (ACM). These properties reflect patterns of interest concerning the controller's perception of malicious behaviour, and are dynamically generated and updated as a result of the autonomic controller's *monitoring* stage. For example, given the relationship r_1 that describes a subject's ability to access a resource with a certain privilege, a set of mutable properties may describe rates of successful or failed access attempts once the relationship has been observed.

Ultimately, properties maintained within the *behaviour model* are used to identify malicious behaviour, based on the autonomic controller's perception of malicious behaviour. As such, the identification of malicious behaviour (referred to as violations) is also recorded within the *behaviour model*.

Maintaining a history of violations enables the analysis of subject behaviour in regards to malicious activity over time. In this case, analysing both new and historical violations is seen as a critical factor when selecting appropriate solutions to mitigate recently detected violations.

Constraints Model

The *constraints model* captures an organisation's security requirements, in terms of controlling the extent of adaptation. It is essential to providing assurances against the realisation of undesirable states of access that are a consequence of adaptation.

Unlike the *access control model* and *behaviour model*, the *constraints model* is static and defined at deployment time. It contains a set of constraints used as test cases to verify any adaptation made against the *access control model*, prior to enactment within the *target system*. The non-compliance of test cases denote an undesirable state of access control, meaning a given adaptation does not conform to the organisation's security requirements.

The type of test cases that can be defined within the *constraints model* is dependent on the *access control model* that the *target system* implements. For example, in the case of RBAC Core, there is a limited set of test cases that can be evaluated, as detailed in Table 3.3. The table exemplifies a set of test case types that can be verified against a modelled state of access that conforms to RBAC Core, whereby an example instantiation of the last test case would be: *The printer resource should be accessible by at least one subject.*

Test Case Type	Description
Subject, Role	Test a subject is assigned a certain role
Role, Permission	Test that a role can perform a permission
Role, Resource	Test that a role can access all actions on a resource
Role	Test a role has been assigned to a minimum set of subjects
Resource	Test that a resource can be accessed by a minimum set of subjects

Table 3.3: Example adaptation constraints as test cases for RBAC core

Constraints, or even the use of a constraints model is subjective to a deployment of SAAF. For example, different organisations will have their own requirements over the criteria of access at development time, and this would be extended to reflect requirements over the criteria of access at runtime. Should no constraints exist, it is assumed adaptation is unrestricted in changing the state of access.

3.4.2 Monitoring

The *monitoring* stage of SAAF is comprised of three activities: change detection, the update of state of access, and update of state of subject behaviour. Figure 3.4 describes the flow between these three activities, and ultimately the initiation of the analysis stage. These activities are described as follows.

Change Detection

The first activity of the *monitoring* stage involves the detection and filtering of changes sent by probes within the *target system* and *environment*. Changes are

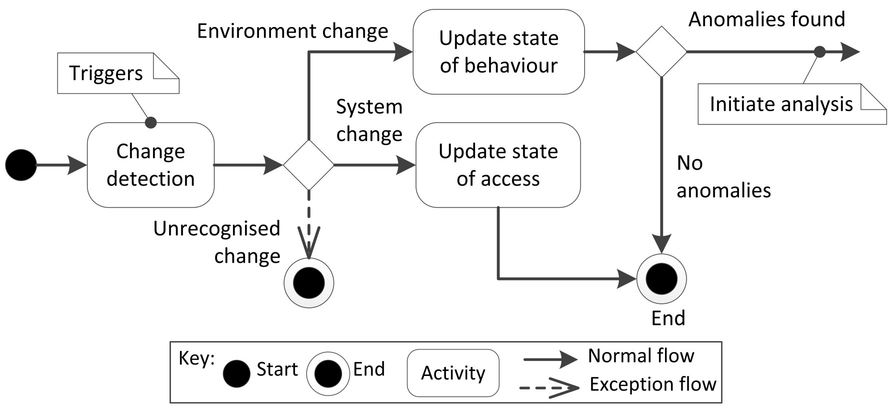


Figure 3.4: Monitoring stage (normal and exceptional flow)

filtered based on relevance to the controller's perception of state. Changes are either relevant to maintaining a synchronised model of access (i.e., the runtime parameters of the *target system*), or relevant to maintaining a perception of subject behaviour.

With regards to the perception of behaviour, the monitoring stage relies on a set of heuristic based rules referred to as triggers. Triggers define the conditions for malicious behaviour, and if met, trigger the need for *analysis*. Each trigger rule describes a *relationship* that can exist within the feedback loop's *access control model*, along with a set of conditions that describe malicious behaviour.

Several types of triggers can be considered. These are constrained to classifying 'known' patterns of malicious behaviour (defined by the deploying organisation). From the viewpoint of the organisation, any behaviour that meets trigger conditions is viewed as malicious. This is considered to be an 'expert' based approach to identifying only known malicious behaviour, where an organisation defines the extremes in behaviour. Adopting alternative approaches, such as a learning-based approach [86] present additional challenges in asserting anomalous activity as being malicious. Whilst a learning-based approach (e.g., machine learning [86]) has the ability to detect unknown malicious behaviour (i.e., not defined by the organisation), it requires greater levels of confidence that detected behaviour is malicious.

In regards to detecting 'known' malicious behaviour, the following example trigger types are defined:

- *Signature based* - A specific action observed, such as access requests made from a blacklisted IP address, or a specific step in an attack.

- *Pattern based* - A pattern of access requests or resource activity, such as a threshold of high frequency access requests to a particular resource by a particular subject / access right / identity service.
- *Transactional based* - A sequence of events that should be adhered to in terms of access or resource activity, such as a subject following an application's workflow.
- *Deviation based* - Thresholds that denote a measure of deviation between current behaviour and past behaviour, such as deviation of a subject's access requests to a particular resource based on average frequency and times of day.

Modelling the State of Access

As changes to the runtime parameters of the *target system* are detected, it is necessary to maintain a synchronised perception of the current state of access. This refers to the causal connection between the *SAAF controller's access control model* and the active access control policies and subject privileges distributed within the *target system*. The activity is critical in ensuring that any potential adaptation has been derived from a perception of state that is consistent with the *target system*. Failing this, adaptation performed cannot be seen as accurate.

The manner in which a *target system* defines its runtime parameters is important, as it is specific to its given implementation. As such, in order to attain a modelled perception of state of access within the *target system*, the implementation's specific view of access must be translated / transformed into a homogeneous representation that a *SAAF controller* can understand. This is not a trivial activity, as transformation must consider the diversity in format, structure, terminology, and the relationship between terms, which may vastly differ in implementations of subject privileges and access control policies.

Various approaches exist to enable transformation, such as model transformation [126] or the use of ontologies [84]. Ontologies have already been demonstrated by Shi et al. [129] in successfully mapping an implementation of RBAC access control policies into another (e.g., the transformation of PERMIS policies [32] into XACML policies [100]).

Regardless of the approach used, the process of transformation should exhibit two necessary outcomes:

- A homogeneous model of access that the *SAAF controller* can understand for a given access control methodology (e.g., RBAC).
- The emergent *relationships* that describe a subject's connection to protected resources, in regards to subject privileges, and defined permissions.

Without the use of a homogeneous model, any implementation of a *SAAF controller* becomes limited to a specific implementation of a *target system*. In addition, the inability to identify a subject's *relationship* to protected resources severely limits the granularity of adaptation in mitigating violations (as the *SAAF controller* has no means to identify which privileges were used to perform violations).

Modelling the State of Behaviour

As with the modelled state of access, a *SAAF controller* seeks to maintain a modelled state of behaviour. Maintaining a perception of behaviour is crucial to identifying the ongoing abuse of access rights. This involves measuring properties of subject activity in their use of access rights and resources in authorised sessions.

The *monitoring stage's change detection* activity employs a technique to dynamically generate properties of a system at runtime. It is based on an approach of using *gauges*, as demonstrated by the Rainbow Framework [53]. A *gauge* is essentially a component of a self-adaptive system that can measure a specific system or environment property. In Rainbow's case, this may be to measure the `latency` of a network device, or `fidelity` of a content delivery service.

In SAAF's case, the scope of properties observable of subject activity is potentially unbounded, where a static approach to configuring such properties presents a number of limitations. For instance, one would need to hardcode every single property specific to a *SAAF controller's* set of triggers, in order to identify violations in behaviour. Therefore, a dynamic approach is needed to generate properties of interest on the fly, as and when relevant subject activity is detected.

To achieve this, *change detection* must dynamically generate a set of *behaviour gauges*, by interrogating the current state of access, and matching relationships of interest presented by triggers. The modelled state of access essentially presents a collection of unique relationship in which properties can exist. Triggers on the other hand express relationships of interest (within the state of access) and a set of conditions. A condition specifies a constraint on the relationship, which indicates malicious behaviour if broken.

For example, a trigger may define the relationship *subject with the role of 'staff' that accesses the resource 'printer'*, and a condition that states *access cannot be greater than 15 requests per hour.* In order to detect a violation of this trigger, it is necessary to measure the rate of access for any members of staff accessing the printer. If there are 20 members of staff that conform to the trigger relationship (i.e., subjects that can access the printer using role 'staff'), then there are at least 20 properties to measure (i.e., the frequency of access per hour).

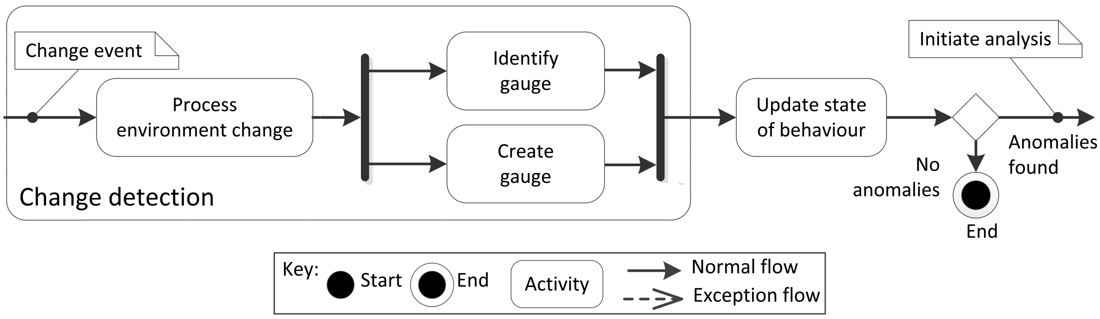


Figure 3.5: Gauge generation (detection of environment change)

For each change event detected by the monitor, the *monitoring* stage goes through the following steps (Figure 3.5) to maintain the modelled state of behaviour:

1. **Process environment change:** Identify relationships within the modelled state of access in which the change event conforms to. In turn, identify the set of triggers which express an interest in conforming relationships;

2. **Identify gauge:** Each gauge contains a unique hash in which to match to a specific property on a relationship, derived from the gauge parent trigger. Should a set of triggers be identified as relevant for a detected change event, this step aims to identify if past gauges have already been created;

3. **Create gauge:** Should no gauges exist but a change event matches a set of triggers, it is necessary to create new gauges in order to evaluate conditions set out by triggers. This is akin to a subject requesting access for the first time, and in order to measure their rate of access to a printer, a gauge is created.

4. **Update state of behaviour:** At the end of change detection, a set of gauges have either been created or identified. In this step the change event is used to update the values measured by a gauge. The state of behaviour is

then said to be updated, given that gauges maintain properties of behaviour tied to relationships within the modelled state of access. Once a gauge becomes full (i.e., its trigger condition is met), a violation has occurred, initiating the *analysis* stage.

3.4.3 Analysis

The *analysis* stage defines three activities to analyse detected change for non-conventional states (i.e., where malicious behaviour is ongoing), and identify possible solutions to respond to such states. Figure 3.7 describes the flow of activities within the analysis stage, including the analysis of behaviour, analysis of solutions, and solution verification.

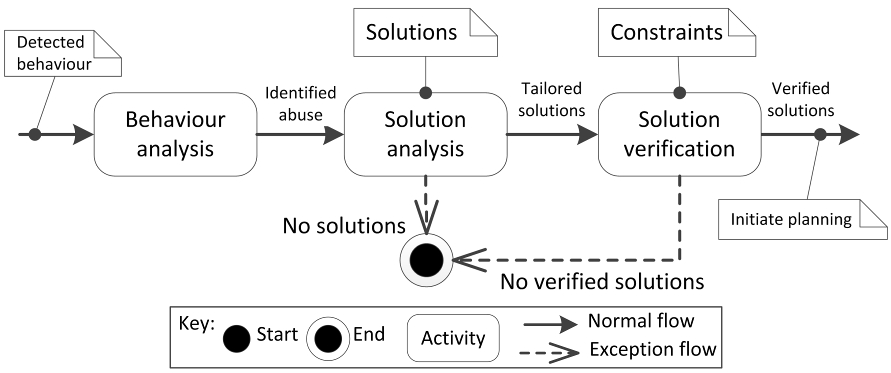


Figure 3.6: Analysis stage (normal and exceptional flow)

Behaviour Analysis

Behaviour analysis seeks to analyse current and historic behaviour in order to first confirm the presence of ongoing malicious behaviour as a violation within the *behaviour model*, and second, to determine the impact of malicious behaviour.

This is a necessary task as a single violation by a subject may not warrant an adaptation on its own. Moreover, each violation may exhibit various consequences to the organisation, given the current state of the system. As such, it is necessary to calculate the subject impact prior to analysing potential solutions.

Calculating a subject's impact is similar to Baracaldo and Joshi's approach in calculating subject trust based on historical behaviour [9]. However, for SAAF it is not a matter of calculating a subject's level of trust, but a level of impact as a consequence of a malicious subject's actions. In turn, the level of impact will become a factor in escalating appropriate and concrete solutions (with greater consequence) to mitigate a subject's activity. In calculating such an impact, it is important to consider multiple dimensions of behaviour (exemplified in Table 3.4) associated with identified (malicious) subjects.

Behaviour Dimension	Description
Cost of identified behaviour	Artificial cost assigned to types of violations
Historical behaviour	Past violations caused by the subject
Rate of detections	Frequency and rapidness in identified behaviour
Collaborators	Subjects identified in performing similar acts of malicious behaviour
Countered adaptation	Whether the identified behaviour is repetition of prior violations, previously thought to be resolved (i.e., reinstated access rights)

Table 3.4: Example criteria of behaviour

For example, a level of impact against the organisation, as dictated by a particular subject's behaviour, can be calculated in terms of the sum of all costs of the subject's identified violations, multiplied by the total number of violations the subject has caused (to escalate response to persistent attacks). This can be improved by associating a level of risk to organisational resources [9], whereby the calculation of impact considers the scope of access a subject has, and criticality of accessible resources.

Solution Analysis

Solution analysis aims to identify a preliminary set of solutions capable of mitigating a subject's malicious behaviour. It relies on a hierarchy of solutions (Figure 3.7) that can be enacted against the state of access, where many solutions can mitigate the behaviour identified. For example, a solution `S2` may seek to remove a specific privilege from a subject (*subject adaptation*), or a solution `S4` may remove a permission assignment to a resource within an access control policy (*policy adaptation*).

Essentially, each solution defines a strategy to constrain / restrict an *access control model*, given the presence of malicious behaviour. As such, each solution is tailorable, where parameters of the solutions are populated with specific artefacts of the detected malicious behaviour (e.g., the permission abused, the source subject, the privilege used, etc.). Each solution exists within a layer of the hierarchy, categorised by *no adaptation* (do nothing), *notification* (warn a subject of their

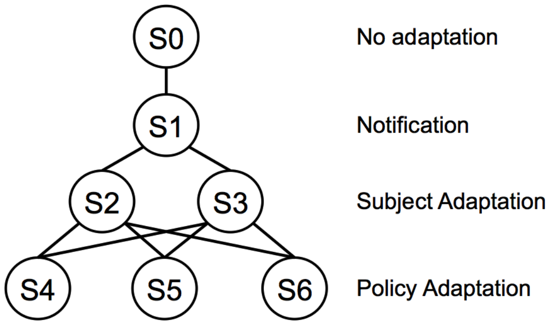


Figure 3.7: Example solution hierarchy in mitigating malicious behaviour

behaviour), *subject adaptation* (impacts only the subject in question), and *policy adaptation* (impacts many subjects in a single adaptation). These layers are considered to exhibit increasing impact to the system, where a number of alternative solutions may exist within each layer.

The use of solutions is akin to a strategy based approach to adaptation, as demonstrated by Rainbow's use of a strategy definition language [36]. A concern with this approach is that an organisation may have to configure an exhaustive set of solutions in order to handle malicious behaviour. However, given an *access control model* there is a minimal and finite scope of actions that can be enacted (e.g., remove role from permission, remove resource from policy).

Considering a subject's calculated impact weighting (identified in *behaviour analysis*), *solution analysis* identifies and tailors a subset of solutions that reflect the behaviour. This is necessary as not all malicious behaviour will clearly warrant high impacting solutions (such as solutions characterised by *policy adaptation*), where analysing all available solutions may result in needless verification and planning (at later stages of the feedback loop). In addition, it is possible for *solution analysis* to identify an empty set of tailored solutions, meaning the subject identified in causing violations presents an impact that is negligible to enacting any of the defined solutions.

Solution Verification

The purpose of solution verification is twofold. First, it is to provide assurances that adaptation result in a consistent state of access in the *target system*, whereby resultant states conform to the implemented access control methodology. For example, in regards to an RBAC implementation, adaptation will not break the inherent constraints of RBAC, such as static separation of duties. Second, it is to provide assurances that adaptation will not conflict with any security requirements specified by the deploying organisation, referred to as adaptation constraints. For example, an organisation may specify fixed criteria for access that regardless of detected violations, must remain active in the state of access.

Verification against an organisation's adaptation constraints is non-trivial, as argued by Montrieux [87], demonstrating the need to automate the process of testing an organisation's *access control model* (i.e., the state of access). Any change to an access control model needs to assure that there are no unwarranted repercussions. This requires a robust approach to checking each relation within the model. Verifying changes to the state of access against adaptation constraints is therefore challenging, and is a particular concern for self-adaptation (due to the need for timeliness in responding to malicious behaviour).

An effective method of verifying a solution is to realise each tailored solution at the model level (prior to adaptation of the *target system*). As the modelled state of access (within the *SAAF controller*) is causally connected to the *target system*, verification of change at model level is akin to verification at the system level. Constraints can therefore be tested against an 'adapted' modelled state of access, in accordance to the specification of a tailored solution.

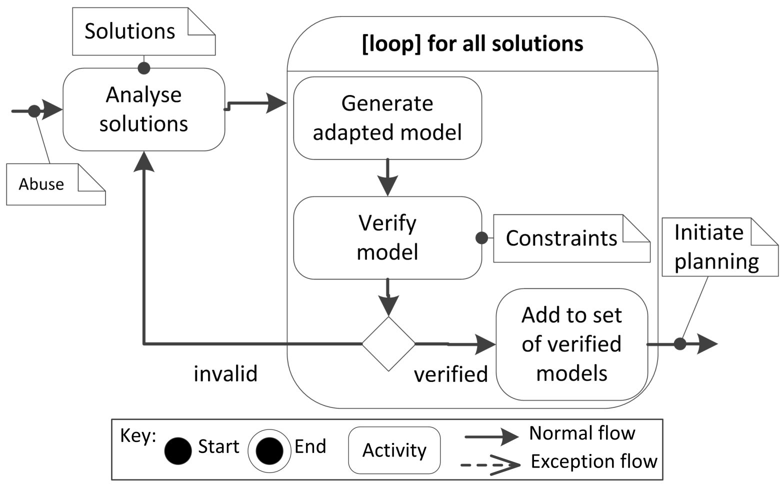


Figure 3.8: Solution verification via model level adaptation

As portrayed by Figure 3.8, each tailored solution results in an adapted model of the state of access, where the model can be verified against the organisation's adaptation constraints, and constraints implicit to the access control methodology. Upon verification of an adapted model, the solution is either added to a list of verified adapted modelled states of access, or rejected. The verified models are then in turn used to initiate the *planning stage*. Failing the outcome of any verified solutions, *solution verification* results in no solutions that can mitigate the identified attack.

3.4.4 Planning

The *planning* stage is focused on how to respond non-conventional states, comprising the two activities shown in Figure 3.9. The first identifies and ranks an appropriate set of solutions out of the applicable set of tailored solutions identified in *analysis*. The second generates a step-by-step executable plan in which to realise an appropriate solution.

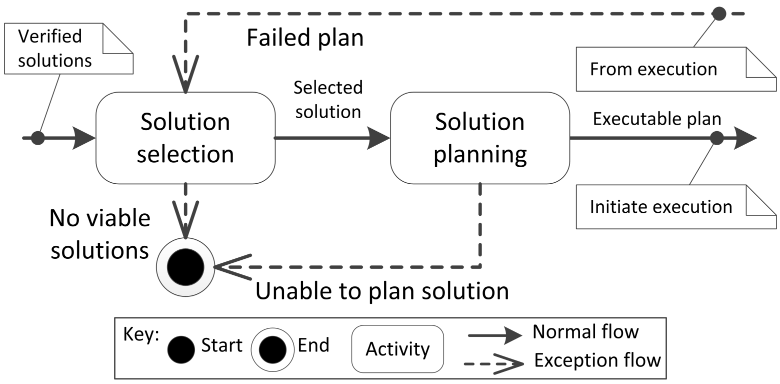


Figure 3.9: Planning stage (normal and exception flow)

Selecting an appropriate solution is critical in limiting the enactment of solutions that result in unnecessary impact to an organisation. Both the detection of behaviour and solution selection are subjective, meaning different deployments will have diverse preferences over what constitutes malicious behaviour, and what constitutes an appropriate solution.

Solution Selection

In order to guide the selection of an appropriate solution, there is a need for a multi-dimensional function to produce a ranking of utility [74], or ranking of cost (cost sensitive modelling [136]). Such a function should consider criteria of identified behaviour (e.g., criticality of abused resources) and criteria of solutions (e.g., service downtime), dependent on the current state of the *access control model* and *behaviour model*. Table 3.5 exemplifies a set of criteria that are applicable to both a utility function approach, and a cost sensitive modelling approach.

Solutions may be discounted at this point, whereby it may be deemed that a candidate solution presents a greater negative impact in comparison to the

Selection Criteria	Description
Behaviour impact	Impact of malicious behaviour on organisation
Recoverability	If a system can recover from a failed adaptation
Availability	Measure of availability of access to resources
Service downtime	Downtime as a result of adaptation
Operational cost	Operational costs in performing adaptation
Adaptation time	Immediacy of adaptation against the target system
Probability of Success	Probability of success based on historic adaptation

Table 3.5: Quality dimensions of solution selection

identified behaviour. In this case, a solution would be inappropriate in mitigating the behaviour. However, assuming a solution does not present a greater impact, it is ranked amongst other candidate solutions. The solution with the lowest rank (i.e., best utility or lowest cost) is then sent for planning. It is possible for *solution selection* to result in a null set of solutions, whereby the identified behaviour cannot be resolved at this point in time.

This process is necessary, as some solutions may become more appropriate over time, and others less appropriate, as further violations are detected. For example, if a single subject abuses their access rights to a printer, a solution may be to modify an authorisation service's access control policy, where a role required to access the printer is no longer valid. The change to the policy would impact all subjects assigned to the role, and for the current state, may not be considered as appropriate due to the potential impact to non-malicious subjects. However, as multiple subjects are identified in abusing their access rights to the same printer, the solution may be seen as appropriate, given the persistence of abuse.

Solution Planning

Solution planning refers to how a solution's actions are to be performed, including the ordering of actions, and identification of action operators. A plan can range from performing a single operation against a target system effector, to a complex set of events that require an ordered set of actions against multiple effectors.

In the case of SAAF, a plan can be quite simple, given the type of operators and effectors available within an authorisation infrastructure. Actions are taken from the selected solution, indicating how the action is to be performed (e.g., encapsulate action as a HTTP request, send to effector), then ordered in terms of generation of new authorisation assets (such as policies, digital certificates), followed by instructions to effectors (e.g., activate policy in authorisation service, deploy new subject digital certificate). Should more complex plans be required, da Silva et al. have shown that the generation of plans can be automated [40], which produces a set of step-by-step instructions with specific details on how to execute an adaptation.

A simple plan to lower a subject's level of access is exemplified in Listing 3.1. The plan is composed of two steps, the first is to obtain the subject's digital certificate, containing the subject's set of assigned access rights, then remove a particular access right to lower the subject's access. An access right in this case is in the form of RBAC roles, stored as a signed digital certificate within an identity service. The lowering of access is achieved through removing a specified attribute from the digital certificate, which was identified in the *solution analysis phase*. The second step is to overwrite a subject's existing attribute certificate within the identity service's attribute repository.

```
[plan: lowerSubjectAccess]
  Sub s = 'cn=bob,ou=orgUnit,o=org,c=gb'   //unique ID for subject
  Service x = ldap://127.0.0.1/          //end point of identity service
  Step 1: Update digital certificate
    Certificate c = getDigitalCertificate(Sub:s, Service:x, Op:ldapsearch)
    removeAttribute(Certificate:c, AttrType:role, AttrValue:OpManager)
    signCertificate(Certificate:c, IssuerCert:ic)
  Step 2: Deploy digital certificate
    updateIdentityService(Service:x, Op:ldapadd, Sub:s, AttrType:attrCert,
                 AttrValue:(Certificate)c)
```

Listing 3.1: Example SAAF controller plan

3.4.5 Execution

The *execution* stage considers two activities (Figure 3.10). The first is the execution of a plan, where actions are executed against relevant effectors within the *target system*. The second is post execution, where the success of the plan is observed and recorded within SAAF's perception of state.

Plan Execution

Plan execution is a simple process in executing the set of actions against *target system* effectors, as specified by a plan. Each action is executed in an idempotent manner, meaning actions are repeated until the desired effector acknowledges the action, or a timeout is met. The outcome of execution results in either a successful

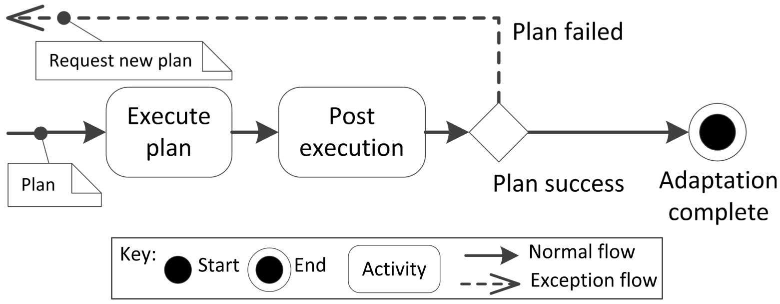


Figure 3.10: Execution stage (normal and exception flow)

or unsuccessful plan. A successful plan is characterised by a positive response from all *target system* effectors. An unsuccessful plan is characterised by either an error message or a failure to respond by at least one of the *target system* effectors.

To perform actions, operators exist within the executing stage of the feedback loop. An operator is similar to that of Rainbow's adaptation operators [53], which define a set of actions that can alter a system's architectural configuration. In SAAF's case, operators enable the modification of parameters of an authorisation infrastructure. For example, an operator is capable of sending a message over a given protocol to a *target system* effector, in order to either perform adaptation of a subject's access rights, or deploy new access control policies.

Post Execution

Upon completion of the successful execution of a plan, *target system* probes update the perception of state within the authorisation infrastructure (as with any system or environment change). This ensures the model of the state of access is synchronised with the *target system* and *environment*. However, if identified malicious behaviour is successfully mitigated it is necessary to reflect this within the feedback loop. This enables future analysis and planning to take into consideration active and unresolved behaviour, whereby a subject who persistently conducts unresolvable behaviour may require alternative solutions.

Should a plan fail (e.g., due to a deactivated effector or error), the execution stage must fall back to the planning stage. As the planning stage ranks a set of candidate solutions, the next ranked solution is selected and planned. If all candidate solutions are exhausted, the identified attack is marked as unresolved within the feedback loop (at the *planning* stage).

3.5 Classifying SAAF

SAAF is classified using the self-protection dimensions provided by Yuan et al. [149]. Of relevance are the dimensions grouped by *approach position* (what SAAF sets out to do) and *approach characterisation* (how SAAF achieves its objectives). A summary of each of Yuan et al.'s dimensions is defined, together with how they were applied to SAAF.

3.5.1 Approach Position

Definition 6 (Self-protection levels [149]) *Defines the extent of self-protection in regards to detection and mitigation of malicious behaviour, as three levels: 'Monitor & Detect' , able to monitor state and detect malicious activity. 'Respond & Protect', able to act on detected activity. 'Plan & Prevent', able to strengthen its security posture based on past events.*

SAAF is characterised as both fulfilling *monitor & detect* capabilities, as well as *respond & protect*. It is capable of monitoring and building a perception of state of an authorisation infrastructure, and detecting the presence of malicious behaviour. It is arguable that SAAF is capable of *plan & prevent*, given the persistent nature of adaptations made to prevent malicious subjects from continuing attacks. However, from a self-protective viewpoint, *plan & prevent* has greater relevance to pre-empting attacks before they can occur. For example, one strategy would be to review authorisation policies and repair security holes before they are exploited. With this in mind, SAAF can only *respond & protect* against current attacks, as opposed to planning and preventing future possible attacks.

Definition 7 (Depths of defence layers [149]) *Classifies a system based on the depth of self-protection at architectural layers, for instance, self-protection at 'network', 'host', 'middleware', or 'application', or 'depth-independent'.*

SAAF's scope of self-protection is confined to *middleware*, where the mitigation of malicious activity is achieved through controlling security measures provided by self-contained services (e.g., authorisation services). SAAF's ability to protect is reliant on its authorisation infrastructure operating as expected in provisioning access, and that protected resources are capable of utilising the authorisation infrastructure as intended.

Definition 8 (Protection goals [149]) *Classifies a system according to the traditional Confidentiality - Integrity - Availability (CIA) model.*

SAAF ensures *confidentiality* through prevention of access as a result of malicious behaviour. For example, any changes made to the criteria for access must conform to a minimum set of security requirements that safeguard against unauthorised access to a business critical resource. To a certain extent SAAF also ensures *integrity* by prevention of access in relation to future abuse. SAAF is able to mitigate such attacks by preventing further access to the attacked resource, as well as other resources the subject has access to. However, it can only be said that *integrity* is assured against resources that have yet to be attacked by an attacker, where a threat was identified but no longer viable, due to the removal of subject access.

Definition 9 (Life-cycle focus [149]) *Defines where self-protection is considered in the life-cycle of a system, such as deployment time or runtime.*

SAAF is positioned for systems at runtime, where an implementation of the framework can operate with existing technologies, to provision self-protection against malicious activity.

Definition 10 (Meta-level separation [149]) *Indicates the extent of separation of concerns between a meta-level subsystem (i.e., a controller) and base-level subsystem (i.e., the target system). The degree of separation is classified by 'no separation', 'partial separation', and 'complete separation'.*

SAAF promotes a *complete separation* between the meta-level and base-level functionality of a system. This is due to SAAF utilising a self-contained controller to exist alongside services of an authorisation infrastructure, where the decision for access, and the decision to adapt, are performed independently.

3.5.2 Approach Characterisation

Definition 11 (Theoretical foundation [149]) *Classifies the theoretical basis in a self-protecting system's decision making methodology, in terms of: 'Logical / formal models', mathematically based techniques for defining security related properties. 'Heuristics-based', knowledge based, or rule-based models that define the conditions to decision making. 'Optimisation', techniques capable of selecting optimal adaptation strategies through quantitative metrics. 'Learning-based', techniques that build upon cognitive, data mining, and probabilistic theory.*

SAAF promotes the mitigation of the abuse of access representative of 'known' attacks. This is achieved through a *heuristics* based approach. *Heuristics* refers to the use of knowledge based rules to identify malicious behaviour and identify an appropriate solution in response to violations.

Definition 12 (Meta-level decision making [149]) *Classifies a system's approach in decision making through coarsely grained categories: 'Single strategy', a single objective approach that applies to detecting a single type of attack or vulnerability. 'Multi strategy', involves multiple levels of decisions, metrics, and tactics. 'Cost sensitive modelling', an approach which involves trade-offs with other non-security related factors in mitigating malicious activity.*

The decision strategy of SAAF is characterised as *multi strategy*. This is due to SAAF promoting the mitigation of abuse of access on several levels (e.g., adaptation of access control policies, and adaptation of user privileges / access rights).

Definition 13 (Control topology [149]) *Classifies the scope of control. This considers whether or not the approach self-protects at a local (i.e., a single host) or global scale (i.e., an authorisation infrastructure). This is further refined as centralised (i.e., a single controller), or decentralised (i.e., multiple connected controllers).*

SAAF follows a *globally centralised* control topology, allowing for a single controller to control distributed components of an authorisation infrastructure.

Definition 14 (Response timing [149]) *Classifies 'when' and 'how' often a self-protecting system performs adaptation. A system is either 'reactive' (adaptation post attack), 'proactive' (adaptation prior to attack), or both.*

Adaptations in SAAF are made in a *reactive* manner, requiring the detection of malicious behaviour, before triggering adaptation.

Definition 15 (Enforcement locale [149]) *Indicates where self-protection is enforced, either at the 'system boundary' (i.e., a firewall) or 'system internal' (i.e., protection of internal components of a system).*

SAAF focuses on detecting and mitigating malicious behaviour attributed to authorised subjects, characterised as *system internal*.

Definition 16 (Self-protection patterns [149]) *The method in which malicious behaviour is mitigated. Approaches are classified as either 'structural patterns', where a system architecture is modified to mitigate attacks, or 'behavioural patterns', where the runtime behaviour of existing components of a system is adapted.*

As no structural change occurs as a result adaptation, SAAF is characterised as following a *behavioural pattern* approach.

3.6 Evaluating SAAF

Self-adaptive systems are challenging to evaluate when compared to traditional systems. This is due to the fact that self-adaptation has the ability to change a system's own state, requiring a test strategy to evaluate changes in state.

To evaluate SAAF's feasibility, the conceptual design is first implemented as a prototype autonomic controller capable of integrating with legacy based authorisation infrastructures (Chapter 4). The prototype controller embodies the concepts of self-adaptive authorisation, whilst also conforming to SAAF's conceptual design. This will enable the evaluation of the principles of SAAF, but also in regards to how well the prototype achieves SAAF's objectives.

A preliminary small scale simulation of an insider threat scenario is necessary to first determine the abilities of the prototype controller (Chapter 4). The simulation is informal, but will indicate the prototype's ability to integrate with legacy services, ability to adapt legacy services, and if adaptation can prevent malicious behaviour from continuing. In addition, it validates the design approach used in implementing the prototype, identifying early limitations that may need to be addressed.

Whilst the preliminary experiment will demonstrate the operation of the SAAF prototype, it is limited in conveying robustness. As such, it is necessary to evaluate the SAAF prototype in a large scale simulation of a complex insider threat scenario (Chapter 5). The simulation will demonstrate the SAAF prototype's ability to repeatedly and robustly mitigate known scenarios of malicious behaviour under various operational conditions. For example, the experiment will aim to identify that SAAF can still achieve its objectives when faced with limitations within its environment, such as a non-cooperating third party organisation. It is intended that through simulation, comparisons can be made against the impact to an organisation, with and without the deployment of self-adaptive authorisation.

Finally, simulation can only demonstrate adaptation in light of known sequences of change within an authorisation infrastructure. What is challenging to simulate is change by intelligent and unpredictable users, which is akin to what SAAF would face in the real-world. To accommodate for this, it is necessary for the SAAF prototype to detect and mitigate unplanned malicious behaviour. Additionally, it is necessary to assess the SAAF prototype under prolonged and large scale change as to emulate the real-world. To achieve this, a gamification [60] approach is used to provide a safe platform in which human users can participate and perform malicious behaviour (Chapter 6). In this case an online game is deployed as a protected resource within an authorisation infrastructure. It is the intention that the SAAF prototype will observe activity within the game, where participants that try to cheat the game are viewed as malicious. In addition, as the experiment involves real users, assessing changes in state (resultant of adaptation) and user activity, will concretely demonstrate a malicious user's inability to continue attacks.

In summary, the evaluation strategy aims to first demonstrate that self-adaptation is possible within an authorisation infrastructure, second to demonstrate robustness in consistently mitigating malicious behaviour, and third, to evaluate mitigation of malicious behaviour by unpredictable and intelligent users. It is through this strategy that conveying the feasibility of SAAF will be achieved, and that self-adaptive authorisation is a worthwhile venture. However, there are limitations. The simulation approach relies on case studies in order to evaluate SAAF against real historic insider attacks. In many cases specific technical data of historic insider attacks is not available, requiring certain assumptions to be made. In addition, the approach does not aim to benchmark the performance of the SAAF prototype, as this is seen as a complementary step to demonstrating feasibility.

3.7 Related Work

To the best of our knowledge there are no other approaches that attempt to solve the problem of abuse of access during runtime through using self-adaptation. However, considering the wider problem of mitigating malicious behaviour, there are a number of approaches that are related to SAAF, as outlined in the literature review. Table 3.6 compares these related approaches to SAAF against some of the dimensions presented by Yuan et al.'s classification of self-protecting systems, and additional dimensions specifically relevant to access control.

- *General Properties*, refers to whether the approach is static or adaptive;
- *Target System*, refers to how the approach integrates with a target system, including, use of existing standards, separation of concerns, integration with legacy systems, and lastly, if applicable for federated environments;
- *Level of Protection*, identifies the extent of protection (Yuan et al);
- *Protection Strategies*, if an approach considers multiple levels of strategies (e.g., adapt at network and middleware level), and if the approach has multiple solutions to a mitigate behaviour (i.e., adapt a policy in multiple ways);
- *Malicious Behaviour*, types of malicious behaviour an approach mitigates;
- *Scope of Adaptation*, whether an approach is capable of adapting and controlling at a policy level (i.e., impact multiple malicious users), on an individual level (i.e., impact an individual malicious user), or other (i.e., structural changes in terms of shutting down networked servers);
- *Adaptation*, properties of the adaptation, either parametric or structural, and whether this is reactive or proactive;
- *Mechanisms*, how adaptation is achieved, such as generation of new access control policies, or dynamic selection of rules at time of access request;
- *Decision Theory*, how adaptation is decided upon (e.g., heuristic based rules), and if decision is static, or dynamic (e,g., capable of changing conditions for decisions during runtime, reflecting on prior adaptation).

As a baseline for comparison, a traditional (static) access control model is compared (RBAC [97]). This is followed by novel approaches to access control, where additional criteria are considered when deciding upon access, to restrict potential abuse of access. For SAAF, access control methodologies, such as RBAC and ABAC, provide the core functionality in protecting an organisations resource. However, these are static and cannot adapt to changes in environment and the system, notably, when an authorised subject abuses their access. Whilst UCON [109], and TrustPDPs [19] extend traditional methods to restrict potential abuse of access, they offer only transient and limited solutions to when malicious behaviour does occur. Persistent solutions (in the form of adaptation) are required that not

	Dimensions	RBAC [97]	UCON [109]	Trust PDPs [19]	Mu et al. [96]	Stakanova et al. [134]	ASBP [151]	RADac [83]	Fuzzy MLS [35]	Baracaldo et al. [9]	SecuriTAS [111]	SAAF
General Properties	Static	✓	✓	✓								
	Adaptive				✓	✓	✓	✓	✓	✓	✓	✓
Target System	Integrates with Standards	✓						✓	✓	✓		✓
	Meta-Level Separation						✓				✓	✓
	Legacy Integration						✓					✓
	Integrates with Federation	✓										✓
Level of Protection	Monitor and Detect				✓	✓	✓	✓	✓	✓	✓	✓
	Respond and Protect				✓	✓	✓	✓	✓	✓		✓
	Plan and Prevent				✓	✓	✓				✓	
Protection Strategies	Single Strategy	✓	✓	✓				✓		✓	✓	✓
	Multi Strategy				✓	✓	✓		✓			
	Multi Solution				✓	✓	✓		✓		✓	✓
Malicious Behaviour	Abuse of Access		✓	✓				✓	✓	✓	✓	✓
	Network Attacks				✓	✓	✓					
	Deployment of Malware						✓					
Scope of Adaptation	Resource				✓	✓	✓					
	Policy				✓	✓					✓	✓
	Subject							✓	✓	✓		✓
	Other				✓	✓	✓					
Adaptation	Parametric				✓	✓		✓	✓	✓	✓	✓
	Structural				✓	✓	✓					
	Reactive				✓	✓	✓	✓	✓	✓		✓
	Proactive										✓	
Mechanisms	Policy Generation				✓	✓						✓
	Access Right Generation											✓
	Dynamic Rule Selection							✓	✓	✓	✓	
	Structural Change				✓	✓	✓					
	Other				✓	✓	✓		✓			
Decision Theory	Heuristic				✓	✓	✓	✓	✓	✓	✓	✓
	Optimisation				✓	✓	✓					
	Learning									✓		
	Fuzzy Logic								✓		✓	
	Static	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
	Dynamic					✓						

Table 3.6: Comparison of solutions related to governance of access and mitigation of malicious behaviour

only reduce the ability for a subject to carry out malicious behaviour, but reacts to when malicious behaviour is identified.

Intrusion response systems are compared (Mu et al. [96] and Stakanova et al. [134]), due to the advancements already demonstrated in mitigating external attacks (e.g., over a network). IRSs are closely related in terms of purpose to SAAF, and as described in Table 3.6, are well suited to mitigating malicious behaviour. Several key differences exist that limit IRSs in mitigating malicious behaviour specific to abuse of access. Both SAAF and IRSs observe differing domains of data to detect malicious activity. An IRS focuses on network traffic and server logs (representative of external attacks), whereas SAAF focuses on data specific to access control and usage of organisational resources within authorised sessions. In terms of adaptation, an IRS can select solutions relevant to the system boundary, such as modification of firewall rules (i.e., policy generation), changes to network structures (i.e., server shutdown), and even carry out counter attacks against the host of an external attacker [135]. For malicious behaviour relevant to insider threat, these types of solutions lack the granularity of control necessary to mitigating internal attacks, such as the fine granularity of access control provided by SAAF.

An architectural-based approach is also compared (ASBP [151]), demonstrating a different angle to mitigating malicious behaviour via structural adaptation. ASBP utilises the Rainbow framework [53] to perform architecture-based self-adaptation. One of ASBP's benefits is the ability to mitigate attacks at a structural level, which has the advantage of tackling types of attack (e.g., network based attacks) beyond SAAF's own scope. However, the mitigation of abuse of access is limited as there is no explicit understanding of how access is defined. ASBP's use of an architectural model to define state is restrictive in this case, given the fact that an access control policy (as a component property) is a composite property, considered a complex model in itself. In addition, these composite properties are inter-related with other component properties; they must be combined to define a complete model of access (i.e., modelling identity services and authorisation services). This is highlighted by the fact that an implementation of SAAF is only relevant for a single type of access control model (e.g., RBAC).

Three risk-based adaptive solutions are then compared (RADac [83], Fuzzy MLS [35], and Baracaldo et al. [9]), where the aims are very similar to that of SAAF, but limited in terms of the granularity of adaptation. These approaches demonstrate transient solutions to mitigating malicious behaviour, performing adaptation at time of request for access. For example, Baracaldo et al. have taken an internal approach where a new access control system is proposed to include detection and adaptation capabilities [9]. Adaptation is achieved through dynamic decision making within a policy decision point (PDP), where a change in risk or trust could prevent a subject gaining access at time of subject access request (despite having all the necessary access rights). In contrast, SAAF presents a separation of concerns approach, where adaptation is applied to existing access control systems through the automated adaptation of access control policies and subject access rights. In SAAF, the separation between access control, detection and response to insider attacks is explicit. Furthermore, SAAF's approach handles insider threat through persistent solutions by adapting access control policies, and subject access rights, for centralised and federated access control systems. The benefit of doing so ensures that malicious subjects are prevented across multiple systems, and enables integration with existing (legacy based) access control technologies (i.e., that were not designed to be self-adaptable).

Lastly, Pasquale et al.'s self-adaptive approach (SecuriTAS [111]) is also compared. Whilst SecuriTAS seeks to prevent a subject from performing certain attacks, it does not react to when subjects carry out an attack. SecuriTAS reduces the likelihood of an attack occurring by restricting the availability of access based on threats and vulnerability (i.e., is preventative). In comparison, SAAF seeks to detect and respond to attacks, and mitigate insider attacks as it happens (i.e., is reactive). The two approaches complement each other in terms of SecuriTAS reducing the overall likelihood (or ability) for an attack to happen, whilst SAAF is capable of directly responding to attacks as they occur.

With the exception of SAAF, no other solution is capable of mitigating malicious behaviour via the self-adaptation of authorisation infrastructures. SAAF integrates with existing technologies and standards, achieving mitigation via fine grained adaptation of access control policies and user privileges. SAAF is by no means a complete solution to mitigating insider attacks, given the fact there are a number of types of attacks a malicious subject could perform beyond the scope of authorisation. For this reason, SAAF is seen as a solution to handling (specifically) the abuse of access rights as and when malicious behaviour is identified. Therefore, solutions, such as Yuan et al.'s architectural-based solution [151], and Pasquale et al.'s SecuriTAS [111] are necessary in terms of handling additional types of internal attacks, as well as reducing the ability of subjects in performing malicious attacks (respectively).

3.8 Summary

In summary, this chapter has presented the Self-Adaptive Authorisation Framework (SAAF) as a means to automate the identification and mitigation of insider attacks. SAAF proposes an approach to achieve self-adaptive authorisation, whereby parameters (access control policies and user privileges) of authorisation infrastructures are adapted, at runtime, in light of malicious behaviour.

SAAF has been discussed at a conceptual level, identifying the framework's *target system*, *environment*, a proposed MAPE-K inspired *controller*, and its integration. The MAPE-K inspired *controller* has been the primary focus, highlighting how, through the use of a feedback loop, autonomic management is achieved within authorisation infrastructures. As such, the *controller* is shown to be a promising component for implementing self-protection in legacy based authorisation infrastructures. A key advantage of the *controller*, compared to traditional approaches, is its robustness when reacting to insider attacks, where circumstances require the authorisation infrastructure to adapt itself in a persistent and timely manner.

To demonstrate the framework's approach for self-adaptive authorisation, and also demonstrate its feasibility in application to real world problems, the rest of this thesis discusses the implementation, deployment, and evaluation of a SAAF *controller* prototype within various authorisation infrastructures. The following chapter positions a model driven implementation of a SAAF *controller* prototype, exemplifying its deployment within a centralised authorisation infrastructure. Specific attention is made in regards to the use of model transformation, which enables a SAAF *controller* to operate with multiple implementations of Attribute and Role Based Access Control (RBAC and ABAC) authorisation infrastructures.

Chapter 4

Model Driven Self-Adaptive Authorisation

4.1 Introduction

This chapter presents a model driven prototype of a SAAF autonomic controller, and its deployment within an authorisation infrastructure. The aim of the prototype is to demonstrate *closing the loop* whereby self-adaptive authorisation can be evaluated. In some aspects, naive methods are used to implement stages of SAAF's feedback loop in order to achieve a proof of concept. As such, it is not the goal of the prototype to evaluate alternative solutions to stages of SAAF's feedback loop, rather its goal is to evaluate the overall concept of SAAF.

Several challenges arise from instantiating SAAF into an autonomic controller, namely: the ability to model, reason, and adapt an authorisation infrastructure at runtime; how an autonomic controller can control diverse implementations of an access control methodology; and lastly, what assurances a controller can provide over adaptation at runtime. A solution to these challenges is to utilise proven methodologies in implementing self-adaptive systems. Particular focus is given to model driven engineering (MDE) [16], where software is developed with reference to the definition and instantiation of domain models (e.g., an abstraction of an access control methodology). MDE is an essential element of the SAAF controller, as the controller benefits from increased interoperability due to the use of 'models', in comparison to adopting a more constrained, and hardcoded approach. Specifically, MDE enables:

1. Generation of models representing implementation specific views of access.

2. Transformation of models into a homogeneous view of access.
3. Model verification to provision assurances before enacting adaptation.

The contribution of this chapter is primarily the realisation of a model driven Self-Adaptive Authorisation Framework (SAAF). This describes the design and implementation of the first operational prototype that enables and demonstrates self-adaptive authorisation. Second, it is the provision of assurances that deployed adaptations will conform to an organisation's security requirements and access control methodology. This involves the manipulation of several diverse models at runtime including: generation, transformation, and verification.

The rest of this chapter is structured as follows. Section 4.2 discusses background topics to this chapter, focusing on model driven engineering and verification. In Section 4.3, the prototype implementation of the SAAF controller is described. Section 4.4 conveys the prototype's integration with a centralised RBAC authorisation infrastructure. In Section 4.5, a preliminary analysis of SAAF describes the application of the SAAF controller prototype to a simulated historic insider attack. In Section 4.6, a summary of the chapter is provided. Finally, Appendix A details complimentary text and results that supports this chapter.

4.2 Background

This section discusses two background topics key to the implementation of SAAF. Namely, model driven engineering [15] applied to runtime modelling of access control, and rbacDSML [87] applied to runtime verification of access control models.

4.2.1 Model Driven Engineering

Model driven engineering (MDE) [16] is a software engineering approach that uses models to abstract and reduce the complexity in implementing systems. It also can be used as a means to capture the 'knowledge' of a system during runtime [2, 16] (i.e., the 'K' in MAPE-K [73]). Models are viewed as first class entities in MDE [125], where they provide understanding of a system (e.g., its structure, state, configuration). A common approach within MDE is to abstract a given system in terms of three layers, known as the 3+1 MDE architecture [15] (Figure 4.1).

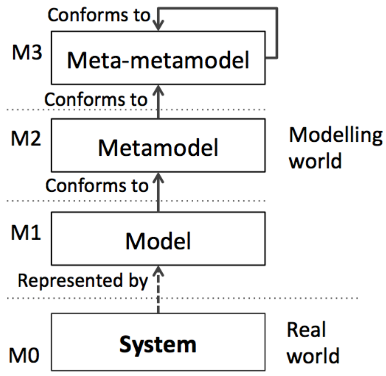


Figure 4.1: The 3+1 MDE architecture [15]

The 3+1 MDE architecture classes model layers in terms of the real world, `M0` (i.e., the system), and the modelled world, `M1` to `M3` (i.e., abstractions of the real world). A concrete example of this is an organisation's requirements of access `M0` is modelled as a UML class diagram `M1`. The class diagram conforms to the UML standard [106], which in this case represents a metamodel `M2` of what can exist within a UML model. Lastly, the UML metamodel must conform to a meta-metamodel `M3`, defining the constructs of the UML modelling language (in this case, the model object family (MOF) [105]).

Models in MDE are contextual, as without context a model is ambiguous. Bezivin et al. argue that in order to provide context, a technical space [16, 78] must be considered. The technical space describes a specific modelling framework in accordance to the 3+1 MDE architecture, where there exists a family of languages to express models, and a set of tools capable of developing, generating, and processing models. For example, the MOF technical space has already been described for UML, another example is the XML technical space.

This research makes use of MDE to develop metamodels that are used to model authorisation infrastructures at runtime. In this case, metamodels are developed to describe states of access control methodologies, such as Attribute Based Access Control (ABAC). The benefits of this approach allows for abstraction from implementation specific knowledge, enabling a SAAF autonomic controller that reasons and adapts at a model level to be free from constraints in interpreting complexity in the target system. It enables a concise definition of the state of access, one that can be transformed (into various implementations) and verified for its conformance to an access control methodology.

Model Transformation

Model transformation [126] is useful for simplifying models, or aggregating multiple models in order to reason about emergent properties. It has the benefit of ensuring output models conform to their corresponding metamodel, negating concerns that an output model may contain invalid types. This can be achieved in terms of transforming a source model to a target model within the same technical space (i.e., endogenous), or to a target model in a different technical space (i.e., exogenous) [85].

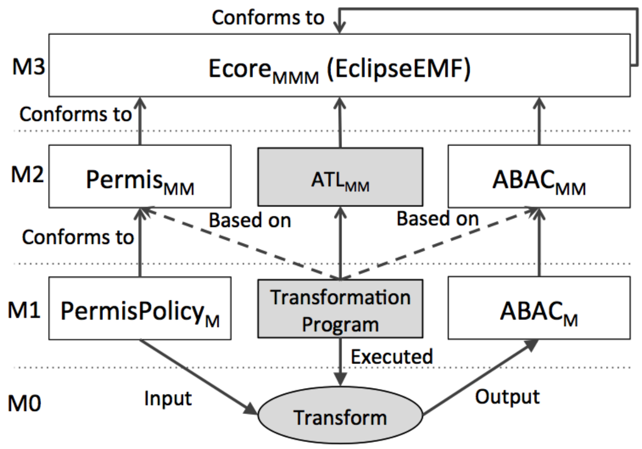


Figure 4.2: Exemplified model transformation pattern [67]

Jouault et al. discuss a model transformation pattern (instantiated in Figure 4.2), for endogenous transformation [67]. In this example, transformation is confined to the Ecore [23] technical space, a prominent model management framework. An ABAC PERMIS access control policy [31] (Section 2.2.4) is modelled in terms of a $PermisPolicy_M$, conforming to a metamodel $Permis_{MM}$. A transformation program aims to transform the $PermisPolicy_M$ into a homogeneous model of access $ABAC_M$, conforming to the $ABAC_{MM}$ metamodel. The transformation program is created using the Atlas transformation language (ATL) [67]), a domain specific language for transforming models within the Ecore technical space. Given a set of diverse ABAC policy formats, transformation enables the generation of a homogeneous model of access that an autonomic controller can reason and adapt.

Model Verification

Model verification enables software developers to ascertain a level of confidence in the modelled state of the system before deployment. It is concerned with how accurately the model achieves its intended goal.

Relevant to this research is the verification of the state of access. Verifying that a model of access meets an organisation's requirements is a complex and non-trivial task, yet several approaches provide programmatic solutions. Hu et al. make use of a model checker [38] in which MAC [75] access control policies are modelled and assessed against logical constraints. In similar work, Hughes and Bultan use a SAT solver [95] to verify XACML access control policies [59]. Here, properties of access control are modelled as mathematical logical formulas that result to true or false, and assessed against access requirements. A final example is verifying RBAC access control through rbacDSML [87], by Montrieux. In Montrieux's approach, requirements are captured within a UML model of access, and are evaluated using OCL constraints [145].

Traditionally, model verification is conducted at development time, where each change requires verification. However, a self-adaptive approach requires verification at runtime, whereby model verification is conducted in line with adaptations. The challenge with adopting such approaches is the cost in verification, as most model verification techniques perform an exhaustive task to produce a high level of confidence of verified models.

4.2.2 rbacDSML: RBAC Verification Tool

As discussed previously, Montrieux's rbacDSML approach [87] allows for the verification of RBAC access control models, enabling large organisations to manage their access control models with greater accuracy, efficiency, and assurance.

rbacDSML defines a domain specific modelling language (DSL). It enables the modelling of RBAC access control criteria in conformance to the RBAC standard. A standalone tool has also been developed [88], and enhanced for self-adaptive authorisation [8]. In addition to enabling the design of RBAC models, it also enables verification of the model against the RBAC standard and access requirements. Finally, given a RBAC model that fails to meet the access requirements, the rbacDSML tool is capable of identifying fixes to the model. However, this is a time consuming process and currently inappropriate for runtime use.

rbacDSML is implemented as a profile that extends the Unified Modelling Language (UML). It defines UML stereotypes and associations to express what can exist in an RBAC model, and how elements of an RBAC model relate. Figure 4.3 conveys rbacDSML's metamodel, in which users (subjects), roles, permissions, and

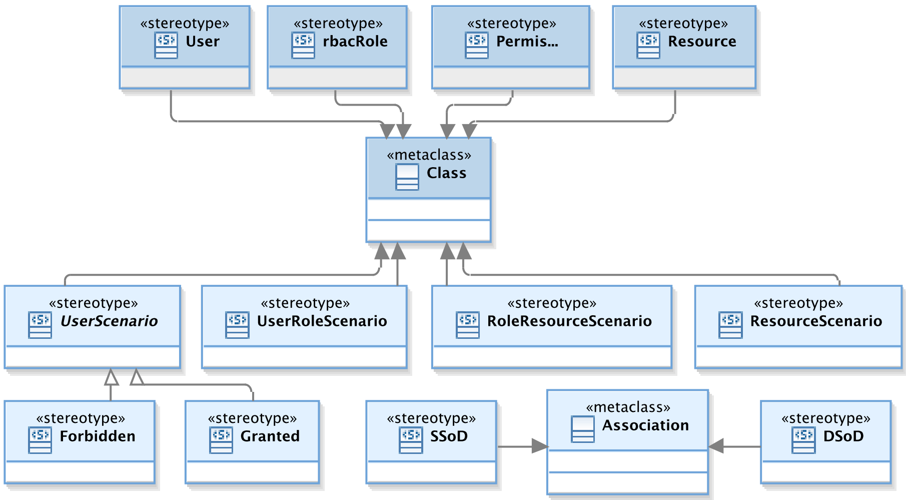


Figure 4.3: Extended rbacDSML metamodel [8] in MOF

resources are defined as stereotypes. Facets of the RBAC standard are defined with rbacDSML's metamodel as stereotyped associations between stereotyped classes.

In addition to the stereotyped classes to represent the RBAC standard, the rbacDSML tool also includes 'scenario' stereotypes. These stereotypes are critical to capturing access requirements that an RBAC model must be verified against. The use of OCL constraints enables the verification of scenarios against the RBAC model, and can verify the model in a variety of ways: *User scenarios* exist to specify an access control requirement, where a user must be granted (or denied) access to a particular resource (e.g., User 'bob' can access Resource 'printer'). *User - role scenarios* express a condition where a role must be assigned to at least one user (e.g., Role 'staff' must have at least one user). *Role - resource scenarios* can be verified against an RBAC model to ensure a particular role has access to a given resource, via the role's permissions. Finally, *Resource scenarios* define a minimum availability of access to a resource, where at least one user should be capable in accessing a given resource.

4.3 SAAF Controller: Prototype

This section describes the design and implementation of the Self-Adaptive Authorisation Framework (SAAF) controller prototype. The aim of the prototype is to showcase a fully working feedback loop, capable of monitoring and controlling a class of authorisations infrastructures (i.e., categorised by a given access control model). For implementation reasons, some processes within the controller's feedback loop have been simplified in order to *close the loop*, these processes can be improved upon by future projects. Statistics pertaining to the substantive nature of the prototype's code base are identified in Appendix A.1.

4.3.1 Controller Architecture

The SAAF controller prototype (Figure 4.4) is implemented as a standalone Java (1.7) application, in conformance to the conceptual design of SAAF. The scope of control is constrained to RBAC and ABAC authorisation infrastructures. The prototype employs an ABAC metamodel to instantiate runtime models of access.

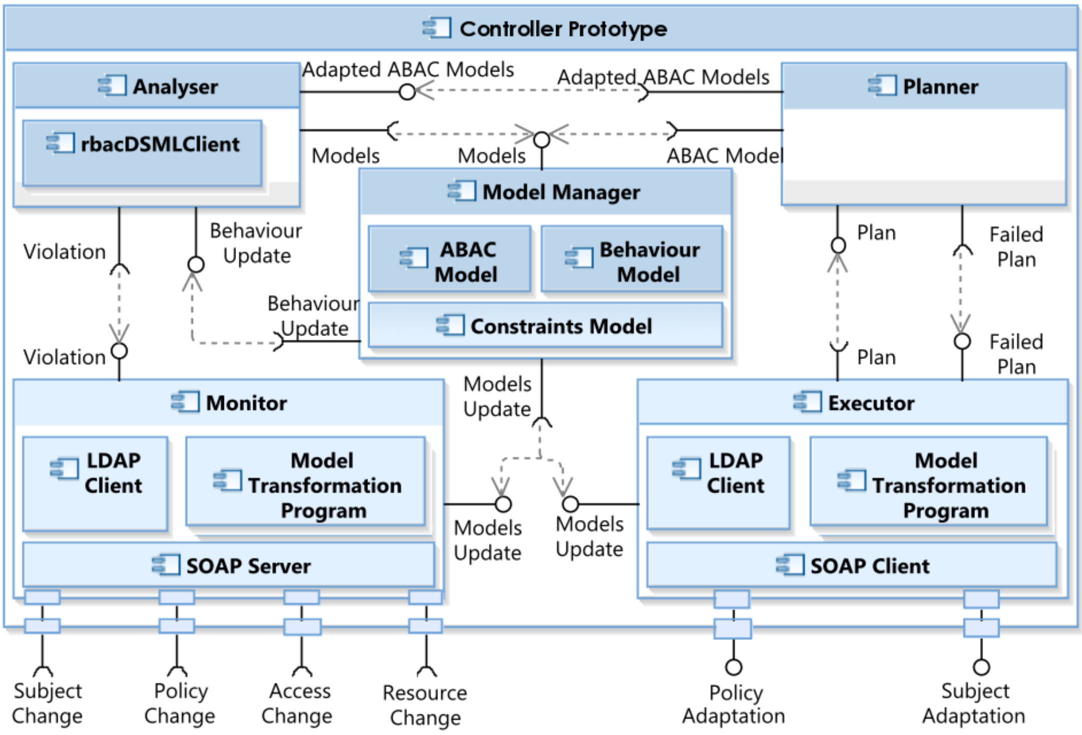


Figure 4.4: SAAF controller prototype

The ABAC metamodel is designed and implemented using the Eclipse Modelling Framework (EclipseEMF) [23]. EclipseEMF provides the ability to design structured data models, which in turn are used to automatically generate Java code. This generated code implements the *SAAF Model Manager*, allowing for the creation, modification, validation, change notification, and automated serialisation of *ABAC models*. The ABAC metamodel is critical to the prototype, as without it the controller is unable to reason about state in both RBAC and ABAC authorisation infrastructures (as RBAC is a specialisation of ABAC).

The *monitor* component, implemented as a concurrent process, continually seeks to detect the four types of changes identified via its required interfaces. Upon detection of change, monitoring results in either: the generation and update of an *ABAC model* (i.e., the controller's *access control model*), or the update of the prototype's *behaviour model*. Changes are observed via a variety of methods. A Simple Object Access Protocol (SOAP) [21] server allows for probes to send messages in a specified format to the prototype's *monitor*. Alternatively, the use of LDAP client tools (LDAP CTools [76]) enable the *monitor* to interface directly with LDAP identity services. Finally, features of the host operating system can be used to interface directly with elements of an authorisation infrastructure, such as an authorisation service's access control logs.

To generate a modelled state of access, the *monitor* must transform 'implementation specific models (views)' of access, from the perspective of services within the controller's authorisation infrastructure. This is with reference to the diverse implementations of authorisation services and identity services that can conform to the same access control methodology. Transformation is not a trivial task, and requires a set of *model transformation programs* that take implementation specific models (created from a target system's authorisation assets), and outputs the prototype's *ABAC model*.

Model transformation programs are implemented in EclipseEMF using the Atlas Transformation Language (ATL) [67]. EclipseEMF is used to either automatically generate metamodels of implementation specific services (e.g., converting an authorisation service's XML policy schema into a metamodel), or through manual generation of metamodels using EclipseEMF's graphical tools. ATL is used to express transformation rules that are capable of mapping elements and relationships within input metamodels, to a target output metamodel.

The *analyser*, *planner*, and *executor* are invoked sequentially, once monitoring has indicated the presence of malicious behaviour. In regards to analysis, a cost function is used to produce a weighted impact of behaviour (behaviour analysis) for malicious subjects, based on an aggregated cost of behaviour. Solution analysis is performed in terms of generating *adapted ABAC models* (as solutions applied to the current *ABAC model*). These adapted models embody the mitigation of malicious behaviour at a model level. Solution verification is implemented through the integration of an existing verification tool *rbacDSML* [88], whereby *adapted ABAC models* are verified against an organisation's security requirements (contained within the *constraints model*). In regards to planning, an appropriate *adapted ABAC model* is selected via a cost sensitive function that weighs the impact of identified malicious behaviour, to the impact of the modelled solution. The selected *adapted ABAC model* is then used to represent the goals of a plan, defining a new desired state of access.

The *executor* realises a plan by first transforming the *adapted ABAC model* back into implementation specific models, used to automatically generate new access control policies and access rights. Available to the *executor* is a set of operators that enable the generation of authorisation assets, and activation of assets within the target system. As with the *monitor*, the *executor* provides three methods in which to interact with the target system:

- A SOAP client, capable of sending authorisation assets via SOAP messages to target system effectors
- An LDAP client (specific to LDAP based identity services) for adaptation of subject privileges
- Host operating system calls for persistent state storage of privileges and polices on a shared file store

4.3.2 Scope of Observation and Control

The scope of the prototype is constrained to an Attribute Based Access Control (ABAC) metamodel, and the availability of relevant probes and effectors.

ABAC Model and ABAC Metamodel

The ABAC metamodel has been implemented in conformance to Ecore. The metamodel allows for the instantiation of both RBAC and ABAC access control models. The ABAC model is used predominately by the analysis and planning components, removing the need for these components to understand diverse implementation specific views of RBAC and ABAC.

Figure 4.5 presents a class diagram of the ABAC metamodel, defined specifically for SAAF. Although ABAC is yet to be standardised, the metamodel used by SAAF is capable of capturing the core facets of ABAC, which builds upon the RBAC standard. Namely, this has replaced a *Role* with the use of *Attribute*, a tuple $\langle AttributeType, AttributeValue \rangle$, such as $\langle role, Staff \rangle$ or $\langle gender, Male \rangle$. The notion of *Issuer* is also introduced, which refers to the growing use of ABAC in

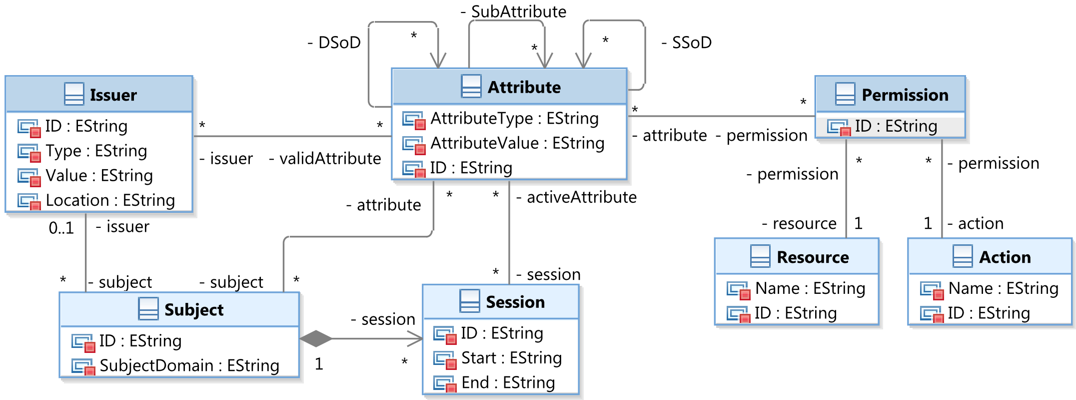


Figure 4.5: SAAF ABAC metamodel defined in Ecore

federated environments. *Issuer* describes an identity service or identity provider organisation. It allows for the containment of subjects within an issuer, as well as definition of *validAttribute* to distinguish which issuers are trusted to issue which attributes for subject as credentials (i.e., credential validation [32]). An instantiation of the ABAC metamodel can be viewed as sets of elements and relationships:

$$ABAC_M = \langle S, SS, IS, A, P, RS, AC, R\rangle$$

Where: S is a set of subjects. SS is a set of subject sessions (i.e., where subjects currently have been awarded access). IS is a set of issuers' of attributes to a subject (i.e., an identity service). A is the set of observed attributes (i.e., access rights). P is a set of ABAC permissions. RS is the set of protected resources, and AC is the set of actions executable on resources. Lastly, a set of *relationships* R exist that contain a subset of model elements. For example, given $r_1 \in R$, r_1 may describe a tuple of: $\langle is \in IS$, $s \in S$, $a \in A$, $p \in P$, $rs \in RS$, $ac \in AC\rangle$. Ultimately for this relation, it describes the capability of a subject from a given issuer, in invoking a permission that allows the subject access to a resource.

Implementation Specific Models

Two implementation specific metamodels are used to demonstrate the integration of the controller prototype with existing implementations of ABAC.

For an identity service, an LDAP metamodel ($LDAP_{MM}$) is created in Ecore to model an LDAP directory (see Appendix A.2). $LDAP_{MM}$ is used to model *subjects* and *attribute assignments* within an LDAP directory. The metamodel has been defined manually, identifying the attributes of the LDAP directory, its contained *subjects* as LDAP entries, and subject *attributes*.

Given that this is a prototype, currently the PERMIS policy language [32] is the only policy language where adaptation will be demonstrated against. An implementation specific model allows for the modelling of instances of PERMIS policies, enabling the controller prototype to adapt PERMIS policies (and thus integrate with PERMIS authorisation services).

Therefore, to demonstrate integration with an authorisation service, a PERMIS metamodel ($PERMIS_{MM}$) is created in Ecore to model PERMIS authorisation policies. A PERMIS authorisation policy can express both RBAC and ABAC access control rules, as well as credential validation rules for validation of *issuers*. The metamodel is generated automatically through EclipseEMF using PERMIS's proprietary XML schema. A simplified variant (due to complexity) of the PERMIS metamodel is described in Appendix A.3, sharing commonalities with SAAF's ABAC metamodel. For example, PERMIS's *roleAssignmentPolicyType* is synonymous with the *validAttribute* association between *issuer* and *attribute* in the prototype's $ABAC_{MM}$. Another example is PERMIS's *targetAccessPolicyType*, which is synonymous to capturing permission attribute assignments within SAAF's $ABAC_{MM}$.

Observation

The prototype (Figure 4.4) is deployed with a set of detectors to interface with its target system and environment. A set of generic detectors are implemented to observe changes in relation to: *Access Decisions*, *Policies*, *Subject Access Rights*, and *Resources*. Detectors observe information via messages sent to SAAF's *SOAP Server* or via the host machine's operating system. The types of change the monitor can detect are formally described in Appendix A.4.

Control

A set of operators (Figure 4.4) are implemented to enable the realisation of *plans* (generated by the *planner* component) via target system effectors. Operators are used to either generate a particular type of authorisation asset, or deliver and apply an authorisation asset on a particular service.

Generic operators, such as *LDAPClient* tools, and the *SOAPSender* enable the deliverance or direct modification of authorisation assets within services of an authorisation infrastructure. These operators require the destination target of a command to conform to a specific message format. These are described in Appendix A.5. Some actions require more specific operators, such as a *PolicyBuilder* and *CertificateEditor*. The *PolicyBuilder* serialises implementation specific 'policy models' transformed from SAAF's ABAC model, into the specific service's required file format and structure. The *CertificateEditor* generates X.509 certificates and is capable of signing policies or statements of access rights on behalf of source of authorities or issuers respectively.

4.3.3 Modelling the State of Access Control (Monitoring)

The prototype's ABAC model ($ABAC_M$) is generated at runtime, syncronised with the target system. To achieve a runtime modelled state of access, the controller undergoes a process of *model generation* and *model transformation*.

Model Generation

The SAAF controller generates implementation specific models via its *monitor* component within the MAPE-K loop, whereby probes obtain information about target system services. Implementation specific model managers, automatically generated through EclipseEMF, are used to instantiate these models. This is done either through the injection of XML, or through bespoke injectors created for specific services (e.g., LDAP). Each generated model is stored in a persistent state that can be retrieved at any point during runtime. As discussed previously, the prototype controller is implemented with two implementation specific metamodels: one for LDAP identity services, and one for PERMIS authorisation services.

A model generated from the LDAP identity service provides a view of active subjects and subject attribute assignments. A model generated from the PERMIS authorisation service provides a view of active RBAC or ABAC access control rules, such as permission assignments. Both of the generated models conform to their implementation specific metamodels.

Model Transformation

Relying on implementation specific models is useful since it is easier to understand and adapt the current state of the LDAP and PERMIS services. However, for analysing and verifying a modelled state of access, the SAAF controller must combine the implementation specific models into a single model. This single, homogeneous model embodies the authorisation infrastructure's access control methodology (i.e., ABAC). Figure 4.6 portrays the creation of SAAF's ABAC model through model transformation, when transforming implementation specific generated models into ABAC.

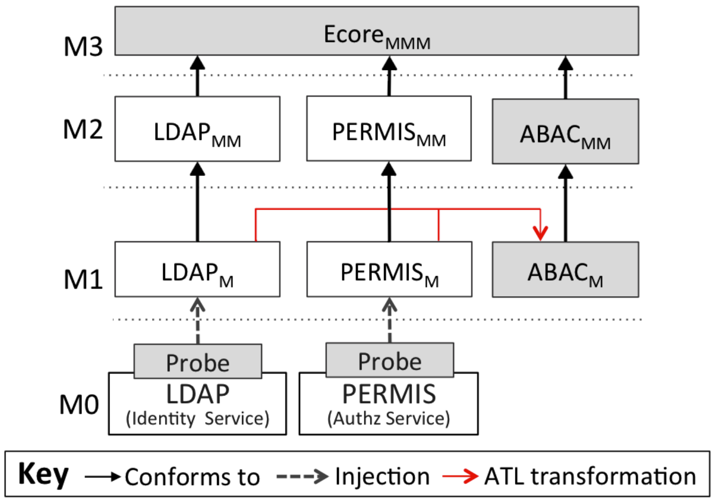


Figure 4.6: Transformation of implementation specific models to ABAC

The ABAC model is the product of an Atlas Transformation Language (ATL) [67] model transformation program, referred to as *PERMIS+LDAP2ABAC*. The transformation program takes as input a model generated from what is injected (i.e., observed) from the the LDAP directory, and a model generated from what is injected from the PERMIS active access control policy. Using the three metamodels defined in Section 4.3.2 (i.e., PERMIS, LDAP, and ABAC), a set of transformation rules are defined that map relationships and elements of each implementation specific metamodels, to SAAF's ABAC metamodel ($ABAC_{MM}$). The output of the transformation program is an ABAC model ($ABAC_M$) that conforms to the prototype's $ABAC_{MM}$.

Each time either implementation specific models are updated, the SAAF controller performs the *PERMIS+LDAP2ABAC* transformation. This ensures that the SAAF controller maintains a synchronised modelled state of access that exists within the authorisation infrastructure. Once the $ABAC_M$ has been created through transformation, the SAAF controller is able to reason about the state of access and adapt the $ABAC_M$ in light of detected violations.

Listing 4.1 provides an excerpt of the PERMIS+LDAP2ABAC transformation program, defined using the ATL language. It describes the generation of a *Resource* type within the ABAC model from a PERMIS's *targetDomainSpec* type, where *policy56* refers to the schema ID of PERMIS. In addition, the creation of *subject* types within the ABAC model is shown, where the LDAP *subject* type is transformed and added to an issuer, within the ABAC model ($ABAC_M$).

```
--Get all the resources from the Permis policy
lazy rule getResources{
  from
    s: policy56!TargetDomainSpec
  to
    t: saafABAC!Resource(
      ID <- s.ID,
      Name <- thisModule.getInclude(s).URL
    )
}

--Create subjects from LDAP model
lazy rule createSubjects{
  from
    s: LDAP!Subject
  to
    t: saafABAC!Subject(
      ID <- s.Identifier,
      Issuer <- thisModule.getSubjectIssuer(),
      Attribute <- thisModule.getSubjectAttributes(s.Attribute)
    )
  do{
    --Add subject to issuer reference
    thisModule.getSubjectIssuer().Subject <- t;
  }
}
```

Listing 4.1: ATL transformation rules for resource and subject

4.3.4 Detecting Malicious Behaviour (Monitoring)

This prototype adopts a heuristic based approach to detecting 'known' malicious behaviour, akin to pattern based rules utilised in intrusion detection systems [120]. Malicious patterns of subject behaviour are represented as a set of *trigger* rules, defined within a *behaviour policy*. When a subject's behaviour conforms to a trigger rule, a violation has occurred, triggering *behaviour analysis*.

SAAF's *behaviour policy*, is defined by a proprietary XML schema specific to SAAF. It is parsed into the *monitor* in order to filter change, generate and populate gauges, and identify violations. It was decided to use a proprietary schema as existing universal policy languages, such as Ponder [41], presented unnecessary complexity to demonstrate the prototype. In addition, a potential policy language built to define violations and guide strategy selection (Stitch [36]) was too specific to architectural based properties to use for the specification of malicious behaviour.

An example policy and its specification is described in Appendix A.6. To summarise, it can contain two sets of *behaviours* known as *base* and *composite* behaviours.

Base behaviours are defined as a tuple that contains a *relationship*, a set of conditions, and a cost weighting. The *relationship* refers to a relation within the controller's $ABAC_M$, such as how a subject, an issuer, an assigned attribute, and a permission are connected. Given the relationship, a set of conditions are described that define unacceptable patterns of behaviour in reference to the relationship. For example, a rate of access over an interval (pattern based), a set of access requests in conformance to a workflow (transaction based), a distance between current and historical activity (deviation based), or access from a blacklisted IP address (signature based). Lastly, a cost weighting defines an artificial cost to the deploying organisation should changes observed in the authorisation infrastructure conform to the set of conditions.

Composite behaviours vary slightly in comparison, where instead of a relationship, they refer to a unordered subset of *base behaviours* that are contained in the *behaviour policy*. This enables a composite behaviour rule to define malicious behaviour in terms of a set of activities, where each activity could be viewed as an iteration in *base behaviours* (e.g., components of a high severity attack). As with a *base behaviour*, a set of conditions are applied, and a cost weighting.

4.3.5 Analysing Behaviour and Solutions (Analysis)

The implemented *analyser* operates on a snapshot of subject behaviour at the time a violation has been detected. To mitigate violations, the *analyser* relies on a set of template solutions that contain actions applicable to the $ABAC_{MM}$. These template solutions, contained within a *solution policy*, define the type of adaptations the controller can apply to its target system.

A limitation to this approach is that the *analyser* requires each violation to be mitigated in order of detection, where this may lead to redundant adaptations or bottlenecks in mitigation. Future implementations of the *analyser* should aggregate detected violations, as well as operate with a fluid perception of behaviour.

Behaviour Analysis

Behaviour analysis is implemented through the assessment of current and previous trigger violations. It requires a perception of behaviour, defined within the controller's *behaviour model* (Bhv_M). The Bhv_M is a collection of real time properties and violations assigned to relationships within the prototype's $ABAC_M$. It is the product of SAAF's *monitoring* and *analysis* stage, where properties reflect trigger conditions defined within the *behaviour policy*.

For example, a set of triggers within the *behaviour policy* drives the dynamic creation of gauges. Each gauge relates to a specific relationship within the $ABAC_M$, such as $r_1{=}\langle subject.Bob, role.Researcher, resource.ElectronicLibrary\rangle$. A gauge creates and maintains a particular property of its $ABAC_M$ relationship, such as *totalDownloads*, based on the conditions described within its respective trigger. Given a set of gauges, a collated set of properties are maintained, such as a tuple of properties $\langle totalDownloads, requestsPerMin, requestsPerHour\rangle$ for each instance of an applicable $ABAC_M$ relationship. As trigger conditions conform to properties, violations are stored within the Bhv_M, maintaining a historical view of all violations associated to their respective $ABAC_M$ relationships. It is assumed that all relationships can be connected to a source subject, in which to derive a complete view of a subject's behaviour and violations.

Given the Bhv_M, *behaviour analysis* must identify if a subject's behaviour warrants adaptation. A normalisation function is implemented to calculate the impact of a subject against the organisation's resources, based on the number and aggregated cost of their violations. Each time a subject is detected in performing a violation, the subject's impact (Sub_{Impact}) is calculated and normalised on a scale of 0 to 1, where 0 represents no malicious impact, and 1 represents the highest malicious impact. Albeit simplistic, the naive approach enables the escalation of a subject's impact, given the persistence in identified violations over time. However, beyond the proof of concept it will be necessary to consider a far wider set of factors in determining impact, over cost alone.

$$Sub_{Impact} = \frac{(\sum v_w \times V_{count}) - Cost_{min}}{Cost_{max} - Cost_{min}}$$

As such, Sub_{Impact} is calculated as the sum of all costs of the subject's violations v_w (derived from the corresponding triggers within the *behaviour policy*) multiplied by number of violations V_{count} the subject has performed. This is normalised based on a minimum and maximum cost that is set by the organisation in order to define the extreme bounds of impact a subject can exert on an organisation.

Solutions

Solutions are characterised in terms of a change to the target system's state of access. The *solution policy* contains a set of template solutions, whereby each solution is described by a tuple of required minimum observed impact (i.e., before considering this solution against a violation), a set of parameterised solution actions (i.e., executable operations against an $ABAC_M$), and a subset of behaviours that a solution can mitigate (i.e., described by the *behaviour policy*). The minimum observed impact is a value between 0 and 1, defining the scope of solutions applicable to a subject's calculated impact (calculated in *behaviour analysis*).

Each solution action describes a tuple of *paramaterised operation* and a *cost weighting*. The paramaterised operation is an action that adapts a particular aspect of the $ABAC_M$ (e.g., *lowerSubjectAccess(Subject)*, or *removeAttributePermission(Attr,Perm)*). Whereas the cost weighting defines an artificial cost that describes the perceived consequence to an organisation (e.g., the potential cost lost to an organisation for completely removing a subject's access).

For the prototype controller, the extent of solution actions are constrained to removing and lowering the scope of access, either on an individual scale (subject adaptation) or collective scale (policy adaptation), in conformance to the prototype's $ABAC_{MM}$.

Likewise to SAAF's *behaviour policy*, the *solution policy* is also defined using a proprietary XML schema specific to SAAF. The *solution policy's* specification, along with a complete list of implemented solution actions are described in Appendix A.7.

Solution Analysis

Once violations have been detected and analysed, a set of applicable solutions are identified. Given the calculated impact of detected violations, solution analysis filters applicable solutions based on violation impact meeting the solution's minimum impact threshold. This enables the controller to scope a set of solutions that mitigate the violation, based on the varying impact and persistence of a subject's malicious behaviour. For example, given a subject persistently identified as causing violations, solutions with greater consequence to the subject are chosen.

Figure 4.7 depicts a partial view of the SAAF controller, capturing the process of analysis, and providing the *planner* component with a set of solutions (as *adapted ABAC models*). Here the analyser has analysed a subject's malicious behaviour, and identified a set of solutions, denoted as $\{S^1, S^2, S^3\}$. Each solution is tailored in terms of populating the required parameters for each solution action, either from the $ABAC_M$, or from the attributes of the detected violations. These tailored solutions are then used to adapt a snapshot of the current access control model, resulting in a set of *adapted ABAC models*, exemplified by $\{ABAC_M{}^1, ABAC_M{}^2, ABAC_M{}^3\}$.

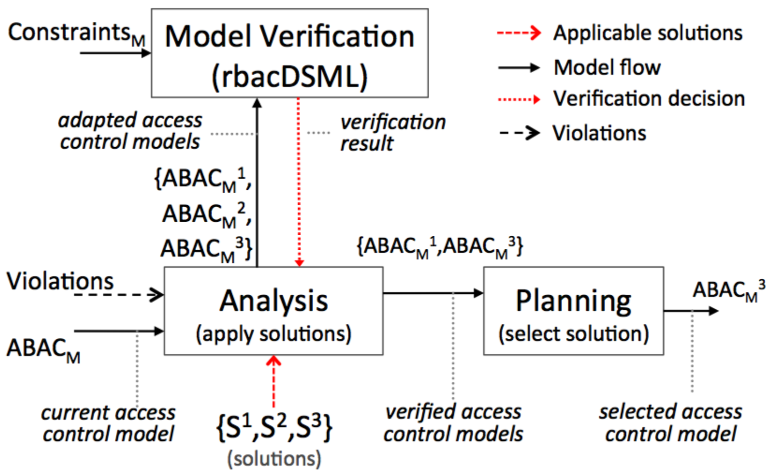


Figure 4.7: SAAF controller model verification

In some cases, where verification has been incorporated (i.e., to verify against an organisation's set of security requirements), these adapted models can be verified using the rbacDSML model verification component[1]. Depending on a positive output of the verification tool (if applicable), the *adapted ABAC models* are collated into a set of verified models, and sent to the *planning* component.

Solution Verification

For enabling verification, the prototype utilises a standalone version of rbacDSML, by Montrieux [88]. Providing the prototype's target system conforms to the RBAC standard, any *adapted ABAC model* can be verified using rbacDSML.

Each *adapted ABAC model* represents a new state of access within the authorisation infrastructure, essentially iterating changes in the design of access. It

[1] Verification is limited to only RBAC compliant variants of SAAF's $ABAC_M$, as rbacDSML can only verify RBAC models

is therefore necessary to obtain assurances that the new design will not conflict with any of the organisation's security requirements. For example, ensuring that all resources are accessible by at least one subject. An organisation's security requirements are defined in SAAF's *constraints model*, which is a static UML model expressed in rbacDSML's own proprietary UML profile.

The standalone version of the rbacDSML tool achieves verification of an access control model via the execution of OCL constraints. Each security requirement, defined as a rbacDSML scenario within SAAF's *constraint model*, has a *context*: a stereotype on which the scenario applies. The verification process evaluates each OCL constraint on every instance of its context stereotype present in the *adapted ABAC model*. For example, an RBAC static separation of duties (SSoD) constraint's context is the rbacDSML's *User stereotype* (synonymous to *subject*). If there are 20 subjects within the *adapted ABAC model*, the SSoD constraint will be evaluated 20 times, once for each subject.

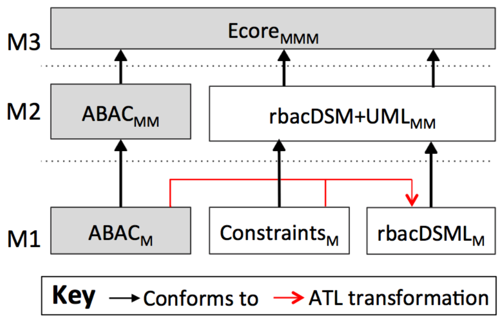


Figure 4.8: Transformation of ABAC to rbacDSML

For the prototype controller to make use of the rbacDSML verification tool, the *constraints model* and *adapted ABAC model* have to be transformed into an rbacDSML compliant UML model of RBAC. This is achieved by invoking an ATL model transformation program: ABAC+CON2RBACDSML, as shown in Figure 4.8. The transformation program combines the *adapted ABAC model* ($ABAC_M$) and the *constraints model* ($Constraints_M$), and outputs the rbacDSML model ($rbacDSML_M$).

Upon completion of the transformation, the model is verified against the constraints now woven within *adapted ABAC model*, in order to evaluate the model against the organisation's security requirements. The result is either a list of violated security requirements, together with their context elements. For example, if the RBAC SSoD constraint has been violated for subject *Bob*, rbacDSML returns *WF SSoD, Bob* as an element of the list. If no constraints have been violated, rbacDSML returns an empty list.

Each of the prototype's generated *adapted ABAC models* undergoes model verification. Should all of the adapted models return violations, the prototype assumes there are no candidate solutions that can mitigate the detected violations. The failure to handle an adaptation is logged by the controller for further action to be taken by human administrators.

4.3.6 Mitigating Behaviour (Planning and Execution)

Mitigation of behaviour is achieved through the selection and planning of an appropriate solution, and execution of the plan against the target system.

Solution Selection and Plan Generation

The prototype's implementation of solution selection and plan generation are simplistic approaches to what is a challenging problem, albeit simulates two important steps: selection of an appropriate solution from a candidate set of solutions, and creation of an executable plan in which to realise the selected solution.

Solution selection is implemented through a weighted calculation based on Strasburg et al.'s cost-sensitive modelling approach [136] (see Chapter 3.4.4). It aims to rank solutions based on a perception of cost that considers the trade-offs between observed violations and enactment of a solution:

$$sl_{cost} = \sum a_{cost} + sl_{impact} - sl_{goodness}$$

Solution cost (sl_{cost}) is the result between the negative consequence of the solution against the state of access, and the positive consequence of the solution in mitigating identified abuse.

The negative consequence of the solution is the aggregate cost of all of the solution's actions a_{cost} (as defined by the *solution policy*) and the cost of solution impact sl_{impact}. Solution impact is simply calculated as the number of non-malicious subjects that will lose access as a result of enacting the solution, multiplied by a base cost value. The base cost value reflects the cost of removing a single subject's access rights (as defined by the *solution's policy*).

The positive consequence of the solution is referred to as solution goodness ($sl_{goodness}$). This is calculated as the aggregated cost of all violations caused by malicious subjects that the solution can mitigate. The prototype's perception of a malicious subject is configurable. For instance, it may refer to all malicious subjects identified in abuse of a particular resource within a given period of time, or refer to all malicious subjects that have yet to be mitigated (i.e., where prior attempts at adaptation have failed).

Given a ranked set of solutions, where the solution that exhibits the least amount of cost is identified as the most appropriate solution, plan generation is enacted. Plan generation involves the ordering of actions of a selected solution for relevant target system services that an executor can follow. For example, generate authorisation assets, activate authorisation assets, and warn people of interest (i.e., subjects, issuers, sources of authority). Should a plan fail, a plan is generated and executed for the next ranked solution. However, if no solutions are available (or all have failed) then it is assumed that all solutions exhibit a greater consequence than the detected abuse.

Model Transformation and Execution

Given a selected *adapted ABAC model*, and a generated *plan*, the *executor* must realise these changes within the target authorisation infrastructure. Whenever the $ABAC_M$ undergoes adaptation, the $ABAC_M$ must be transformed back into the implementation specific models. Implementation specific models are then reflected within their respective services of the authorisation infrastructure. Transformation programs are beneficial here, as the SAAF controller is not concerned with how to adapt implementation specific models. Instead, the controller relies on the transformation program to realise the changes made against the ABAC model. This enables the use of model transformation programs to automatically generate relevant implementation specific models.

Two separate transformation programs have been created to enable this: *ABAC2PERMIS*, generating a new PERMIS policy model containg changes to access control rules within the *adapted ABAC model*, and *ABAC2LDAP*, generating a new LDAP model specifying new attribute assignments.

Once new implementation specific models have been produced, operators within the *executor* generate new authorisations assets and deploy these via relevant effectors, as shown in Figure 4.9. For example, changes to the $LDAP_M$ model are used to generate new X.509 (digital) certificates. Here, an operator referred to as the *CertificateEditor* (Section 4.3.2) takes attributes for adapted subjects, and generates and signs digital certificates on behalf of the subject's issuer. The certificate is then deployed via an *LDAP client*, updating the subject's access.

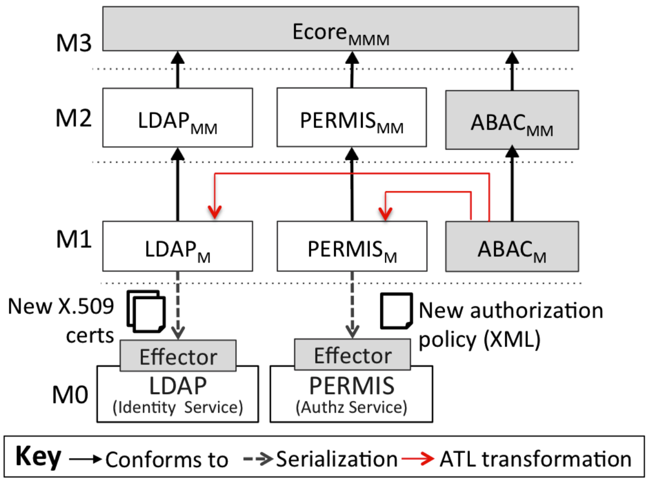


Figure 4.9: Model transformation of ABAC to implementation specific

4.4 Authorisation Infrastructure Integration

This section describes the integration of the prototype controller within a centralised RBAC authorisation infrastructure. The prototype is also capable of being deployed within other configurations of authorisation infrastructures, such as one that is maintained within a federated environment (see Chapter 5).

The centralised authorisation infrastructure, described in Figure 4.10, is representative of the prototype SAAF controller being deployed within a single organisation. The infrastructure utilises three servers: a resource server, an LDAP server, and an authorisation server.

The resource server hosts a set of resources (e.g., web applications). A policy enforcement point (PEP) is used by these resources in which to enforce access control decisions made by an external authorisation service. Upon the PEP enforcing a grant decision, a subject can access the resource within an authorised session. Probes can exist to observe changes within resources (i.e., a change due to subject activity), which provides context to a subject's authorised session.

The LDAP server is an example of an organisation's identity service, hosting an LDAP directory. The LDAP directory contains subject entries and subject role assignments (expressed as plain text role attributes, or within a binary attribute containing a signed digital certificate). Subjects authenticate with the LDAP directory in order to gain initial access to the organisation's resources. This is achieved either via a single sign on service, or via authentication procedures implemented directly within the resources. A probe on the LDAP server exists to monitor changes to the LDAP directory, generating *Subject Change* notifications

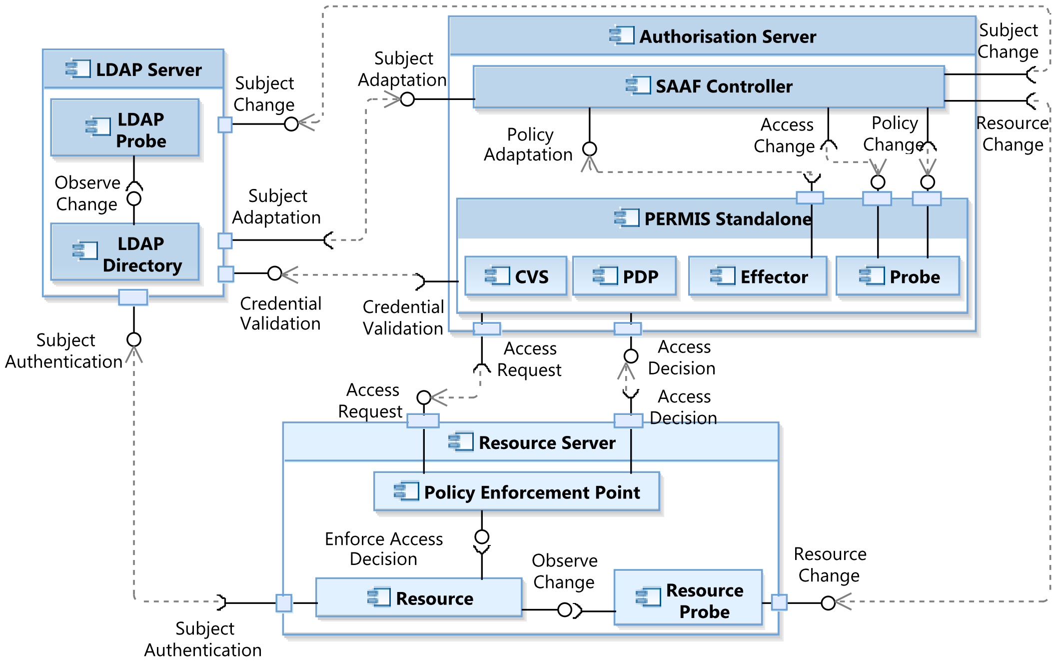


Figure 4.10: Centralised authorisation infrastructure

when a subject's access rights have been modified or created.

The authorisation server contains a PERMIS standalone authorisation service [114], and a deployment of the prototype SAAF controller. PERMIS protects an organisation's resources through the analysis of ABAC attribute - permission assignments and the request for access, via its policy decision point (PDP). A policy deployed by PERMIS in this deployment conforms to RBAC, and is also capable of expressing validation rules for subject role assignments (should a subject's access rights exist as digitally signed certificates).

Validation of subject access is performed by PERMIS's credential validation service (CVS), whereby a subject's set of access rights can be validated against the issuer of the access rights. In this architecture, PERMIS receives access requests that identify the subject by the subject's unique LDAP 'distinguished name' (DN), along with the resource and action the subject wishes to access. Using the subject's DN, PERMIS retrieves the subject's access rights directly from the LDAP directory, either in the form of a digital certificate or as unsigned attributes.

The SAAF controller is deployed on the authorisation server where it is best suited for control of the PERMIS authorisation service. It generates a model of access (using its ABAC metamodel) based on detection of *Subject Change* and *Policy Change*. Each change causes a model transformation to produce a new state of access (the prototype's *ABAC model*).

An effector allows for direct manipulation of the PERMIS authorisation service, whereby generated policies (by the prototype's *executor*) are stored in a persistent state, and activated via initiating a reboot of the PERMIS Standalone service[2]. To perform adaptations against the LDAP directory, LDAP client tools are used to update either plain text attributes, or deploy newly created digital certificates into LDAP binary attributes.

4.5 Preliminary Analysis of SAAF

In this section, the SAAF controller prototype is demonstrated through a scaled simulation inspired by a historic insider attack [27]. The prototype controller is shown considering several solutions while handling the simulated attack, whereby only verified solutions are enacted.

4.5.1 Chemical Researcher Insider Attack

In late 2005, a chemical company was a victim of an insider attack [27, 54] in which a single attacker carried out theft of intellectual property. As a consequence of the attack the company suffered $400 million dollars' worth of trade secrets being stolen [54]. The malicious insider is said to have downloaded 17,000 sensitive PDF documents from the company's *Electronic Library*, as well as 22,000 abstracts. The malicious insider was an employee of the chemical company, whereby it is assumed he had legitimate access rights to the *Electronic Library*.

Assumptions

It is assumed the chemical company operated an authorisation service to manage access to their *Electronic Library*, and utilised identity services to maintain access rights of their subjects. For the purpose of demonstrating an implementation of SAAF it is assumed that the chemical company implements RBAC (due to RBAC's popularity in industry) as their access control methodology. In this instance, subjects have relevant roles, such as *Researcher*, who have permissions, such as *Get Document from Electronic Library*. Finally, it is assumed the combination of authorisation services, identity services, and access control is one that

[2] A limitation in PERMIS is that it must be restarted to activate new policies at runtime.

can be instantiated within the centralised authorisation infrastructure, portrayed in Figure 4.10.

Attack Properties

The chemical company only identified the attack once the attacker had ended his contract and had begun work at a competing company. From the account of the attack it is difficult to surmise a complete picture in regards to detection. However, the properties of the attack are characterised in terms of the attacker's usage of the *Electronic Library* (i.e., technical factors):

- 17,000 PDF files and 22,000 abstracts were downloaded over a period of four months, and were not related to attacker's role;
- Abuse of access took place on-site during office hours;
- Files accessed were downloaded over several periods of 15 to 20 hours
- Total documents downloaded were 15 times greater than the next highest user of the *Electronic Library*.

These technical factors define observable activity within the company's *Electronic Library*. Notably, the number of documents downloaded is 15 times greater than the next highest user of the *Electronic Library*. This suggests that even within the upper limits of what could be considered as normal behaviour, the malicious insider's activity was significantly higher. As downloads were spread across 15 to 20 hour periods, it suggests that documents were downloaded in batches. As such, the malicious insider's activity profiles a high frequency of downloads per daily time intervals.

4.5.2 Adaptation Scenario

The chemical company insider attack is simulated within the self-adaptive authorisation infrastructure, described in Section 4.4. The authorisation infrastructure's state of access (pre-adaptation) is identified in Figure 4.11. At deployment-time, 8 users are active, with assigned roles held in the LDAP directory. The PERMIS authorisation service has one active access control policy, which states users with the role *Supervisor*, *Researcher* and *Administrator* can access the *Electronic Library_GetDoc* resource. Note in this case, multiple roles have access to the same permission expressed in the PERMIS access control policy.

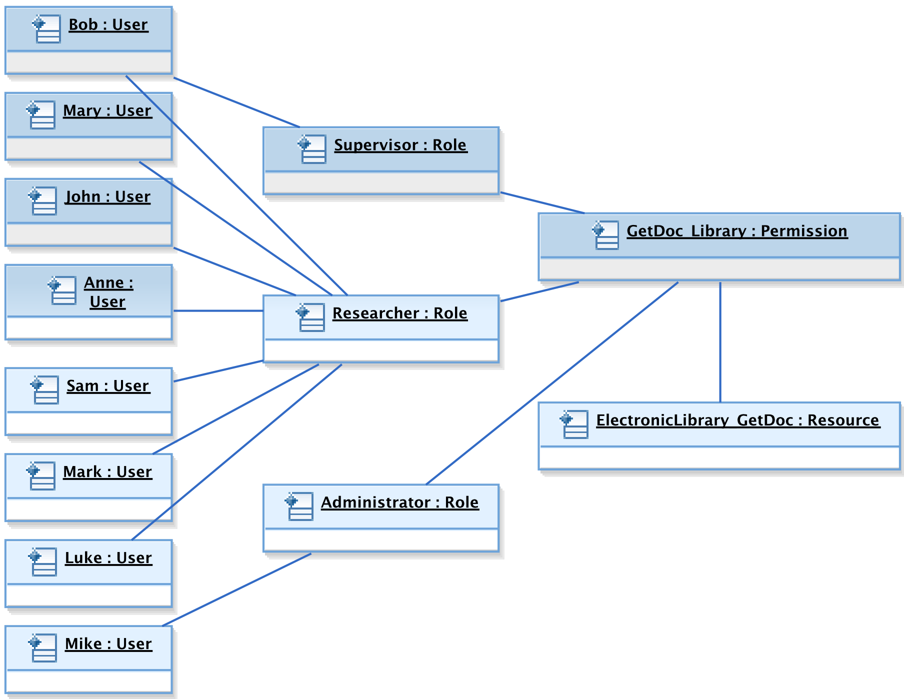


Figure 4.11: Deployed state of access (RBAC)

The properties of the chemical company case depict a long term attack, whereby a single user downloaded a high volume of documents over a period of 4 months. In addition, the detection of the attack suggests it was made through calculating the deviation of the attacker's historic usage of the *Electronic Library* to other users, where the attacker's usage was 15 times greater [27]. The attack is scaled down to simulate usage of the *Electronic Library* within a period of 4 hours (as opposed to the 4 months in which the attack was conducted), and instead of 22,000 documents, it is scaled to 240. It is also assumed that the acceptable number of downloaded documents within that period of 4 hours is 16, which is 15 times less than the scaled figure of 240 downloads.

Deployment

The self-adaptive authorisation infrastructure is hosted across two virtual machines, each with 1024MB of RAM and running Ubuntu 12.04.3 LTS. The virtual machines represent an identity service, containing an *LDAP* directory, and an authorisation service, containing a deployment of *PERMIS*. The SAAF controller is also deployed on the authorisation service machine, where it is best suited to managing PERMIS access control policies. Finally, the existence of the *Electronic*

Library is simulated via access requests made in the form of HTTP requests from a Windows 7 machine (2GB of RAM). Access is simulated through the use of an automated script, sending access requests to *PERMIS*, whereby user access rights are evaluated. Each granted request to download a document from the *Electronic Library* is seen as synonymous with a user downloading a document.

Domain Security Requirements

The domain represents the victim organisation (the chemical company), whereby the organisation owns the authorisation infrastructure, its deployed services, the prototype controller, and the protected resources (i.e., *Electronic Library*). The domain's requirements are necessary in governing the extent of adaptation, regardless of what malicious behaviour is detected, and are modelled as rbacDSML scenarios (adaptation constraints). In these cases, it may be that the chemical company is only willing to risk automated adaptation where only low level workers are impacted. To reflect these concerns, the following adaptation constraints are deployed, captured within SAAF's *constraints model*:

- **C1** *Administrator* role must maintain access to all resources (*Role Resource Scenario*)
- **C2** At least one user must be assigned the *Administrator* role (*User Role Scenario*)
- **C3** Each resource should be accessible by at least one user (*Resource Scenario*)

Despite some redundancy, these constraints aim to demonstrate different types of constraint scenarios to evaluate against SAAF's adapted models.

Identification

Considering the properties of the attack, the prototype controller is deployed with a pattern based behaviour rule. Should the prototype controller detect usage of the *Electronic Library* from users with the role of *Researcher* as greater than 3 times the frequency of average number of downloads (within the 4 hour interval), a violation is identified. In addition, to classify severe behaviour, a composite rule is applied, which indicates that after the first rule has been broken multiple times, the behaviour is severe enough to warrant adaptations to access control policies (i.e., adaptations that generate greater impact).

Solutions

The prototype controller is deployed with a set of solutions, tailorable to detected abuse. All of the below solutions are considered to be capable in resolving malicious behaviour detected by the pattern based behaviour rule expressed earlier. Solution *S1* indicates adaptation to the individual, whereas, solutions *S2* to *S5* indicates policy adaptation, impacting many individuals. They remain fixed throughout runtime.

- **S1** Remove all roles from ⟨user⟩
- **S2** Remove ⟨permission⟩ from ⟨role⟩
- **S3** Remove all permissions to ⟨resource⟩
- **S4** Remove all permissions from ⟨role⟩
- **S5** Remove all permissions from all roles

Execution

To demonstrate the full extent of adaptation and verification, the characteristics of the chemical company insider attack are simulated as a coordinated attack between 4 users with the *Researcher* role, with the intent to carry out IP theft against the *Electronic Library*. There are 4 stages of the attack. In each stage a new user is simulated to violate the prototype's behaviour rules, allowing the prototype to identify the malicious behaviour and respond to it accordingly. All but three *Researchers* and the one *Administrator* take part in the attack. As each stage is simulated, the number of solutions applicable to the behaviour may increase, indicating that the *Electronic Library* is under persistent attack.

The first stage demonstrates the user *Anne* breaking the behaviour rule by downloading a high number of documents at the start of the 4 hour attack period, using her assigned *Researcher* role. The second and third stage simulate users *John* and *Mary* carrying out similar activity to stage one, again within the same 4 hour window and using the *Researcher* role. Finally, the fourth stage simulates the user *Bob* breaking the same behaviour rule, using his *Researcher* role. Each stage considers a set of solutions, whereby the set of verified solutions is captured, and the result of the adaptation engine is shown in terms of a selected verified solution.

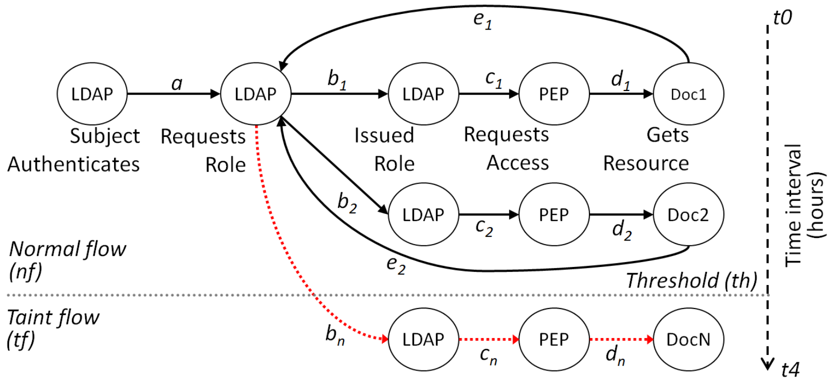


Figure 4.12: Stage 1 Taint Flow

As an example, Figure 4.12 conveys the steps that user *Anne* takes to carry out malicious behaviour. The *normal flow (nf)* of Anne's behaviour is comprised of her first authenticating her identity with the LDAP service (a). From here Anne is able to request (b) her role (in the form of an attribute). This role is then issued (c) and used in conjunction with a policy enforcement point (PEP) to gain access and retrieve (d) a document from the *Electronic Library*. Given that the perception of malicious behaviour is based on the frequency at which Anne downloads documents, Anne's *taint flow (tf)* (i.e., when Anne's behaviour is viewed as malicious) is identified once her total number of downloads crosses the prescribed *threshold* (i.e., 240) in a 4 hour time interval.

4.5.3 Adaptation Results

The following presents a high level summary of the adaptations performed by the SAAF controller prototype. A more detailed explanation of the adaptation results can be reviewed in Appendix A.8.

The attacks were simulated over a period of 4 hours, in accordance to the 4 stages described in Section 4.5.2. A set of solutions {*S1*, *S2*, *S3*, *S4*, *S5*} were deployed in the prototype controller, relevant to handling the pattern based behaviour rule. The solutions were chosen to demonstrate the verification of invalid and valid RBAC models at runtime. To gain a performance average for the response to each attack stage, the experiment was executed 30 times. For practical reasons, performance averages were obtained from simulating the attack in a reduced attack period of 5 minutes, where adaptation and verification results showed negligible difference to the 4 hour simulation.

Table 4.1 portrays the 4 stages of attack. In the first two stages, the prototype

Stage	User	Identified Solutions	Valid Solutions	Selected Solutions	Avg. Response Time (sec)	Std. Dev.
1	Anne	S1	S1	S1	18.70	1.11
2	John	S1	S1	S1	10.74	0.64
3	Mary	S1, S2, S3, S4, S5	S1, S2, S4	S1	45.12	1.30
4	Bob	S1, S2, S3, S4, S5	S1, S2, S4	S2	44.79	1.31

Table 4.1: Verification and adaptation results

controller considers the malicious behaviour to be minor, only identifying solution *S1* as a relevant solution. At this point solution *S1* has been tailored to the role the user is activating (*Researcher*) and the resource they are accessing (*Electronic Library*). In both stages, the tailored solutions result in a verified RBAC model since there is no conflict with the 3 constraints described in Section 4.5.2. Solution *S1*, which removes *Anne* and *John*'s access rights, thus their ability to access the *Electronic Library*, is chosen as it is the only valid solution available. The solution is realised by transforming the adapted RBAC model into an LDAP user model, which is then used to update the current state of access rights within the LDAP directory. Adaptation, from detection to response within the authorisation infrastructure, took an average of 18.7 seconds to complete in the first stage, and an average 10.74 seconds to complete in the second stage. The difference in time is assumed to be a result of the Java virtual machine warming up, in which both the prototype controller and verification tool is executed on.

Once the third stage of the attack was executed, the prototype controller identified that there was persistent malicious activity regarding the role of *Researcher* and the resource *Electronic Library*. As a result, the SAAF controller selects a wider set of solutions {*S1*, *S2*, *S3*, *S4*, *S5*}. In this case, multiple adapted RBAC models are created in accordance to the tailored solutions. These are verified using the rbacDSML tool; resulting in solutions *S3* and *S5* as invalid. This is due to solution *S3* removing all access to the *Electronic Library*, violating adaptation constraints *C1*, and *C3*. The same violation of constraints *C1* and *C3* occurred when the SAAF controller deactivated all permissions within the RBAC model, for solution *S5*.

Solution *S1* is ultimately chosen as the most appropriate solution, given the severity of the attack and the solutions available. This is a result of the prototype controller calculating that adaptations *S2* and *S4* would cause greater impact than allowing the attack to continue (Appendix A.8). As with stage 1 and 2, *Mary's* access right to the *Electronic Library* has now been removed, preventing her from further access. The adaptation of stage 3, from detection to response, took a total average of 45.12 seconds. This is due to additional RBAC models undergoing adaption, transformation into a rbacDSML UML model, and verification using the rbacDSML verification tool.

Finally, in the last stage of the attack, the same solutions are identified and verified similarly to stage 3. However, solution *S2* is selected. Solution *S2* removes the permission that has been abused from the *Researcher* role, preventing any future user with the same role executing that permission. The solution is realised by transforming the adapted RBAC model into a new PERMIS model, which is then deployed as a new access control policy. This has a negative consequence on the remaining three users with the *Researcher* role, as they are no longer capable of accessing the *Electronic Library*. However, the prototype controller has selected this solution due to the persistent attacks against the *Electronic Library*, and its resulting cost sensitive calculation (Appendix A.8). A contributing dimension of the cost sensitive approach, is that in stage 4, there are more malicious subjects that own the *Researcher* role when compared to non-malicious subjects.

In summary, subject adaptation has had the consequence of preventing a subject's ability to be issued with the necessary roles to request access to a resource. This is essentially blocking the flow of information at point b, conveyed in Figure 4.12. In addition, policy adaptation has had the consequence of preventing all *Researcher* subjects ability to gain access due to removal of permissions. This is achieved by blocking the flow of information at point d, conveyed in Figure 4.12.

4.5.4 Scalability of Verification

To demonstrate the scalability of verification, a set of randomly generated RBAC models, increasing in size of model elements, were verified. For consistency, each model generated contained a ratio of 50% subject elements, 15% role elements, 10% permission elements, 10% resource elements, and 15% constraint scenario elements. The size of each model generated initially was set to 10 model elements, increasing to 500, 1000, and then by intervals of 1000.

The verification of each model was repeated 10 times to obtain an average and standard deviation. The scalability results are shown in Figure 4.13. As the ABAC model size increased, the verification times were shown to follow a linear pattern. Note that these performance measures only capture the time it takes to complete a verification cycle, and does not represent a complete adaptation cycle.

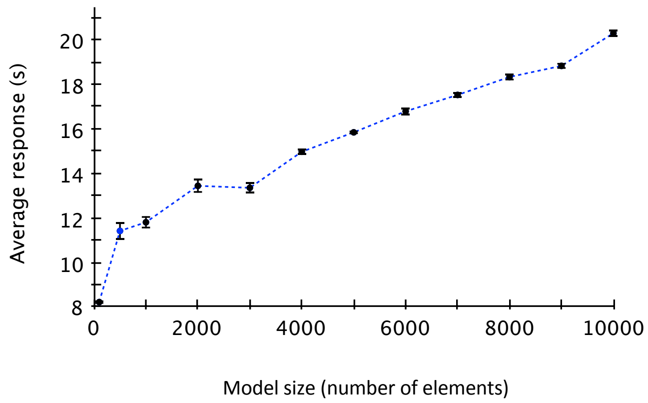


Figure 4.13: Scalability of model verification

4.5.5 Prototype Discussion

The prototype has demonstrated the detection, verification, and mitigation of a simple abuse scenario. Notably, it has demonstrated the verification of RBAC models, which has provided assurances against the deployment of inappropriate solutions at runtime (within the target authorisation infrastructure). In addition, the results have shown the escalation of abuse to be met with the verification and enactment of stronger solutions, ultimately stopping the collaborated attack through an adaptation to the access control policy. As such, the prototype has been shown to successfully perform adaptation in regards to subject access rights, and authorisation policies, mitigating detected abuse through persistent prevention and limitation of access.

One limitation with respect to verification, is that mandatory constraints must always be verified per adaptation. In some cases it can be argued that different levels of verification are needed. For example, given a minor attack on the *Electronic Library*, the organisation may require a constraint guaranteeing at least one researcher maintains access to the resource. However, should the attack continue and become severe, the deploying organisation may require that the constraint is no longer applicable, since the *Electronic Library* has suffered a severe attack. One solution to this is to classify identified attacks against available constraints, indicating which constraints should be verified per attack (in addition to mandatory constraints).

In addition, there are several limitations regarding the prototype itself. Notably, the prototype's MAPE-K components are very much interlinked, limiting the ability to employ modular solutions within each stage of the feedback loop (e.g., alternative behaviour detection techniques). However, in order to achieve a proof of concept for self-adaptive authorisation, this interlinked approach has been intentional. In addition, each stage of SAAF's feedback loop offers a number of challenges, with an expansive problem domain to explore. As such, naive mechanisms were employed to demonstrate some stages of SAAF's feedback loop as to evaluate the holistic goal of SAAF, which is the feasibility of self-adaptive authorisation.

4.6 Summary

In summary, this chapter has described the implementation of a prototype of the Self-Adaptive Authorisation Framework (SAAF) controller, and its integration within a centralised RBAC authorisation infrastructure. The implementation conforms to the conceptual design of SAAF, presented in Chapter 3. The core functionality of SAAF has then been exemplified through a preliminary experiment, essentially *closing the loop*, and demonstrating self-adaptive authorisation.

A key contribution of this chapter is the use of model generation, model transformation, and model verification. This has been used to present the runtime model driven self-adaptive approach to access control, capable of mitigating malicious behaviour whilst providing assurances to adaptations. Second to this, it is the use of model transformation to enabling adaptation of many RBAC and ABAC implementations of identity and authorisation services. This has been shown through the use of several model transformation programs that transforms implementation specific models of two existing technologies into a homogeneous model of access. The use of homogeneous and implementation specific models has allowed for reasoning and analysis of access in a common format, whilst realising adaptations in conformance to a given implementation's view of access.

Whilst the preliminary experiment demonstrated the operation of SAAF at a high level, further evaluation is required. As such, the following two chapters seek to evaluate SAAF under different conditions and environments. Chapter 5 presents an approach in formally identifying 'malicious change' to simulate and evaluate the runtime mitigation of abuse of access within a federated ABAC authorisation infrastructure. Finally, Chapter 6 seeks to observe the consequence of adaptation where the SAAF controller prototype is shown mitigating abuse exhibited by real users.

Chapter 5

Simulating Insider Threat

5.1 Introduction

This chapter adopts a formal approach to evaluating the Self-Adaptive Authorisation Framework (SAAF). The approach aims to evaluate, via simulation, SAAF's robustness under repeatable conditions and environment change.

Existing approaches in evaluating self-adaptive software systems require the observation and analysis of quality dimensions that are common to self-adaptive systems [142]. Many approaches rely on the observation of performance as a measure of success. However, evaluating performance alone provides a limited view of success, in particular in some application domains where the social impact of adaptation must be considered. This is the case for self-adaptive authorisation, where adaptation may result in undesirable states, including the loss of access to critical resources.

A common way for evaluating self-adaptive systems is through the use of case studies. Case studies can be used to represent environment and system change, which are expected to stimulate adaptation, thus providing the basis for evaluating the impact of adaptation. For example, Rainbow's 'ZNN' case study [53] is used to evaluate architectural-based adaptation.

This chapter uses a fictitious case study describing a set of insider attacks within a federated environment. The case study is applied to a self-adaptive authorisation infrastructure using SAAF, and it aims to answer the following questions: Can SAAF detect and mitigate malicious behaviour in federated deployments? Does SAAF impact the performance and availability of its target authorisation infrastructure? Finally, can SAAF detect and mitigate malicious behaviour in a consistent manner?

The contribution of this chapter is the evaluation of SAAF, using as a basis a fictitious case study. Specifically, the evaluation demonstrates SAAF operating within a federated environment, and how SAAF handles malicious behaviour given the existence of non-cooperating third party organisations. Second to this is the application of changeload [24] in formally describing malicious behaviour in the context of authorisation infrastructures. Changeload is a means to describe a set of environment and system change types that can be instantiated to stimulate self-adaptation. In this chapter, a malicious changeload for the fictitious case study is defined to simulate the abuse of access. The malicious changeload is then simulated to evaluate SAAF under various operational conditions.

The rest of this chapter is structured as follows: Section 5.2 positions a motivating case study used as a basis for evaluation. Section 5.3 provides the definition of changeload in the context of authorisation infrastructures. Section 5.4 describes the malicious changeload of the case study. Section 5.5 describes a set of experiments and results that observes the runtime stimulation of malicious changeload, and adaptation of a target system. Section 5.6 reflects on the outcome of the experiments, along with the benefits and challenges of self-adaptive authorisation. In Section 5.7 the chapter is summarised. Finally, in Appendix B additional information and results are provided that support this chapter.

5.2 Case Study: LGZLogistics

Given the challenges in obtaining detailed data on actual cases of insider attacks, this fictitious case study draws upon several historical cases discussed in the CERT guide to insider threat [27]. The type of malicious behaviour depicted in this case study is categorised as data theft attacks. These attacks are performed internally to a fictitious logistics company, called *LGZLogistics*, representing a service provider and identity provider within a federated authorisation infrastructure (see Chapter 2.2.5). Malicious behaviour is conducted by disgruntled employees of the logistics company, as well as employees of an external *Trusted Business Partner* (TBP) [27]. The role of a TBP is key to this case study, as it is representative of the relationship a service provider organisation has with an identity provider organisation (e.g., *LGZLogistics* trusts the TBP to provide IT help desk services).

The case study focuses on two areas of insider threat that organisations are highly vulnerable to: the abuse of user access rights by employees of the organisation, and the abuse of access rights by TBP organisations [104].

5.2.1 Context and Architecture

LGZLogistics portrays a small to medium sized company of 1000 employees, ten of which are IT staff that support and administer a set of protected resources. These resources are protected via an instantiated Attribute Based Access Control (ABAC) model, in the form of subject attribute assignments within identity services, and an access control policy within an authorisation service.

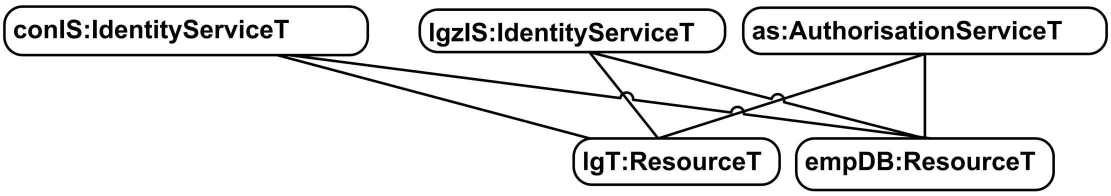


Figure 5.1: LGZLogistics authorisation infrastructure architecture

LGZLogistics maintains a SimpleSAMLphp [131] identity service lgzIS to authenticate its subjects (employees) and issue access rights (as credentials). The organisation also maintains a PERMIS standalone authorisation service as [114], to authorise subject access to its resources. These resources include an employee database empDB, and a bespoke logistics tool lgT. The employee database contains personnel information about the logistic company's employees, which is required for general IT help desk enquires.

LGZLogistics utilises the authorisation service as to authorise access for its own subjects, as well as subjects from a second offshore contractor organisation (a TBP). *LGZLogistics* trusts the contractor organisation to issue access rights to their subjects as part of a business contract, for providing IT help desk services.

As such, the contractor organisation also operates a SimpleSAMLphp identity service conIS that manages and releases its own employees' access rights to requesting service providers (i.e., *LGZLogistics*). As part of their contract, subjects from the contractor organisation are permitted access to empDB to facilitate help desk duties. Access for subjects from either identity service is obtained as follows:

1. A subject attempts to perform an action on a resource;
2. The resource enacts a policy enforcement point (PEP) that requires the subject to authenticate with their identity service (i.e., lgzIS or conIS);
3. Upon authentication, a short term credential is released to the resource's PEP, denoting a signed set of subject attributes (e.g., a SAML assertion [99]);
4. The PEP forwards the subject's issued credential to the authorisation service as, which validates the contents of the credential to ensure attributes released have been issued by a trusted identity provider;

5. If valid, the attributes are used to request access via the authorisation service `as`, along with the resource, and action to be performed.
6. Lastly the authorisation service `as` decides whether to grant access in accordance to its authorisation policy.

5.2.2 Access Control Model

LGZLogistics employ an ABAC methodology to protect its resources. As such, an instantiation of ABAC considers the subjects of *LGZLogistics* and the subjects of the contractor organisation. Each set of subjects have a permissible scope of access rights that can be assigned to them.

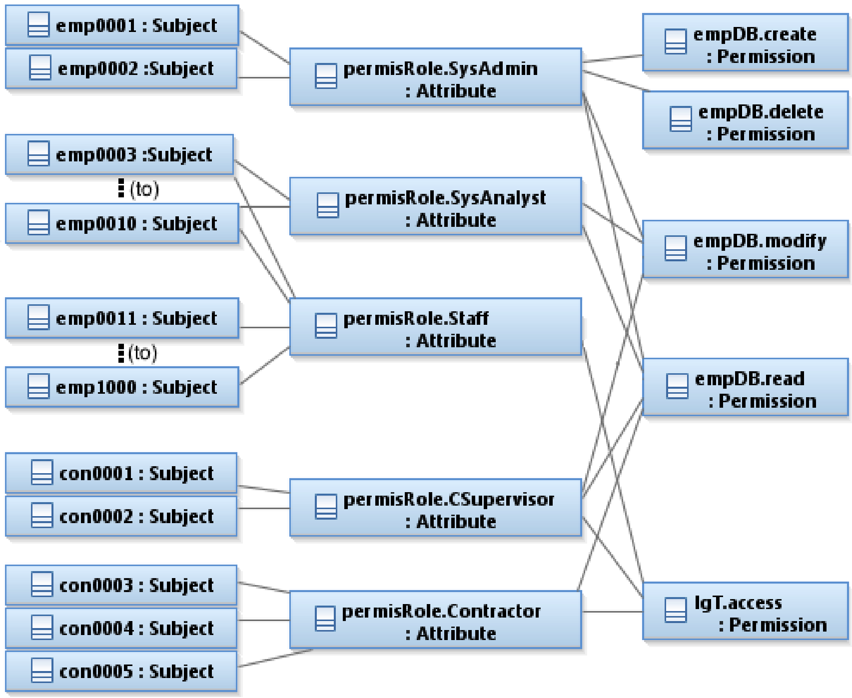


Figure 5.2: LGZLogistics subject attribute permission assignments

Figure 5.2 defines access in the form of a class diagram. There are five 'permisRole type' attributes [31] (specific to the the PERMIS standalone authorisation service) with corresponding values. Subjects are assigned these attributes, which can then be used to invoke permissions.

In addition to the subject attribute assignments and attribute permission assignments, *LGZLogistics* define a set of valid attribute assignment rules (within its authorisation policy). Figure 5.3 specifies what attributes an

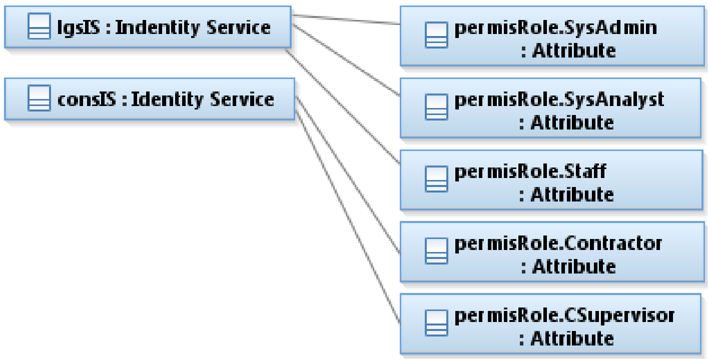


Figure 5.3: LGZLogistics valid attribute assignments

identity provider is trusted to issue on behalf of its employees. For example, *LGZLogistics* identity provider lgzIS is trusted to assign attributes $\langle permisRole, SysAdmin\rangle$, $\langle permisRole, SysAnalyst\rangle$, and $\langle permisRole, Staff\rangle$ to its employees. The contractor organisation identity provider conIS can only assign attributes $\langle permisRole, ContractorSupervisor\rangle$ and $\langle permisRole, Contractor\rangle$.

5.2.3 Subject Behaviour

This section identifies typical subject behaviour for the day to day operations of *LGZLogistics*, as well as a malicious behaviour scenario.

Typical Behaviour

The following describes a base line of subject behaviour, detailing the average usage of the authorisation infrastructure likely to occur in the day to day operations of *LGZLogistics*:

- Each staff member requests 'access' to the lgT resource on average two times per day;
- Contractors receive on average fifty calls per day, each call requiring one 'read' access to empDB;
- On average, one in five calls require access to 'modify' the empDB, which can only be performed by a contractor supervisor, systems analyst, or system administrator;
- On average, each system analyst performs ten 'read' requests, and five 'modify' requests per day to the empDB;
- A system admin performs on average one 'read', 'modify', 'delete', and 'create' request per day to the empDB.

Malicious Behaviour Scenario

The logistics company is victim to an insider attack, largely as a result of a catalyst event [98]. The catalyst event refers to a notification to several key IT workers that they have been selected for job redundancy.

A systems analyst that has been selected for redundancy is unhappy about the decision, and attempts to damage the company in three ways. The first is to attack the empDB resource by randomly corrupting employee records, invoking the permission 'modify' empDB. The second is an attempt to disrupt access to the lgT resource, essentially flooding the resource by initiating numerous authorised sessions. The final attempt is socially motivated, whereby the analyst, who works closely with employees of the contractor organisation, informs them that *LGZLogistics* is going to cancel their contract to cut costs.

A contractor supervisor, now fearing job redundancy, decides to steal data from the empDB resource. The supervisor has links with the internet underground [27], and is aware of anonymous buyers looking for data fit for identify theft. By persuading his peers, three other contractors decide to collaborate in stealing employee information from the empDB, to sell it to the internet underground.

5.3 Defining Malicious Changeload

This section presents a definition of a malicious changeload model [24] related specifically to malicious behaviour in the context of authorisation infrastructures. Essentially, it applies Cámara et al.'s existing definitions of a changeload model [24] (which is specific to architectural-based self-adaptation) to authorisation infrastructures. As such, Cámara et al.'s definitions are included in order for this chapter to be self-contained. However, some definitions have been slightly modified in order to take into consideration malicious behaviour, supported with examples relating to the *LGZLogistics* case study.

Cámara et al.'s changeload model was chosen in order to concretely define the scope of change within an authorisation infrastructure. In addition, it enables the definition of non-conventional states that describe a system with ongoing malicious activity, where a set of events can trigger self-adaptation. Through the specification of changeload, it is intended to concisely describe case studies of insider threat within authorisation infrastructures, for the purpose of simulating insider threat, and evaluating threat mitigation (i.e., adaptation).

5.3.1 System and Environment Models

Following Cámara et al.'s approach, the *LGZLogistics* authorisation infrastructure is formally defined in terms of an architecture model (Figure 5.4).

Definition 17 (Architecture Model [24]) *"An architecture model is a tuple* $\mathcal{A} = (\mathcal{T}, \mathcal{G}, \Gamma)$*, where:*

- $\mathcal{T} = \mathcal{T}_{comp} \cup \mathcal{T}_{conn}$ *is a set of architectural types, where* $\mathcal{T}_{comp}$ *and* $\mathcal{T}_{conn}$ *are the sets of component and connector types, respectively.*
- $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ *is a graph describing a system configuration, where:*
  - $\mathcal{N}$ *is a set of nodes, where a typing* Λ *assigns an architectural type* $\Lambda(\mathfrak{n}) \in \mathcal{T}$ *to every* $\mathfrak{n} \in \mathcal{N}$*.*
  - $\mathcal{E}$ *is a set of edges, where each one of them consists of an unordered pair of nodes* $(\mathfrak{n}, \mathfrak{n}')$*, such that* $\Lambda(\mathfrak{n}) \in \mathcal{T}_{comp}$ *and* $\Lambda(\mathfrak{n}') \in \mathcal{T}_{comp}$*.*
- Γ *is a function that assigns a set of properties* $\Gamma(\mathfrak{ty})$ *to each architectural type* $\mathfrak{ty} \in \mathcal{T}$*."*

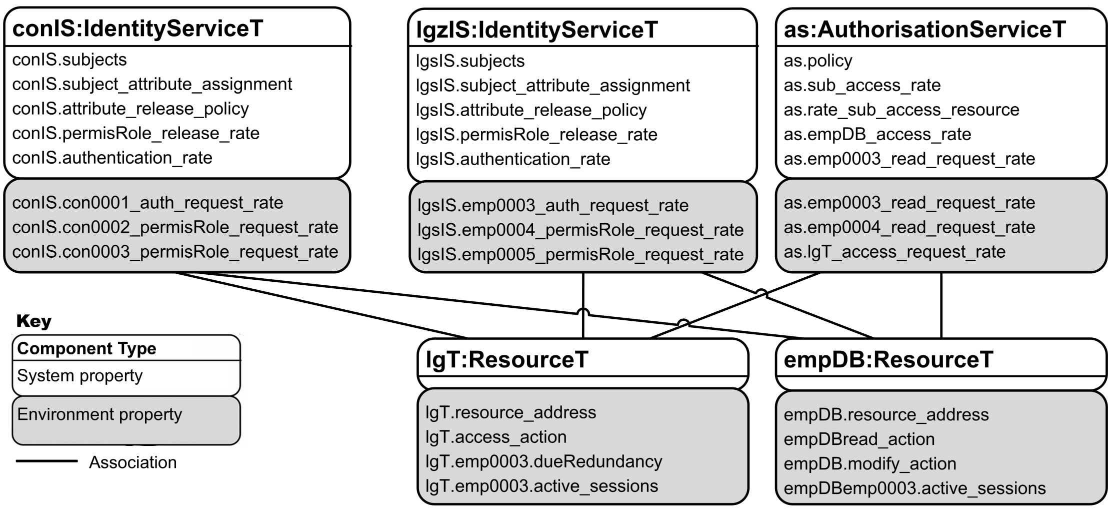


Figure 5.4: Example architecture model for an authorisation infrastructure.

In the case of SAAF, an *access control model* provides the necessary relations between components of an architectural model (i.e., how a subject of an identity service component can access a resource component). Despite this, the use of an architectural model in this case is beneficial in defining properties of a system and the environment. It enables the specification of properties of the system that describe an authorisation infrastructure's runtime parameters and workload, and properties of the environment that characterises the operational conditions imposed on an authorisation infrastructure. These properties are said to be contained within a system model and environment model, derived from the architecture model.

Definition 18 (System Model [24]) *A system model associated with an architecture model* $\mathcal{A} = (\mathcal{T}, \mathcal{G}, \Gamma_{sys})$ *is a function* Γ_{sys} *that assigns a set of system properties* $\Gamma_{sys}(\mathfrak{ty})$ *to each architectural type* $\mathfrak{ty} \in \mathcal{T}$

Definition 19 (Environment Model [24]) *"An environment model associated with an architecture model* $\mathcal{A} = (\mathcal{T}, \mathcal{G}, \Gamma_{env})$ *is a function* Γ_{env} *that assigns a set of environment properties* $\Gamma_{env}(\mathfrak{ty})$ *to each architectural type* $\mathfrak{ty} \in \mathcal{T}$*".*

Example 1 *Figure 5.4 displays an architecture model of the LGZLogistics authorisation infrastructure, where the set of architectural types is* $\mathcal{T}$*={IdentityServiceT,AuthorisationServiceT,ResourceT}. Examples of system properties include* Γ_{sys}*(AuthorisationServiceT)={policy,sub_access_rate}. Environment properties (displayed inside the grey boxes) include* Γ_{env}*(AuthorisationServiceT)={sub_access_req_rate, lgT_access_request_rate}, and* Γ_{env}*(ResourceT)={activeSessions,latency}.*

There can be a plethora of properties contained within both system and environment models, where such properties are dependent to a given deployment of an authorisation infrastructure and its protected resources.

5.3.2 System and Environment State

Definition 20 (State Condition [24]) *"A state condition is a tuple* (A, B, V_A, V_B) *that corresponds to the description of the evolution of a set of (either system or environment) properties over time:*

- $A = \langle a_1, \ldots, a_k \rangle$ *is a vector of* attributes *that enumerates the specific properties of interest (variables) for the (environment or system) state;*
- $B = \langle b_1, \ldots, b_k \rangle$ *describes the dynamics of the attributes in A (how they evolve over time,* e.g.*, through a polynomial, exponential, or step function);*
- $V_A = \langle v_{A1}, \ldots, v_{Ak} \rangle$ *is a vector of attribute values instantiating the attributes in A;*

- $V_B = \langle v_{B1}, \ldots, v_{Bk} \rangle$ *is a vector of behaviour instances of the elements in* B (i.e., *a behaviour instance* v_{Bi}, $i \in \{1, \ldots, k\}$ *is a function of type* $b_i \in B$, *describing the evolution over time of the attribute* $v_{Ai} \in V_A$*).”*

System State

The system state (Sys_{state}), defined in terms of a state condition, captures the value of system properties and the execution of services at a given moment in time. For example, the authentication of subjects, the release of subject attributes, the validation of subject attributes, and the authorisation of subject access. It is seen as a snapshot of the system, whereby multiple changes may have already occurred. A typical system state (Sys^t_{state}) assigns values to the identified properties, dependent on the subjects and protected resources that exist within the authorisation infrastructure (e.g., a subject's assigned privileges, what permissions exist to protect a resource).

Example 2 *In this example, two system attributes are identified that denote execution of the authorisation infrastructure: rate of attribute releases (of any kind) from the identity service* lgzIS, *and rate of successful read requests per interval to* empDB. *For typical execution of the identity service* lgzIS, *there is a constant throughput of one attribute release per minute. For typical execution of the authorisation service* as, *there is a constant throughput of three successful access decisions to* empDB.

- $A = \langle lgzIS.sub\_attr\_release\_rate,\ as.empDB\_read\_rate \rangle$
- $B = \langle constant\_function,\ constant\_function \rangle$
- $V_A = \langle 1,\ 3 \rangle$
- $V_B = \langle \theta_{lgzIS}(t) = 1/min,\ \theta_{as}(t) = 3/min \rangle$

Example 3 *In this example, two system attributes are identified that denote authorisation assets. Authorisation assets govern the outcome of execution within system components, such as whether an access request is granted by an authorisation service. For example, the authorisation policy in the authorisation service* as *is identified as* AP_1*. These are composite attributes, whereby* AP_1 *is a tuple of vector attributes.*

- $A = \langle as.policy,\ lgzIS.emp0003.permisRole,\ lgzIS.emp0999.permisRole \rangle$

- $B = \langle constant\_function,\ constant\_function,\ constant\_function \rangle$
- $V_A = \langle AP_1,\ SysAnalyst,\ Staff \rangle$
- $V_B = \langle \theta_{as.policy}(t) = AP_1,\ \theta_{as}(emp0003.permisRole) = SysAnalyst,$
  $\theta_{as}(emp0999.permisRole) = Staff \rangle$

Environment State

The environment state Env_{state}, defined in terms of a state condition, captures operational conditions of external systems and users that interact with the authorisation infrastructure (Chapter 3, Figure 3.1). This includes conditions, such as the rate of access requests by subjects, or number of active sessions in resources. Building a perception of Env_{state} is essential to identifying states that exhibit malicious behaviour (e.g., subjects exhibiting excessive deviation from normal activity).

Example 4 *In this example, three environment attributes are identified that denote operational conditions: (i) the number of active sessions in* empDB*, (ii) the rate of authentication requests made by all subjects against the identity service* lgzIS*, and (iii) the rate of access requests to access the resource* empDB *to authorisation service* as*. The values associated with these operational conditions are, respectively, five active sessions, a throughput of one authentication requests per minute, and a throughput of three access requests per minute.*

- $A = \langle empDB.active\_sessions, lgzIS.sub\_authentications\_req\_rate,$
  $as.empDB\_read\_req\_rate \rangle$
- $B = \langle constant\_function,\ constant\_function,\ constant\_function \rangle$
- $V_A = \langle 5,\ 1,\ 3 \rangle$
- $V_B = \langle \theta_{empDB}(t) = 5,\ \theta_{lgzIS}(t) = 1/min,\ \theta_{as}(t) = 3/min \rangle$

5.3.3 Operational Profiles

An authorisation infrastructure can be considered to be in one of two types of states, a conventional operational state, or non-conventional operational state [24]. In the context of this work, a conventional operational state refers to a state where there is no ongoing abuse of access rights. A non-conventional operational state refers to a state where there is ongoing abuse of access rights.

Definition 21 (Conventional Operational Profile [24]) *The conventional operational profile (COP) of a system is the region of the state space $S = [\mathbb{R}^n]_X$ where no anomalies related to malicious behaviour hold:*

$$COP = \{s \in S \mid \forall\, \alpha \in AB,\ s \nvDash \alpha\}$$

AB is the set of all possible abuse of access that a system state can experience, and S is the region of state space that does not hold any cases of abuse.

Example 5 *A conventional operational profile is described as a set of states that does not contain any known patterns of abuse of access (i.e., violations).*

$$COP = \{s \in S \mid s \models \neg(empDBViolation)\}$$

A violation describes a predicate, that if true, denotes malicious behaviour within the environment state.

Definition 22 (Non-conventional Operational Profile [24]) *A non conventional operational profile (NCOP) associated with malicious behaviour, is the region of the state space $S = [\mathbb{R}^n]_X$ where the state holds at least one case of abuse α:*

$$NCOP = \{s \in S \mid \exists\, \alpha \in AB,\ s \models \alpha\}.$$

Example 6 *A non-conventional operational profile is described as a set of states that contain one or more occurrences of malicious behaviours. In this case, a violation (empDBViolation) denotes a specific violation in access to the* empDB*.*

$$NCOP_{empDBViolation} = \{s \in S \mid s \models empDBViolation\}$$

The violation empDBViolation is focused on determining if any particular subject is requesting access to the empDB *resource in a rapid manner. A subject that requests access to* empDB *at a rate ($subAccessReqRate_{empDB}$) greater than a maximum prescribed rate ($maxSubAccessReqRate_{empDB}$) is considered to be malicious.*

$$empDBViolation = subAccessReqRate_{empDB} > maxSubAccessReqRate_{empDB}$$

5.3.4 Change Types and Changes

This section describes the relevant change types applicable to an authorisation infrastructures, followed by example instantiations of change types, referred to as changes.

Change Types

Change types affect either identity services or authorisation services, which are characterised as part of an authorisation infrastructure, or its environment, consisting mainly of protected resources. Change types are defined as a vector of 'attributes' that describe a change and the dynamics of a change.

Note that the domain of authorisation infrastructures refer to 'attributes' as a piece of information that expresses something about the subject or the current conditions within an accessed resource. This is not to be confused with attributes of a formal model of change (i.e., changeload). However, authorisation attributes can exist as vector attributes within such a formal model.

Definition 23 (Change Type [24]) *"A change type, given a set of architectural types $\mathcal{T}$, is defined as a tuple (src, A, B) that characterises a change, where:*

- *$src \in \mathcal{T}$ identifies the* source *of the change;*
- *$A = \langle a_1, \ldots, a_k \rangle$ is a vector of* attributes *that hold information about the specific properties (variables) associated with the change type;*
- *$B = \langle b_1, \ldots, b_k \rangle$ describes the dynamics of the attributes in A (how they evolve over time,* e.g.*, through a polynomial, exponential, or step function)."*

In application to authorisation infrastructures, a change type describes an observable event within identity services, authorisation services, or protected resources. Essentially, the observation of such change will have a consequence on properties contained within the system and environment model.

Definition 24 (Environment Change Model [24]) *"An environment change model $\mathcal{CM}_{env}$ is a set of change types applicable to the environment properties (Γ_{env}) of a system family with some degree of commonality (* e.g*, common subset of architectural types)."*

Example 7 *In the following, several examples of low level environment change types are exemplified, depicting the process of a subject requesting access to a resource. The instantiation of these change types will have a consequence on one or more environment properties.*

(i) ***Authentication request type*** *captures (within an identity service) the identity service receiving a request for authentication of a user.*

$$\boldsymbol{auth\_request\_type} = \langle identity\_service,\\ \langle authRequest(username, password)\rangle,\\ \langle event\rangle\rangle$$

(ii) ***Attribute release request type*** *captures a request received by the identity service made by a service provider for a subject's identity attributes.*

$$\boldsymbol{attr\_release\_request\_type} = \langle identity\_service,\\ \langle attrRequest(identity,\\ \langle iAttribute\_type_1, ..., iAttribute\_type_n\rangle, target)\rangle,\\ \langle event\rangle\rangle$$

$$\boldsymbol{identity} = \langle identity\_type, identity\_value\rangle$$

It describes the request of a service provider (target) for a set of identity attributes (iAttribute_type) that have been issued to a subject (identity). The set of attributes requested can be a null set, therefore requesting all releasable attribute types for the subject identity. Note that an identity is referred to by a type of identifier and a value. For example, identity_type may be an LDAP distinguished name.

(iii) ***Credential validation request type*** *is the receipt of a credential validation request within an authorisation service.*

$$\boldsymbol{cred\_validation\_request\_type} =\\ \langle auth\_service,\\ \langle valRequest(identity, issuer, \langle iCondition_1, ..., iCondition_n\rangle,\\ \langle iAttribute_1, ..., iAttribute_n\rangle)\rangle,\\ \langle event\rangle\rangle$$

It contains attributes issued by a given identity provider (issuer) for a requesting subject, detailing a request to validate a subject's attributes. A set of conditions specified by the issuer can also be contained, whereby a condition refers to a type / value tuple, such as a single use declaration, or validity time. A credential validation request can either push the subject's known attributes, or (given a null set) require the authorisation service to pull the subject's known attributes from the subject's identity provider. In the latter case, the authorisation service invokes an attribute release request.

(iv) ***Access request type*** *is the request, received by an authorisation service, and made by a resource on behalf of a subject.*

$$\begin{aligned}\textbf{\textit{access\_request\_type}} = \langle & \textit{auth\_service}, \\ & \langle \textit{accessRequest}(\langle \textit{iAttribute}_1, ..., \textit{iAttribute}_n \rangle, \textit{resource}, \\ & \quad \textit{action}, \langle \textit{rAttribute}_1, ..., \textit{rAttribute}_n \rangle, \textit{identity}) \rangle, \\ & \langle \textit{event} \rangle \rangle\end{aligned}$$

$$\textbf{\textit{iAttribute}} = \langle \textit{iAttribute\_type}, \textit{iAttribute\_value} \rangle$$

$$\textbf{\textit{rAttribute}} = \langle \textit{rAttribute\_type}, \textit{rAttribute\_value} \rangle$$

The request contains 1) the subject's identity attributes (iAttribute), 2) the resource and action to be carried out by the subject, 3) a set of resource environment attributes (rAttribute) provided by the resource (e.g., $\langle \textit{timeOfDay\_type}, 11\textit{am} \rangle$), and 4) the requesting subject's identity.

(v) ***Resource action step type*** *models an action that has occurred within any protected resource. The type is generic as resources are generally unique to the organisation and their purpose, unlike with an authorisation service type that exists to fulfil access control requirements.*

$$\begin{aligned}\textbf{\textit{resource\_action\_step\_type}} = \langle & \textit{resource}, \\ & \langle \textit{rAttribute} \rangle, \\ & \langle \textit{step\_function} \rangle \rangle\end{aligned}$$

The type identifies an attribute modification by means of a step function. The attribute modified (rAttribute) is a tuple of type / value, and can represent anything modelled within the resource type, be it generic or specific. For example, this type could be instantiated to increase the total amount of bandwidth consumed by a subject, within a given session.

Definition 25 (System Change Model [24]) *"A system change model $\mathcal{CM}_{sys}$ is a set of change types applicable to the system properties (Γ_{sys}) of a family of systems that share some degree of commonality (*e.g.*, common subset of architectural types)."*

Example 8 *In the following, several examples of system change types are described, conveying the system's response to a subject requesting access.*

(i) ***Authentication decision type*** *captures the consequence (within an identity service) of an authentication request being responded to.*

$$\begin{aligned}\textbf{\textit{auth\_decision\_type}} = \langle & \textit{identity\_service}, \\ & \langle \textit{authDecision}(\textit{auth\_request}) \rangle, \\ & \langle \textit{event} \rangle \rangle\end{aligned}$$

(ii) ***Attribute release type*** *is the consequence of an attribute release request, within an identity service.*

$$\textbf{\textit{attr\_release\_type}} = \langle identity\_service, \langle attrRelease(attr\_release\_request)\rangle, \langle event\rangle\rangle$$

$$\textbf{\textit{attrRelease(attr\_release\_request)}} = \langle issuer, identity, \langle iCondition_1, ..., iCondition_n\rangle \langle iAttribute_1, ..., iAttribute_n\rangle\rangle$$

It details the releasable identity attributes (iAttribute) as a tuple stating the type of identity attribute and its value. Identity attributes are released along with the issuer of the attributes (i.e., an ID of the identity provider or individual whom assigned these attributes), the identity of the subject (i.e., a persistent ID), and a set of conditions. Conditions are a type value tuple, detailing the use of the released attributes. For example, a condition may assert the released attributes may only be used once, or can only be used in a given time frame.

(iii) ***Credential validation type*** *is the consequence of a credential validation request, within an authorisation service.*

$$\textbf{\textit{cred\_validation\_type}} = \langle auth\_service, \langle valCredentials(cred\_validation\_request)\rangle, \langle event\rangle\rangle$$

$$\textbf{\textit{valCredentials(cred\_validation\_request)}} = \langle viAttribute_1, ..., viAttribute_n\rangle$$

It returns valid attributes (viAttribute) for a subject if the provided iAttributes conform to the authorisation service's credential validation policy. These are effectively the same as identity attributes, however, they are referred to as valid because an authorisation service has checked that the identity service is trusted to issue them.

(iv) ***Access decision type*** *is the consequence of an access request, providing a decision based on the attributes within an access request, and an authorisation service's access control policy.*

$$\textbf{\textit{access\_decision\_type}} = \langle auth\_service, \langle accessDecision(access\_request)\rangle, \langle event\rangle\rangle$$

$$\textbf{\textit{accessDecision(access\_request)}} = decision$$

Changes

A change is an instantiation of a change type. Once enacted, the perception of state (either system or environment) has changed.

Definition 26 (Change [24]) *"Given a set of change types CT defined for a set of architecture types $\mathcal{T}$, and an architecture model $\mathcal{A} = (\mathcal{T}, \mathcal{G}, \Gamma)$, a* change *is a tuple $(ct, srcinst, V_A, V_B, ti, d)$ that corresponds to an instantiation of a change type, where:*

- *$ct = (src, A, B) \in CT$ determines the change type to be instanced as a change;*
- *$srcinst \in \mathcal{N}$, such that $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\Lambda(srcinst) = src$, is the instance of the source of change (*i.e.*, where it actually occurs);*
- *$V_A = \langle v_{A1}, \ldots, v_{Ak} \rangle$ is a vector of attribute values instantiating the attributes in A;*
- *$V_B = \langle v_{B1}, \ldots, v_{Bk} \rangle$ is a vector of behavior instances of the elements in B (*i.e.*, a behavior instance v_{Bi}, $i \in \{1, \ldots, k\}$ is a function of type $b_i \in B$, describing the evolution over time of the attribute $v_{Ai} \in V_A$);*
- *$ti \in \mathbb{R}_0^+$ determines the* time instant *in which the change instance is triggered;*
- *$d \in \mathbb{R}_0^+$ is the* duration *associated with the change."*

A set of exemplified changes are described in Appendix B.1. These changes demonstrate the instantiation of change types for both environment and system change, detailing the progression of a subject authenticating, requesting, and gaining access to a resource.

5.3.5 Scenarios and Changeload

A scenario encompasses a set of changes over time, in light of a set of system goals, and a given state. It is used to formally describe malicious behaviour over time, such as a progression of violations.

Definition 27 (Scenario [24]) *A scenario is a tuple $(Sys_{state}, Env_{state}, G, C)$, where:*

- Sys_{state} *is a state condition that represents the state of the system* (e.g., *workload, which is the amount and type of work assigned to the system);*
- Env_{state} *is a state condition that represents the state of the environment of the system* (e.g., *operational conditions of software and hardware resources needed for the system to perform its service, and operations that is beyond the scope of control);*
- G *is a set of system goals;*
- C *is a set of changes applied to the state determined by conditions within the system and environment.*

Key to a scenario is the definition of goals that should be fulfilled as the system undergoes change. In relation to detecting and mitigating malicious behaviour, a goal may refer to an error margin in detecting attacks, maximum response time to resolving attacks, and impact of attacks before required policy changes. In addition, is the distinction between a base scenario and a change scenario:

Definition 28 (Base Scenario [24]) *A base scenario is a tuple* $(Sys^t_{state}, Env^t_{state}, G_f, \emptyset)$*, where:*

- Sys^t_{state} *is a state condition that represents a typical state of the system;*
- Env^t_{state} *is a state condition that represents the typical environmental state of the system;*
- G_f *is a set of fixed system goals.*

Base scenarios enable the definition of a state that conforms to a system's conventional operational profile. It is assumed that a base scenario defines a state where no known malicious behaviour is present. Such an assumption requires that only malicious behaviour intended to be stimulated against the base scenario can be evaluated, as it is not possible to rule out the existence of unknown malicious behaviour.

There can be numerous valid base scenarios to the *LGZLogistics* case study. For example, a base scenario could describe the typical workload during a normal business day within *LGZLogistics*. This includes a typical definition of criteria and assignment of access. Alternatively, it could represent the initial deployment of its authorisation infrastructure (i.e., no workload).

Example 9 *A base scenario for the LGZLogistics case study portrays the authorisation infrastructure and its expected system properties, and environment properties, for a typical work day. For simplicity, only the system and environment properties relating to subject access are described. DailyAccess captures a base scenario of a typical system state relating to access decisions, and a typical environment state relating to access requests.*

$$DailyAccess_{BaseScenario} = (SysAccess^t_{state}, EnvReqs^t_{state}, G_f, \emptyset)$$

Each element of the base scenario tuple is expressed below. The system state combines properties that indicate the runtime parameters of services (e.g., authorisation policies), as well as system workload properties (e.g., rate of permitted access for a given subject). Both the system and environment states are defined in conformance to LGZLogistic's access control model (Section 5.2.2), and its definition of typical behaviour (Section 5.2.3).

$$
\begin{aligned}
&SysAccess^t_{state} = \\
&\quad \langle\langle as.policy,\ lgzIS.emp0003.attr,\ conIS.con0003.attr, \\
&\qquad as.emp0003.empDB.read,\ as.con0003.empDB.read\rangle, \\
&\quad \langle constant\_function,\ constant\_function,\ constant\_function, \\
&\qquad constant\_function,\ constant\_function\rangle, \\
&\quad \langle AP_1,\ \{Staff, SysAnalyst\},\ \{Contractor\},\ 0.6,\ 1.25\rangle, \\
&\quad \langle \theta_{as.policy}(t) = AP_1,\ \theta_{as}(emp0003.permisRole) = \{Staff, SysAnalyst\}, \\
&\qquad \theta_{as}(con0003.permisRole) = \{Contractor\},\ \theta_{as}(t) = 0.6/min, \\
&\qquad \theta_{as}(t) = 1.25/min\rangle \\
&\quad \rangle
\end{aligned}
$$

$SysAccess^t_{state}$ defines the state of access control, including policies and attribute assignments. For example, subject emp0003 from identity service lgzIS, *is assigned attributes $\langle permisRole, \{Staff, SysAnalyst\}\rangle$. AP_1 denotes a PERMIS authorisation policy that implements the valid attribute assignment rules in Figure 5.3, and attribute permission assignments in Figure 5.2. Note, the system state defined is not exhaustive, rather it focuses only on: system properties that define the current state of access; provides an example of attribute assignment to a subject from each identity provider; and an example rate of permitted access to the* empDB *resource.*

$$
\begin{aligned}
&EnvReqs^t_{state} = \\
&\quad \langle\langle as.SysAdmin.empDB.Read, as.SysAdmin.empDB.Modify,
\end{aligned}
$$

$as.SysAdmin.empDB.Create$, $as.SysAdmin.empDB.Delete$,
$as.SysAnalyst.empDB.Read$, $as.SysAnalyst.empDB.Modify$,
$as.ContractorSupervisor.empDB.Read$,
$as.ContractorSupervisor.empDB.Modify$,
$as.Contractor.empDB.Read$, $as.Staff.logisticsTool.Access\rangle$,
$\langle$*constant_function*, *constant_function*, *constant_function*
constant_function, *constant_function*, *constant_function*
constant_function, *constant_function*, *constant_function*
contstant_function$\rangle$,
$\langle 0.8,\ 0.4,\ 0.2,\ 0.2,\ 4.8,\ 1.6,\ 6.7,\ 6.7,\ 3.8,\ 20.0\rangle$,
$\langle \theta_{as}(t) = 0.8/min,\ \theta_{as}(t) = 0.4/min,\ \theta_{as}(t) = 0.2/min,\ \theta_{as}(t) = 0.2/min,$
$\theta_{as}(t) = 4.8/min,\ \theta_{as}(t) = 1.6/min,\ \theta_{as}(t) = 6.7/min,$
$\theta_{as}(t) = 6.7/min, \theta_{as}(t) = 3.8/min,\ \theta_{as}(t) = 20.0/min\rangle$
$\rangle$

$EnvReqs^t_{state}$ defines the state of the environment with regards to subjects requesting access. The environment properties identified in the state condition refer to collective behaviour per attribute per permission. For example, for subjects requesting access to 'read' empDB, *with attribute $\langle permisRole, SysAdmin\rangle$, a rate of 0.8 requests per minute is observed. As there are two subjects with this attribute (Figure 5.2), it is assumed that each subject has an average rate of 0.4 requests per minute (i.e., one request every 150 seconds).*

The fixed goals (G_f) define the conditions that must be maintained within the authorisation infrastructure, regardless of change. Ultimately, a goal requires a system to be brought out of a non-conventional operational state (once identified). However, goals also focus on a wider scope of conditions that attempt to ensure that only necessary adaptations are taken, once in a non-conventional state. The following describes a set of goals relevant to the LGZLogistics case study:

- *Probability of 99% that all instances of known violation types are detected;*
- *Probability of 90% that violations are mitigated through subject adaptation;*
- *Probability of 99% that all adaptations performed exhibit a lower cost than current and unmitigated violations, to the organisation.*

Probabilities cited are pseudo values that indicate LGZLogistics requirements for mitigation. However, an accurate probability can only be achieved through rigorous benchmarking of the scenario in an off-line environment [24]. In any case, *probabilities defined are specific to the deployment environment, and configuration of the authorisation infrastructure.*

Cámara et al. state that a change scenario represents a set of changes applied to a base scenario [24]. As such, a change scenario instantiates a set of changes within the authorisation infrastructure when it is in a particular state. Through the application of change scenarios, it is expected to bring the authorisation infrastructure into an non-conventional state, where the authorisation infrastructure's fixed goals can be evaluated.

Definition 29 (Change Scenario [24]) *A change scenario is a tuple (Sys^t_{state}, Env^t_{state}, G_f, C). It is defined by a typical condition of the system followed by a non-empty set of changes.*

- Sys^t_{state} *is a state condition that represents a typical state of the system;*
- Env^t_{state} *is a state condition that represents the typical environmental state of the system;*
- G_f *is a set of fixed system goals;*
- $C \neq \emptyset$ *is a sequence of changes applied to the state determined by the system and environment.*

Example 10 *The following sequence of changes describes subject* emp0003 *accessing the* empDB *resource.*

$$
\begin{aligned}
\mathbf{c_1} = (&access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB,\\
&read, \langle NULL\rangle, pid = bxu915810faa4910)\rangle, \langle event\rangle, 5, 0)\\
\mathbf{c_2} = (&access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB,\\
&read, \langle NULL\rangle, pid = bxu915810faa4910)\rangle, \langle event\rangle, 10, 0)\\
\mathbf{c_3} = (&access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB,\\
&read, \langle NULL\rangle, pid = bxu915810faa4910)\rangle, \langle event\rangle, 15, 0)
\end{aligned}
$$

$\mathbf{c_1}$ *describes a single access request for emp0003, identified by privacy protected id (PID) bxu915810faa4910, using attribute $\langle permisRole, SysAnalyst\rangle$, to access* empDB*. Thereafter, at 5 second intervals, new changes are instantiated, whereby the request for access to* empDB *is repeated. As a result, the subject affects a number of environment properties associated to the system, namely, the subject's rate of access to* empDB*.*

The sequence of changes describes a rapid rate of access to the empDB *resource (labelled as $C_{rapidAccess}$). With the sequence of changes defined, it is applied to the base scenario with the following notation:*

$$ExpectedToRapidUsage_{ChangeScenario} = (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{rapidAccess})$$
$$C_{rapidAccess} = \{c_1, c_2, c_3\}$$

Changeload

Definition 30 (Changeload [24]) *"A changeload is a set of change scenarios that demonstrates changes either: valid within a conventional operational profile, invalid, thus stimulating adaptation, or the result of adaptation."*

Cámara et al. formulated their changeload model primarily to classify system and environment change that stimulates adaptation. As such, a malicious changeload, in the context of authorisation infrastructures, is one that drives stimulation of adaptation in response to the abuse of access control (i.e., places a system into a non-conventional operational state). It is considered that both environment and system stimulation are capable in generating non-conventional operational states (and are often a by-product of each other), whereby environment change leads to system change.

5.4 Case Study: Changeload

Before stimulating change in a runtime execution of the *LGZLogistics* case study, it is necessary to first identify the violations relevant to the case study, along with change types, and the scope of change (i.e., what changes are likely to stimulate adaptation).

5.4.1 Violations

A set of violations are defined as the upper bounds of abnormal behaviour, based on the normal behaviour described in the *LGZLogistics* case study. It is assumed that historical data of subject behaviour (if present), coupled with an expert approach, is used to define relevant violations. With reference to the Self-Adaptive Authorisation Framework (SAAF), each violation is defined as a trigger rule (with an associated cost).

The following violations detail patterns of access against *LGZLogistic's* resources, regarding short term and long term rates of access invoking certain permissions. For each violation a maximum rate of access is defined, whereby a short term rate refers to subject access within a minute interval, and a long term rate refers to subject access within a 10 minute interval (to simulate a scaled measure of prolonged change).

For example, violation `empDBShortRead` and violation `empDBLongRead` classifies malicious behaviour as any subject successfully requesting access to invoke the 'read' action on empDB, at a greater rate than a max allowable. A constraint is applied to the violation, whereby this violation only applies to subjects whom do not have the attribute $\langle permisRole, SysAdmin\rangle$.

$$\begin{aligned}&\texttt{empDBShortRead} =\\&\quad(\texttt{subAccessReqRate}_{\texttt{empDB.read}} > \texttt{MaxAccessReqShortRate}_{\texttt{empDB.read}}) \wedge\\&\quad(\texttt{subAttribute} <> \langle permisRole, SysAdmin\rangle)\end{aligned}$$

$$\begin{aligned}&\texttt{empDBLongRead} =\\&\quad(\texttt{subAccessReqRate}_{\texttt{empDB.read}} > \texttt{MaxAccessReqLongRate}_{\texttt{empDB.read}}) \wedge\\&\quad(\texttt{subAttribute} <> \langle permisRole, SysAdmin\rangle)\end{aligned}$$

For violations `empDBShortModify` and `empDBLongModify`, malicious behaviour is classified in terms of any subject successfully requesting access to invoke the 'modify' action on empDB, at a greater rate than a max allowable. As with the aforementioned violations, a constraint is applied meaning the violation is only applicable to subjects who do not have the attribute $\langle permisRole, SysAdmin\rangle$.

$$\begin{aligned}&\texttt{empDBShortModify} =\\&\quad(\texttt{subAccessReqRate}_{\texttt{empDB.modify}} > \texttt{MaxAccessReqShortRate}_{\texttt{empDB.modify}}) \wedge\\&\quad(\texttt{subAttribute} <> \langle permisRole, SysAdmin\rangle)\end{aligned}$$

$$\begin{aligned}&\texttt{empDBLongModify} =\\&\quad(\texttt{subAccessReqRate}_{\texttt{empDB.modify}} > \texttt{MaxAccessReqLongRate}_{\texttt{empDB.modify}}) \wedge\\&\quad(\texttt{subAttribute} <> \langle permisRole, SysAdmin\rangle)\end{aligned}$$

Violation `empDBShortDelete` classifies malicious behaviour in a subject rapidly gaining access to delete entries within the `emphDB` resource.

$$\begin{aligned}&\texttt{empDBShortDelete} =\\&\quad(\texttt{subAccessReqRate}_{\texttt{empDB.delete}} > \texttt{MaxAccessReqShortRate}_{\texttt{empDB.delete}})\end{aligned}$$

Violation lgTShortAccess classifies malicious behaviour of subjects rapidly accessing the lgT (logistic tool) resource.

$$\texttt{lgTShortAccess} = (\texttt{subAccessReqRate}_{\mathsf{lgT}} > \texttt{MaxAccessReqFastRate}_{\mathsf{lgT}})$$

Violation empDBTransaction is slightly different, whereby it classifies a transaction of non-conventional change. This type of violation denotes a pattern whereby a rate of transactional requests are compared against a maximum rate. The violation requires an environment property that measures the rate of access requests, by a subject, in performing a read action succeeded by a modify action against empDB. Basically, it aims to identify subjects who rapidly read and write to the empDB resource.

$$\begin{aligned}&\texttt{empDBTransaction} =\\&\quad(\texttt{subAccessReqRate}_{\texttt{empDB.readModifyTransaction}} >\\&\qquad\texttt{MaxAccessReqLongRate}_{\texttt{empDB.readModifyTransaction}})\end{aligned}$$

A final violation, albeit by contrast does not capture subject activity directly, is dueRedundancy. This violation is a consequence of a change made within the empDB resource, indicating that a subject has been marked for job redundancy. A subject facing the prospect of redundancy is seen as a potential risk, and as such, a violation is used to increase the impact a subject has on an organisation. This is viewed as a motivator for adaptation, as when combined with previously identified violations, the subject's activity may now warrant adaptation.

$$\texttt{dueRedundancy} = (\texttt{subDueRedundancy} == \textit{true})$$

5.4.2 Identifying Change Types and Change

To stimulate violations within the context of the *LGZLogistics* case study, it is necessary to identify properties of interest and the change types that will impact such properties. For this specific case study, only environment properties are considered. These are properties that concern subject activity that cannot be directly controlled (e.g., a subject's rate of access requests).

Environment Properties

For each violation, and for each subject, there exists a set of environment properties that measure the extent of change in the environment. Many environment properties represent composite properties of subject-related changes over time. In reference to SAAF, these properties are dynamically created as mutable attributes within SAAF's behaviour model (Chapter 3, Section 3.4.1), and updated through the observation of environment change via probes.

For example, `empDBShortRead` asserts that if a subject's access rate in requesting a read on `empDB` (who is not a *SysAdmin*) goes beyond a maximum number of requests within a minute interval, a violation has occurred. To measure against this violation, an environment property of $\texttt{as.subject.AcessReqRate}_{\texttt{empDB.read}}$ is required (e.g., $\texttt{as.emp0003.AccessReqRate}_{\texttt{empDB.read}}$).

Change Types and Changes

Once environment properties are identified it is necessary to select relevant change types (and changes) that result in a non-conventional operational state. For example, a non-conventional operational profile that contains the violation `empDBShortRead` is realised through a succession of changes, whereby a single subject successfully requests access to 'read' `empDB`. The violation occurs when a subject, e.g., emp0003, has performed a number of **Access request** change type, and is permitted by an **Access decision** change type.

The **Access request** change type is the result of a number of sequential changes, such as the subject first authenticating with their identity provider, requesting a release of attributes as credentials, and validation of attributes. In this instance, these changes need to be realised before a subject performs an **Access request** change type.

All but one violation described for the *LGZLogistics* case study is triggered by an **Access request** change type. The violation `dueRedundancy` is triggered by a **Resource action step** change, whereby an environment property for a given subject indicates a subject is due for job redundancy (e.g., `empDB.emp0003.isSetForRedundancy`).

5.4.3 Malicious Changeload

Using the *LGZLogistics* malicious behaviour scenario, the following set of change scenarios are defined. Together they represent the malicious changeload for the case study. There are seven change scenarios defined within the malicious changeload, representative of the case study's malicious behaviour scenario. Each change scenario is applicable to the base scenario defined in Section 5.3.5 (Example 9).

The first change scenario ($setSubjectRedundancies_{ChangeScenario}$) considers a set of resource changes relevant to empDB, where changes identify four system analysts are to be made redundant (`dueRedundancy`).

$$
\begin{aligned}
&setSubjectRedundancies_{ChangeScenario} = \\
&\qquad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{setRedundancies}) \\
&C_{setRedundancies} = \{ \\
&\quad \mathbf{c_1} = (resource\_action\_step\_type, empDB, \langle emp0003.Redundancy\rangle, \\
&\qquad \langle emp0003.Redundancy = true\rangle, 0, 0) \\
&\quad \mathbf{c_2} = (resource\_action\_step\_type, empDB, \langle emp0004.Redundancy\rangle, \\
&\qquad \langle emp0004.Redundancy = true\rangle, 0, 0) \\
&\quad \mathbf{c_3} = (resource\_action\_step\_type, empDB, \langle emp0005.Redundancy\rangle, \\
&\qquad \langle emp0005.Redundancy = true\rangle, 0, 0) \\
&\quad \mathbf{c_4} = (resource\_action\_step\_type, empDB, \langle emp0006.Redundancy\rangle, \\
&\qquad \langle emp0006.Redundancy = true\rangle, 0, 0)\}
\end{aligned}
$$

The second scenario ($emp0003ReadModify_{ChangeScenario}$) describes a malicious change scenario resulting in violations `empDBLongReadModify`, `empDBLongRead`, and `empDBLongModify`, whereby subject emp0003 persistently reads and modifies records in the empDB resource, every four seconds. The function δ is defined in order to calculate the time at which a change is executed within the change scenario. For the following change scenario, δ is defined as $\delta(n) = \frac{1}{2}(4n - (-1)^n + 1)$, where n refers to the nth change in the change scenario.

$$
\begin{aligned}
&emp0003ReadModify_{ChangeScenario} = \\
&\qquad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{maliciousTransactions}) \\
&C_{maliciousTransactions} = \{ \\
&\quad \mathbf{c_1} = (access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB, \\
&\qquad read, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, 3, 0) \\
&\quad \mathbf{c_2} = (access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB, \\
&\qquad modify, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, 4, 0) \\
&\quad \ldots \\
&\quad \mathbf{c_n} = (access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB, \\
&\qquad read, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, \delta(n), 0) \\
&\quad \mathbf{c_{n+1}} = (access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, \\
&\qquad\quad empDB, modify, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, \delta(n), 0)\}
\end{aligned}
$$

The third scenario ($con0002FastRead_{ChangeScenario}$) describes a malicious change scenario by subject con0002, resulting in violations `empDBShortRead` and `empDBLongRead`. In the scenario, a contractor supervisor persistently accesses the empDB resource at a rate of 2 seconds, in order to obtain employee data. The scenario begins 150 seconds relative to the start of malicious changeload. The function δ denotes the progression in time (seconds) between changes, defined as $\delta(n) = 2n$.

$$
\begin{aligned}
&con0002FastRead_{ChangeScenario} = \\
&\qquad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{con0002FastRead}) \\
&C_{con0003FastRead} = \{ \\
&\quad \mathbf{c_1} = (access\_request\_type, as, \langle request(\langle\langle permisRole, ContractorSupervisor\rangle\rangle, \\
&\qquad empDB, read, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, 150, 0) \\
&\quad \ldots \\
&\quad \mathbf{c_n} = (access\_request\_type, as, \langle request(\langle\langle permisRole, ContractorSupervisor\rangle\rangle, \\
&\qquad empDB, modify, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, \delta(n), 0)\}
\end{aligned}
$$

For contractors con0003, con0004, and con0005, similar change scenarios exist based on $con0002FastRead_{ChangeScenario}$. However, the scenarios are introduced in stages of 30 second intervals (i.e., con0003 begins at 3 minutes from the start of the malicious changeload, con0004 begins at 3.5minutes, etc.). The rate of changes is defined as $\delta(n) = 2.5n$, where subjects utilise their $\langle permisRole, Contractor\rangle$ attribute, in accessing empDB.

Lastly, $emp0003FastAccess_{ChangeScenario}$ describes a further malicious change scenario by emp0003, resulting in violation `lgTShortAccess`. In this scenario, emp0003 persistently requests access every 370ms and gains access to lgT resource (logisticsTool resource), to disrupt the performance of the resource. The scenario begins 900 seconds relative to the start of malicious changeload. δ denotes a function whereby progression in time (seconds) is defined as $\delta(n) = 0.37n$.

$$
\begin{aligned}
&emp0003FastAccess_{ChangeScenario} = \\
&\qquad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{emp0003FastAccess}) \\
&C_{emp0003FastAccess} = \{ \\
&\quad \mathbf{c_1} = (access\_request\_type, as, \langle request(\langle\langle permisRole, Staff\rangle\rangle, lgT, \\
&\qquad read, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, 900, 0) \\
&\quad \ldots \\
&\quad \mathbf{c_n} = (access\_request\_type, as, \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, lgT, \\
&\qquad read, \langle NULL\rangle, pid = emp0003)\rangle, \langle event\rangle, \delta(n), 0)\}
\end{aligned}
$$

This malicious changeload (consisting of the seven change scenarios) concisely describes the *LGZLogistics* malicious behaviour scenario. It is the intention that the changeload can be repeated under various operational conditions, and also used to compare future approaches to self-adaptive authorisation. As such, it can be exploited to execute simulation of changes within an authorisation infrastructure, in order to evaluate the impact of violations, and trigger self-adaptive responses. However, one limitation is that no parser currently exists to execute a defined changeload. Therefore, a changeload can only be viewed as a model of change, which must be manually transformed into an executable script (e.g., Jmeter simulation scripts).

5.5 Experiments

The *LGZLogistics* case study is simulated within a live self-adaptive authorisation infrastructure. This self-adaptive authorisation infrastructure is instantiated across across four individual machines. Two machines running **DebianLinuxv6.0.5** (512MB of memory) are deployed hosting an LDAP directory and an installation of SimpleSAMLphp (v1.9.2) [131]. These are configured to operate as the lgzIS and conIS identity services, respectively. A single machine running **UbuntuLinuxv10.10** (2048MB of memory) is deployed hosting an installation of the PERMIS standalone service (v.0.3.2), instantiating authorisation service as, and a prototype of the SAAF controller. Lastly, a single 'client' machine running **Windows7** (2048MB) is deployed to simulate activity between subjects accessing a resource, and communicating with services of the authorisation infrastructure.

The rest of this section details a brief overview of the deployment of the prototype of the SAAF controller, a description of how the malicious changeload is simulated within the environment, the execution of experiments, and lastly, a summary of results.

5.5.1 Deploying the SAAF Controller

In contrast to the centralised authorisation infrastructure conveyed in Chapter 4, in the *LGZLogistics* case study, SAAF is deployed within a federated authorisation infrastructure. The configuration of these two infrastructures differs greatly, where additional services in the federated infrastructure are deployed to facilitate access (and adaptation).

Figure 5.5 portrays *LGZLogistic's* federated authorisation infrastructure, based on the architectural model described in Figure 5.1. Here, the infrastructure is distributed across multiple management domains (identity provider and service provider domains). *LGZLogistics* operates a service provider domain (to handle authorisation and provision access to resources), and their own identity provider domain (to handle identity management of their own employees). In addition, the *contractor* organisation is said to operate their own identity provider domain (to handle identity management of their own employees).

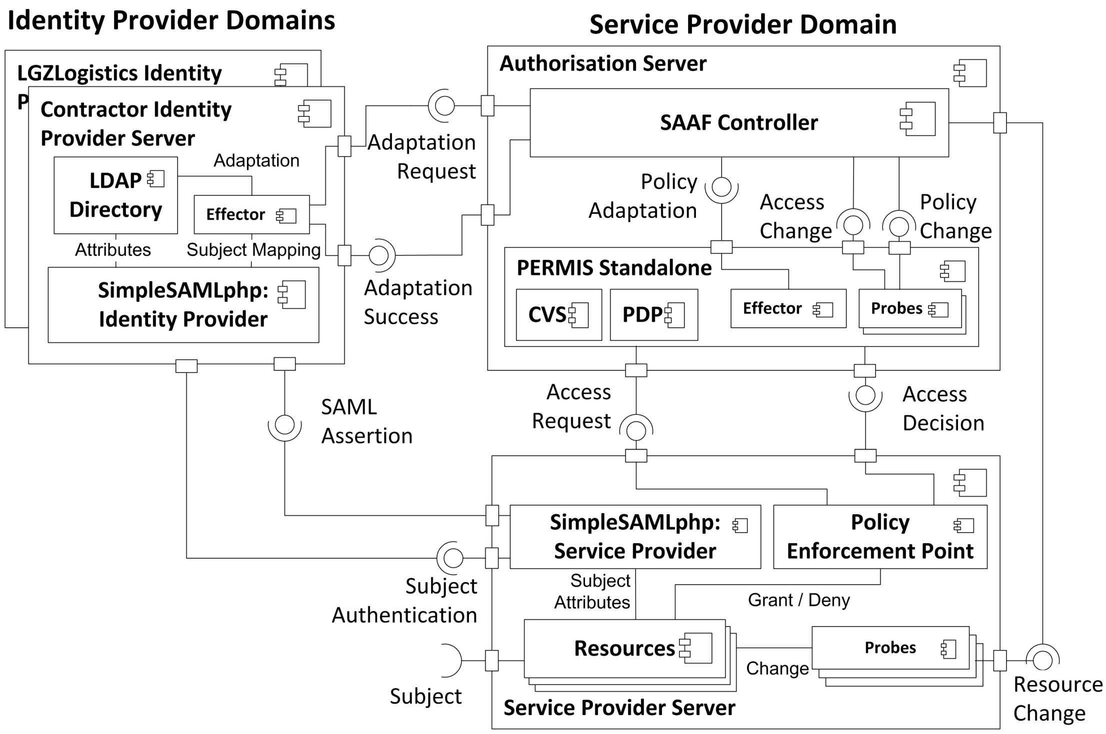


Figure 5.5: Self-adaptive federated authorisation infrastructure

SimpleSAMLphp [131] is used as the enabling technology to facilitate communication between these management domains. It provides a layer of control over 'what' information can be released or requested (in regards to subjects), and how subjects can be authenticated. Deployments of SimpleSAMLphp are capable of exchanging information via signed or unsigned SAML assertions [99], such as messages containing a set of subject attributes and the subject's unique identifier.

A SAAF prototype controller is deployed within *LGZLogistics* service provider domain, whereby it is expected to manage authorisation assets across both management domains. However, self-adaptation over multiple management domains is a challenging and non-trivial problem. Identity providers often do not release uniquely identifiable personal information to service providers, and use transient (TID) or persistent (PID) IDs to allow service providers to identify subjects. In addition, identity providers may not be as forthcoming to accepting adaptations from an organisation outside of their management domain, meaning the SAAF controller can only 'request' adaptation.

A solution to enabling adaptation across multiple management domains is the deployment of an effector managed by the identity provider domain [7] (see Appendix B.2). Here an effector can map a service provider's view of a subject (i.e., from a subject TID / PID to subject LDAP entry), and govern which adaptations to perform. Instantiations of this effector are deployed on each of the identity services (*subject adaptation*), as well as an effector capable of deploying and activating policies within the PERMIS authorisation service (*policy adaptation*).

A resource probe is deployed on the empDB resource to observe changes to the state of an employee's job redundancy property (*resource change*). In addition, a probe is deployed on the PERMIS authorisation service to detect *access change* and *policy change*. A probe is not deployed on the contractor's identity service (conIS) simulating a limitation in federated authorisation infrastructures, where third party organisations may prevent immediate access to their subject's attributes (*subject change*). This limits the SAAF prototype's view of the state of access, whereby the SAAF prototype must infer its perception of subjects from the observation of access requests (via the authorisation service as).

The SAAF controller is configured (Appendix B.3.1) to detect and mitigate the set of violations described in Section 5.4.1. Here, a solution policy exists containing a set of solutions applicable to mitigating instances of these violations. Each solution contains a weighting of cost to the deploying organisation (e.g., the cost in removing subject access, or removing the trust in an identity provider). A minimum subject impact weighting, on a scale of 0 to 1, is also defined, which is used to constrain a subset of solutions relevant to resolving differing scales of malicious subject behaviour. These weightings are used as part of solution analysis and solution selection, as described in Chapter 4. The following solutions are configured for this deployment:

- `S0: noAdaptation` default solution for when all other solutions cause greater impact over an observed behaviour
- `S1: removeSubjectAttribute` removal of an individual abused attribute from a subject (i.e., the cause of a violation)
- `S2: removeAllSubjectAttributes` removal of all attributes from a subject, typical for

when subjects are persistently abusing access

- S3: `removeAttributeAssignment` removal of trust in an identity provider in issuing valid attributes (policy change)
- S4: `removeAllAttributeAssignments` removal of all trust in an identity provider in issuing valid attributes (policy change)
- S5: `deactivatePolicy` removal of all access to all of *LGZLogistic's* resources

A limitation in this deployment is the inability to use the integrated rbacDSML tool, preventing solution verification from taking place. This is due to the deployment operating within a federated environment that conforms to ABAC, which rbacDSML is unable to accommodate for. It was decided that evaluating SAAF within a federated environment provided greater contributions as opposed to enhancing rbacDSML to operate within a federated ABAC environment. For example, focusing on self-adaptation in federated deployments enables the evaluation of adaptation when faced with non-cooperating management domains. As a consequence, a SAAF constraints model is not defined, meaning all solutions are assumed to result in acceptable changes to the implemented access control model.

5.5.2 Executing LGZLogistics Changeload

The execution of the *LGZLogistics* malicious changeload (Section 5.4.3) is achieved through enacting environment change via a number of protocols:

1. LDAP binds [76], for the authentication of subjects within identity providers.
2. SAML assertions [99], for the requesting and deliverance of released attributes as signed credentials (to and from identity provider services).
3. SOAP messages [21], for credential validation requests, credential validation responses, access requests, and access decisions (to and from protected resources and authorisation services).

An installation of the Jmeter testing (v.2.11) tool [3] (deployed on the Windows 'client' machine) automates each of the change types applicable to an authorisation infrastructure, using the aforementioned protocols. Here, subjects are simulated in authenticating, requesting, and obtaining access to protected resources.

Using the experimentation profile proposed by Cámara et al. [24], the malicious changeload is executed across multiple runs as part of four experiments. The four experiments are designed to evaluate the SAAF prototype. Exp1 and Exp2 evaluate the prototype in mitigating the malicious changeload under normal and high loads, respectively. Exp3 and Exp4 also evaluate under normal and high loads, respectively, but simulate limited control due to the deactivation of the contractor's identity provider effector.

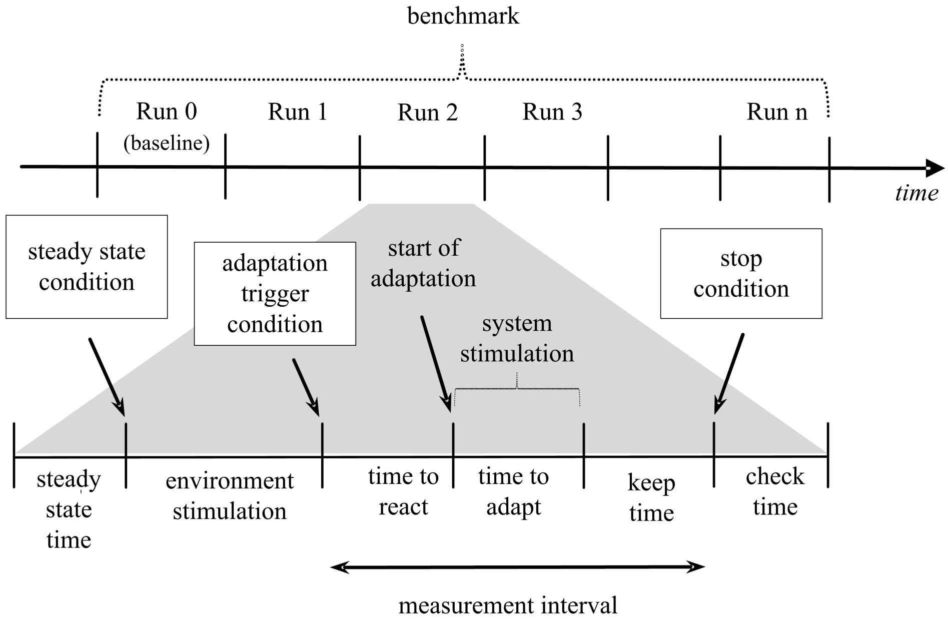


Figure 5.6: Executing changeload experimentation profile [24]

Each experiment is executed six times (referred to as 'runs'), where each run follows the set of stages stated in Figure 5.6. A run adheres to a steady state time (realisation of the base scenario, described in Example 9), environment stimulation (the execution of the malicious changeload), time to react (detection of malicious behaviour and decision to act), time to adapt (time it takes to perform adaptations), a keep time (to observe system recovery post adaptation), and check time (post analysis of each run). At the end of each run the system and environment states are reset before performing the next run.

Steady state time is maintained for a period of 30 minutes in order to ensure the controller and authorisation infrastructure is evaluated in a warmed up state. During this time, the baseline scenario is simulated within the authorisation infrastructure. After 30 minutes, the malicious changeload scenario is initiated (environment stimulation). From this period on there is a set of staggered violations in which several periods of 'time to react' and 'time to adapt' overlap environment stimulation. This is necessary in order to evaluate the prototype's ability to detect and mitigate multiple attacks that have been conducted collaboratively. Post adaptation is referred to as keep time, where the baseline scenario resumes and no further adaptation takes place. Lastly, keep time remains the same for each run within an experiment.

5.5.3 Experiments Execution

The first two experiments, Exp1 and Exp2, demonstrate adaptation under increasing loads on the controller (in terms of processing environment change). The last two experiments, Exp3 and Exp4, duplicate the same normal and high loads on the controller, yet simulates a scenario where the contractor identity provider effector has failed, or has purposely been deactivated to prevent adaptation.

Baseline execution

Baseline runs identify the impact of the malicious changeload whereby no adaptation takes place. During these runs, the prototype controller is active, yet limited to only detecting the number and types of violations that have occurred.

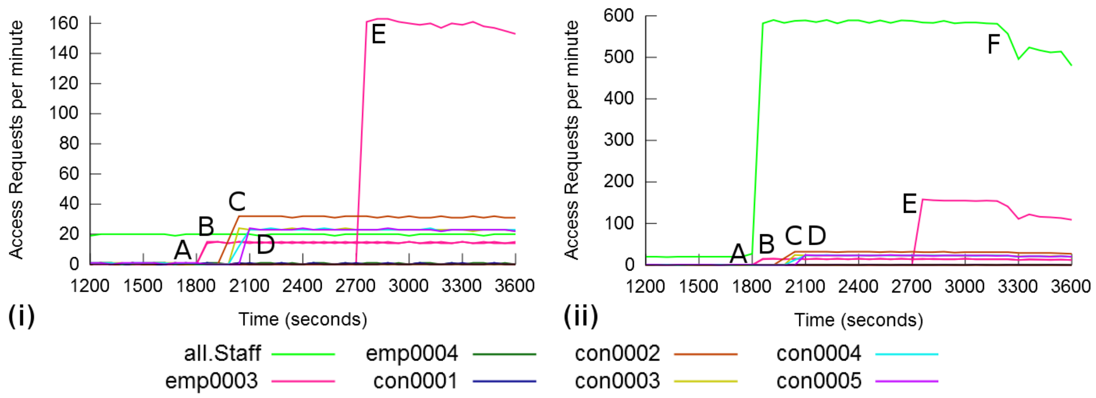


Figure 5.7: Baseline (i) normal load, (ii) high load

Figure 5.7 (i) and (ii) describe the rate of access of key subjects within the *LGZLogistics* authorisation infrastructure (taken at minute intervals). Note that 'all.Staff' indicates an aggregate rate for all subjects with attribute $\langle permisRole, staff \rangle$, whereas, all others represent the access requests of an individual. Figure 5.7 (i) depicts execution of malicious changeload under normal load, simulated as the continuation of the base scenario throughout environment stimulation. Figure 5.7 (ii) depicts execution of malicious changeload under high load, simulated as an increase to the base scenario's 'staff' rate of access, from 20req/min to 600req/min.

The normal load baseline (i) is representative of a baseline run for Exp1 and Exp3, whereas the high load baseline (ii) is representative of the baseline run for Exp2 and Exp4. This is because each pair of experiments undergo the same steady state and malicious changeload scenarios for their corresponding runs.

Comparing the baseline runs portrayed minimal difference in violations observed. Point A indicates the start of the malicious changeload (1800 seconds into the run), where the *setSubjectRedundancies* change scenario is executed, sending the controller several resource change events. It also indicates the start of *emp0003ReadModify* change scenario, at point B, where a system analyst begins to persistently read and modify records in empDB at a rate of 15req/min. At point C a contractor supervisor (*con0002FastRead*) begins to persistently read the empDB resource at a rate of 33req/min. This is followed by D, where three contractors also begin malicious behaviour, exhibiting a slightly lower request rate of 24req/min. Lastly, at point E, *emp0003FastAccess* change scenario is stimulated, representing a system analyst attempting to disrupt the performance of resource lgT, accessing at a rate of 160req/min.

The only exception between the two baselines is indicated at point F (high load baseline). As a result of the client machine being pushed to its limits (overloaded by *emp0003FastAccess*), a slowdown in load occurred after 3000 seconds into the run. Whilst this presented an anomaly to the baseline, adaptation runs were not impacted, as shown in Figures 5.8 and 5.9 (due to adaptation occurring before a slowdown could occur on the adaptation runs).

Regarding the detection of malicious behaviour, the controller detected 275 violations in normal load (i), and 260 violations in high load (ii). These were confined to six types of violations: `dueRedundancy`, `empDBTransaction`, `empDBShortRead`, `empLongRead`, `lgTShortAccess`, `empLongModify`. The high load baseline had fewer detections due to the slowdown of client requests at 3000 seconds into the run.

Exp: 1 & 2, Normal and High Loads

Experiments Exp1 and Exp2 undergo the same malicious changeload, albeit against a normal and high load, respectively. These are discussed in the context of the Figure 5.8 and Table 5.1 (adaptation under normal load). The table of results for adaptation under high load can be viewed in Appendix B.3.3 (Table B.5). In addition, a break down of solution selection is described in Appendix B.3.2.

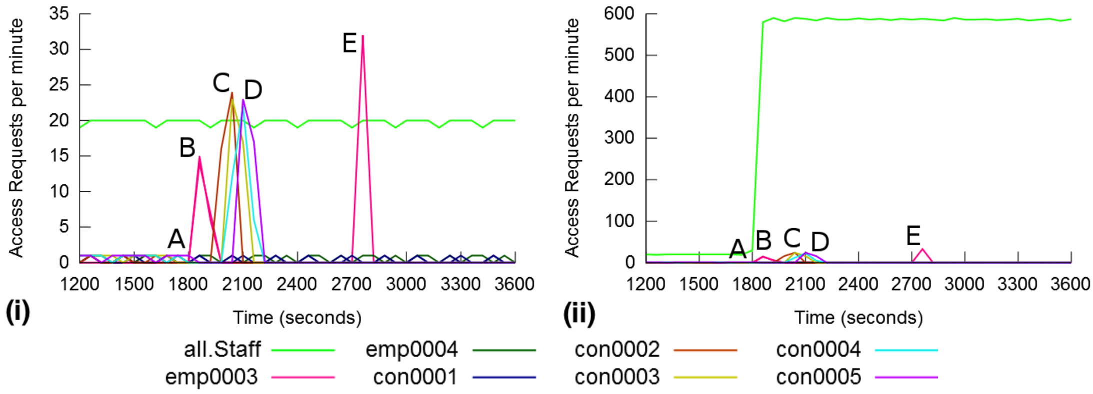


Figure 5.8: (i) Exp1 normal load, (ii) Exp2 high load

Step	Subject	Impact	Violation	Identified Solutions	Selected Solutions	R$_{Time}$ (Avg, Std)	A$_{Time}$ (Avg, Std)	Result
1	emp03	0.07	`dueRedundancy`	`S0`	`S0`	4.6, 0.5	N/A	N/A
2	emp04	0.07	`dueRedundancy`	`S0`	`S0`	2, 0	N/A	N/A
3	emp05	0.07	`dueRedundancy`	`S0`	`S0`	2, 0	N/A	N/A
4	emp06	0.07	`dueRedundancy`	`S0`	`S0`	1.8, 0.4	N/A	N/A
5	emp03	0.4	`empDBTransaction`	`S1`	`S1`	291.6, 59.6	182.2, 48.6	1
6	con02	0.07	`empShortRead`	`S0`	`S0`	1.4, 0.5	N/A	N/A
7	con02	0.27	`empShortRead`	`S1`	`S1`	186.6, 77.5	153.4, 22	1
8	con03	0.07	`empShortRead`	`S0`	`S0`	2.4, 1.1	N/A	N/A
9	con04	0.07	`empShortRead`	`S0`	`S0`	1.6, 0.5	N/A	N/A
10	con03	0.27	`empShortRead`	`S1`	`S1`	94.8, 33.8	165.2, 63.2	1
11	con05	0.07	`empShortRead`	`S0`	`S0`	4.6, 2.8	N/A	N/A
12	con04	0.27	`empShortRead`	`S1`	`S1`	139.8, 23.1	146.2, 38.9	1
13	con05	0.27	`empShortRead`	`S1`	`S1`	70.2, 9.7	120.4, 27.7	1
14	emp03	0.8	`lgTShortAccess`	`S2,S3,S4,S5`	`S2`	297.8, 35.8	189.4, 63.8	1

Table 5.1: Exp1: Adaptation with normal load (time in milliseconds)

Both experiments resulted in the consistent identification and selection of solutions to violations, where 14 attack steps were identified and responded to. Table 5.1 details these attack steps, where each *step* describes:

- A malicious *subject* who has carried out a violation;
- The calculated *impact* of the *subject* to the organisation;
- The *violation* observed;
- A set of *identified solutions* in which to mitigate the violation;
- The *selected solution* used to mitigate the violation;
- The response time (R_{Time}) in which to select the solution;

- The time to carry out an adaptation A_{Time};
- The result as to whether a solution was successful (1), failed (0), or not applicable (i.e., no adaptation performed).

Reviewing Table 5.1, steps 1 to 4 identify a resource change event (at point A), which triggered the violation `dueRedundancy`. For each subject, an impact level was calculated based on past behaviour observed. The controller calculated a low impact for these subjects (0.07), as none of these subjects have any previous violations. However, the result of this means that the controller is less tolerable to future violations.

Step five portrays the controller's first adaptation in response to subject *emp0003* triggering a second violation (`empDBTransaction` at point B). As a result, the subject's impact level was recalculated from 0.07 to 0.4. Solution S1 was identified (as being within scope of the subject's level of impact), and realised in the form of an adapted ABAC model (whereby the subject's $\langle \textit{permisRole}, \textit{SysAnalyst} \rangle$ attribute is removed). The adapted ABAC model was then assessed by the controller's planning stage, ensuring the identified solution does not cause a greater cost to the organisation over observed violations. As the solution only impacts the malicious subject, and no other solution is applicable to the impact of the subject, solution S1 is selected. Solution S1 is enacted as a SOAP message [7] sent to the subject's relevant identity provider (lgzIS) effector. The effector accepts the request and removes *emp0003's SysAnalyst* attribute. The consequence of this adaptation is that *emp0003* is no longer able to gain future access to empDB, as the employee lacks the necessary access rights.

In steps 6 to 13, the contractor supervisor (*con0002*) and three other contractors are detected at points C and D respectively, triggering violation `empDBShortRead`. Detection results in a similar process to that of the first attack by *emp0003* (step 5), where each subject's impact is recalculated and appropriate solutions are identified and enacted. Eventually, each contractor's relevant attributes in accessing the *empDB* resource are identified and removed via a SOAP message sent to the contractor's identity provider (conIS) effector.

In the final step, *emp0003* rapidly accesses the lgT resource, this time using their remaining attribute $\langle \textit{permisRole}, \textit{Staff} \rangle$. This triggered the violation `lgTShortAccess`, whereby the controller calculates the subject's impact level as 0.8. Several solutions are now applicable given this impact weighting, including solutions that result in policy adaptation. However, as the subject has been identified as the source of two previous violations, but is the only subject that has abused their *permisRole* attribute of *SysAnalyst* and *Staff*, solution S2 is enacted. This results in the complete removal of access for subject *emp0003*.

As a result of subject adaptation, malicious subjects were mitigated given the abuse of access rights. Moreover, subject adaptation ensured that non-malicious subjects remained able to request and gain access to protected resources, as evident by the rate of access maintained for *all.staff*, as shown in Figure 5.8.

Exp: 3 & 4, Deactivated Contractor Identity Provider Effector

In experiments Exp3 and Exp4, the contractor's identity provider effector was disabled, and the controller was placed under normal and high loads, respectively. Under either load the controller consistently identified violations, and performed mitigation responses, in relation to 23 attack steps.

Figure 5.9 portrays the malicious changeload for Exp3 (i) and Exp4 (ii). Table 5.2 details the attack steps and mitigation for Exp3, where steps 1 to 5 are omitted as the same violations are previously described in Table 5.1. Exp4 can be viewed in Appendix B.3.3 (Table B.6), and a break down of solution selection is described in Appendix B.3.2. In both experiments there are three significant turning points in relation to detection and mitigation of violations. These being:

1. The failure to mitigate initial violations due to lack of control (steps 6 to 17).

2. The escalation to considering a wider scope of solutions in response to persistent violations (steps 11, 14, 17, 20, and 21).

3. The selection and enactment of successful policy adaptations to halt persistent violations (steps 19 and 22).

Given the fact that the contractor identity provider is deactivated, solutions S1 and S2 (subject adaptation, see Section 5.5.1) are ineffective in mitigating malicious behaviour by contractors. However, the controller persists in attempting this solution until the malicious subject exhibits a greater impact to the organisation, as seen in step 11.

Now that the subject exhibits a greater impact, a wider set of solutions to mitigate the subject's behaviour can be selected. However, as a result of the

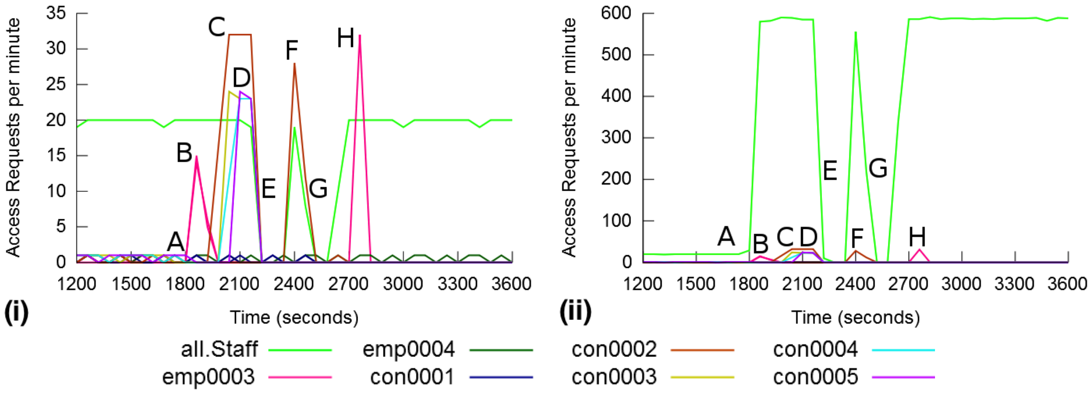


Figure 5.9: (i) Exp3 normal load, (ii) Exp4 high load

Step	Subject	Impact	Violation	Identified Solutions	Selected Solution	React Time (ms) (Avg, Std)	Adapt Time (ms) (Avg, Std)	Result
6	con02	0.07	empShortRead	S0	S0	1.6, 0.5	N/A	N/A
7	con02	0.27	empShortRead	S1	S1	134, 47.4	129.2, 16.3	0
8	con03	0.07	empShortRead	S0	S0	4.8, 3.1	N/A	N/A
9	con04	0.07	empShortRead	S0	S0	1.6, 0.5	N/A	N/A
10	con02	0.6	empShortRead	S2	S2	99.4, 24	124.2, 31.2	0
11	con02	1	empLongRead	S2,S3,S4,S5	S2	405.2, 99.3	102.4, 14.3	0
12	con03	0.27	empShortRead	S1	S1	82.2, 40.5	102.4, 31.7	0
13	con05	0.07	empShortRead	S0	S0	1.8, 0.4	N/A	N/A
14	con02	1	empShortRead	S2,S3,S4,S5	S2	319.4, 24.1	99.8, 20.1	0
15	con04	0.27	empShortRead	S1	S1	39.6, 10.6	89.9, 42.2	0
16	con03	0.6	empShortRead	S2	S2	37.4, 5.6	72.6, 8.3	0
17	con02	1	empShortRead	S2,S3,S4,S5	S2	153.6, 24.4	94.8, 44.2	0
18	con05	0.27	empShortRead	S1	S1	33.8, 8.2	69.2, 6.7	0
19	con03	1	empLongRead	S2,S3,S4,S5	S2(F),S3	154.6, 32.1	985.6, 60.5	1
20	con02	1	empShortRead	S2,S3,S4,S5	S2	280.8, 58.1	104, 21	0
21	con02	1	empLongRead	S2,S3,S4,S5	S2	196.8, 31	82.6, 7.7	0
22	con02	1	empShortRead	S2,S3,S4,S5	S2(F),S4	185.6, 71.1	805.4, 36.4	1
23	emp03	0.8	lgTShortAccess	S2,S3,S4,S5	S2	194.6, 47.8	143.4, 77.3	1

Table 5.2: Exp3: Adaptation with normal load (contractor IdP effector failure)

controller's solution selection activity (see Appendix B.3.2) solution S2 is chosen. This is because all other solutions exhibited a greater cost to the behaviour identified, meaning that only solution S2 could be considered as a candidate solution (which ultimately fails).

It is only by step 19 that the controller identified that a policy adaptation (solution S4, partial removal of trust in the contractor identity provider) exhibited a lower cost over detected violations. In this step, S2 and S4 are ranked and enacted in order of lowest cost. As such, the controller attempted to enact S2 first, in the form of a SOAP message to the contractor's identity provider effector. The solution ultimately failed, which resulted in the controller attempting to enact the next ranked solution, S4. This resulted in a policy adaptation (point E, Figure 5.9) where trust was removed from the contractor identity provider in issuing certain access rights.

Further violations are mitigated in a similar fashion (post step 19), where the contractor supervisor *con0002* is mitigated via a policy adaptation (point G in Figure 5.9). The only exception being that the remaining attacker from *LGZLogistics* (*emp0003*) is mitigated in exactly the same manner as Exp1 and Exp2. This is due to the fact that whilst the contractor's identity provider effector was deactivated, *LGZLogistic's* identity provider effector remained active, allowing for successful subject adaptation of *LGZLogistic* subjects.

An important observation is the impact of policy adaptation, where a temporary sudden drop in access requests was observed (Figure 5.9). Policy adaptation results in model transformation and serialisation of persistent policy documents, which are activated via a restart of the PERMIS authorisation service. As a consequence there is a short period of time in which no access can be processed whilst the authorisation service restarts[1].

5.5.4 Summary of Results

For each experiment, adaptation resulted in preventing the detected malicious subject(s) from gaining further access. This was achieved through removing the abused access right (assigned attribute), removing all of a subject's access rights at identity provider level, or removing varying degrees in trust of the contractor identity provider.

In Exp1 and Exp2, no impact was made to authorisation services in terms of the service being able to perform its duties. This reflects the fundamental design of SAAF, which promotes separation of concerns between adaptation and authorisation. However, in Exp3 and Exp4, the availability of the authorisation service was temporarily impacted on two occasions, as a result of policy adaptation. This availability issue is a result of a limitation in the PERMIS standalone service, whereby it cannot activate new policies without being restarted.

In both sets of experiments, the impact on identity provider services was negligible. There was no observable rise in latency in subject authentication and attribute release as a result of identity provider adaptation. However, malicious subjects were impacted at identity provider level, in terms of attribute removal, yet this was the desired consequence of adaptation.

[1]To accommodate for this, upon encountering a failed connection to the authorisation service all subject access requests were subject to a 3 minute pause, before changeload was resumed.

Subject versus Policy Adaptation

The experiments portray two types of scenarios that exemplify subject and policy adaptation. A scenario where a controller is capable of performing subject adaptation against all identity providers, and a scenario where the controller is limited in performing subject adaptation (requiring policy adaptation).

Subject adaptation is seen as the economical choice, whereby malicious behaviour can be mitigated with no impact to non-malicious subjects. When subject adaptation was possible, the malicious subjects' behaviour was mitigated almost immediately, preventing future violations. However, where subject adaptation was not possible, subjects were capable in repeating violations until the controller identified that the cost of unresolved violations warranted policy adaptation.

Policy adaptation has far greater consequence in comparison to subject adaptation, which is calculated (in part) by the number of non-malicious subjects that will lose access to resources as a result of change. For example, Table 5.2, step 19, represents the tipping point between the impact of the contractors' persistent malicious behaviour, outweighing the impact in *LGZLogistics* removing the trust in the contractor identity provider issuing the attribute $\langle permisRole, Contractor \rangle$.

Regardless of type, each adaptation results in a concrete change to the authorisation infrastructure. Changes ultimately control the outcome of future access decisions, and whether or not subjects can be authorised in accessing resources.

Performance

Whilst benchmarking the performance is not an objective for this thesis, the performance observed in the experiments requires some explanation. Of particular interest is the performance of different types of adaptation. Performance is directly related to the number of violations the controller can identify, the size of its access control model, the number of previously identified violations, the number of solutions applicable to an identified violation, and the type of adaptation to be performed. For each experiment, these factors remained persistent, relative to the given experiment step.

A concern was the high standard deviation observed (max 99ms) between experiment runs for some adaptations, specifically in regards to the time it took the controller to react and decide upon solutions. Steady state time was used to place the controller in a warmed up state. However, due to a mix of factors the standard deviation failed to improve. Some of these factors include network fluctuation between communication of the prototype controller and its effectors, the triggering of Java garbage collection and Java's code optimisation, and that the controller prototype is yet to be optimised. To compensate for this, further experiment runs are required, but were limited to 6 runs per experiment (due to the hour long runtime of each run).

Focusing on experiment Exp3 (normal load), Table 5.2, the first time the controller performs adaptation (step 5) took 329ms to react. However, the same adaptation made against a different subject at step 7 is much faster (134ms), and throughout remaining steps performance in similar adaptation improves consistently throughout each run. This is due to Java optimising frequently used code.

When the controller repeatedly processed multiple solutions, faster reaction times (e.g., step 11 compared to step 5) were observed, and again follows similar improvements in performance over the course of the run. Lastly, it was observed that policy adaptation takes much longer to enact (step 19, 986ms) when compared to subject adaptation (step 5, 164ms). This is due to execution of model transformation programs, deployment of a policy on the authorisation service, and the initial failure in carrying out a subject adaptation.

5.6 Experiment Discussion

The *LGZLogistics* case study has provided a scenario for demonstrating and evaluating the detection and mitigation of insider threat. Through execution of this case study, this chapter has demonstrated the SAAF prototype's capabilities in:

1. Realising self-adaptation within existing technologies;

2. Detecting malicious behaviour through the observation of access and resource change;

3. Consistently mitigating malicious behaviour through the automated adaptation of access rights and authorisation policies under varying operational conditions.

Of note, self-adaptation has been achieved within a federated environment, where challenges exist as a consequence of multiple management domains. Probes and effectors are shown to facilitate automated adaptation across these management domains, where there is arguably a greater need for automation given the fact that federations contain large and unknown user bases.

5.6.1 Evaluation Approach

The experiment was designed to demonstrate the robustness of the SAAF prototype in mitigating malicious behaviour under repeatable conditions. This required simulating a known malicious changeload in which to trigger self-adaptation, and capture responses from the SAAF prototype.

The simulation approach allowed for the evaluation of the SAAF prototype within a large scale deployment, akin to a small to medium sized organisation. This was critical to providing evidence of the prototype's feasibility in operating within the real world, and that the prototype would consistently mitigate violations in a robust manner. As such, it was observed that the prototype was capable in mitigating violations when operating under high loads, and when faced with non-cooperating management domains.

A clear advantage of the approach was to demonstrate the prototype's ability to select and enact appropriate solutions within a complex scenario. This was demonstrated by the escalation to high severity solutions (i.e., removal of trust in an identity provider organisation) when faced with persistent malicious behaviour and the failure to mitigate by alternative means. This also had the benefit of highlighting the consequences to an non-cooperating organisation, where they risk losing access to a resource in its entirety.

The simulation approach does have several limitations, indicative of the nature of simulation. Specifically, simulation presents a certain amount of bias whereby the violations performed are known, and the prototype controller can be configured in an optimum way to best handle such violations. Therefore, the simulation approach can only be seen as a means to demonstrate the prototype's robustness in handling known violations. What it cannot evaluate is how the prototype will handle unknown malicious behaviour, and in particular, unpredictable change within its environment and system. This type of behaviour is challenging to simulate, where it is necessary to evaluate the prototype in a live environment.

5.6.2 Detection and Adaptation

The goal of this thesis is not to improve upon detection methods, rather, demonstrate a new process in handling insider threat. With that said, detection within the SAAF controller prototype is worth discussing. The SAAF prototype utilises detectors to identify known types of attacks, typically focused on thresholds, which is a common approach in detection of malicious behaviour [37, 144].

Adopting a threshold approach to detection has the advantage of clearly detecting extremes in user behaviour, as it is assumed detection rules are formed by experts certain in the perception of normal and abnormal behaviour. Therefore, if user behaviour violates these rules, malicious behaviour is identified. However, it does require experts to be absolute in their decision for malicious behaviour, which could be seen as restrictive. In addition, if a rule is incorrect or inappropriate for the current state of the system, there is the potential for many false positives. For example, a subject that conforms to current behaviour rules specified for their role may be assigned to a new project. As a result, their legitimate behaviour may violate behaviour rules. Clearly a challenge for SAAF is to employ detection techniques that can evolve and accommodate such legitimate changes in behaviour.

Past approaches

In preliminary implementations of SAAF [5, 6], violations led to the immediate decision to perform an adaptation. This is problematic, as different violations may yield variable impact to an organisation (e.g., a subject abusing their access rights on resource ‘X’ poses far greater impact over resource ‘Y’). Moreover, the culmination of different behaviours may require a solution with a greater impact (e.g., the complete removal of access of a subject) over one with a smaller impact (e.g., warning the subject or removing a single access right).

Therefore, to enable appropriate selection of solutions, SAAF’s current approach utilises cost sensitive modelling [136] to assess subject impact and impact of solutions. This approach has allowed the aggregation of multiple violations before enacting an appropriate solution. Multiple occurrences of violations arguably strengthens the perception in the subject being malicious[2], as well as judge the extent of appropriate adaptation. Lastly, through this approach, the deploying

[2]One exception to this is if the behaviour rules specified are incorrect, which is addressed as part of SAAF’s limitations.

organisation has the ability to fine tune the enactment of solutions, through specification of cost of behaviour and solutions.

Triggering Adaptation from Observation of Access

In the experiments discussed, the SAAF prototype considers the metric of rate of access requests as the primary environment property in identifying malicious behaviour. Whilst using this metric has shown to be successful in identifying attacks, for it to be efficient the level of access control must be fine grained. In addition, a subject's ability to access a resource should be determined by short term (or one time use) credentials issued by their identity provider.

This presents two concerns. Firstly, if it is not possible to implement fine grained access control, a greater emphasis must be placed on resource probes. For example, a subject granted holistic 'access' to the empDB resource initiates an authorised session in which the subject can 'read', 'write', 'delete', and 'create' multiple times. Utilising a probe on the authorisation service alone would simply identify a single access request, foregoing a large amount of information that could be used to detect attacks. In this instance, a resource probe is essential for capturing the missing information[3].

Secondly, if access is awarded based on long term credentials (e.g., a digital certificate), the ability to stop a subject's future access is delayed until the end of a subject's authenticated session (within their identity provider). Whilst the case study does not demonstrate the use of long term credentials, it is an important aspect to consider, as adaptation in this case requires actions (at effector level) to revoke long term credentials (e.g., revocation of subject X.509 certificates, and update to a revocation list). In effect, the action must result in a resource policy enforcement point (PEP) requesting the release of a subject's attributes (access rights) as they are updated. This can be achieved through additional effectors within the identity provider, yet would require the resource PEPs to make use of such revocation lists.

Similarly, in these experiments the prototype controller only considers successful access (i.e., permitted access) to identify malicious behaviour. This focuses on the adaptation of subject access in accordance to the use of valid access rights that subjects' own. Multiple deny requests could indicate malicious behaviour whereby a subject is trying to identify vulnerabilities in access, similar to a subject scanning a network for open ports [120].

[3]To strengthen this position, Chapter 6 demonstrates the necessity of resource probes.

Selecting Solutions for Adaptation

The experiment demonstrated the selection and escalation of solutions in response to detected violations. Whilst this was successful and ultimately viewed as enacting 'appropriate' solutions to violations, the cost sensitive modelling approach employed has several limitations.

Notably, the approach relies upon weighting solutions by a perceived cost of negative impact to an organisation, which is then compared to a perceived cost of subject activity (as conveyed by Table B.4). Although not observed within the experiment itself, there is potential for multiple solutions in conjunction with observed behaviour to present identical costs (i.e., benefits) to an organisation. In SAAF's current form, no solution would be prioritised, and as a default the last solution processed (of equal measure) is selected. This strengthens the need to improve upon the cost sensitive modelling approach, where additional criteria (beyond cost) is factored into solution selection.

Bottlenecks in Adaptation

One property not exemplified by the discussed experiments, is the presence of bottlenecks. Given that this implementation of SAAF is a prototype, a notable deficiency in its design is its inability to consider multiple violations during a single iteration of its feedback loop. If violations are detected during the prototype's current analysis of behaviour, multiple violations are queued, analysed, planned and executed in a sequential manner. The result of this is increased response times in mitigating behaviour identified in the aforementioned manner, due to failed or redundant adaptations if a previous adaptation has already resolved the violation.

This is a general challenge facing self-adaptive systems, whereby a self-adaptive system should address how to handle change whilst it is already responding to previous change. In regards to SAAF, it is necessary for future refinements to group and analyse violations at every step within its feedback loop (as demonstrated by the Rainbow Framework [53]), reviewing any updates to the state of access and detected violations prior to mitigation. Adaptation time is still likely to increase. However, this would allow SAAF to make more informed decisions and avoid enacting redundant adaptations.

5.6.3 Federation Challenges

Exp3 and Exp4 demonstrated the consequence of when a SAAF controller is limited in performing subject adaptation. Whilst policy adaptation occurred at the point where the impact of violations outweighed the impact of removing trust in the contractor identity provider, Exp1 and Exp2 showed that the attacks could equally be resolved more effectively on an individual scale. However, as demonstrated in a previous paper [6], policy adaptation is necessary in regards to large numbers of subjects committing malicious behaviour (i.e., when a service provider, despite mitigating attacks from individuals, is seeing persistent attacks from a given identity provider).

Associated with this is the reliance on an authorisation service's ability to validate subject credentials. Credential validation enables solutions that manage the trust in identity providers. Without it, SAAF is limited in performing fine grained adaptations against identity providers, resorting to policy adaptation that may impact all subjects from all identity providers.

In regards to control over observation, in the *LGZLogistics* case study, there are no probes deployed within the contractor identity provider. This highlights the fact that many third party organisations may not provide a complete view of subject attributes, in particular, the release of personal identifiable data. As a result, the model of access generated is representative only of subjects that have requested access, and what valid attributes (post validation) have been used.

Generating a model of access in this fashion is limited, as the model only contains a view of active subjects, and does not present a complete view of access (i.e., subjects that have yet to request access will not be modelled). The repercussion of this is that the calculation of impact of solutions against the current modelled state of access may well be higher, due to an incomplete modelled state of access.

One potential solution that overcomes the problem of restricted observation of subjects, is a probe at the identity provider that operates in a similar fashion to SAAF's SimpleSAMLphp effector [7] (see Appendix B.2). Applying the same concepts to a probe, an identity provider could control what subject information is released, how subjects are identified, and which subjects can be observed. This would allow for synchronised models of access within federated environments. However, a potential risk is if an identity provider is hijacked, information sent via a identity provider managed probe could become unreliable.

5.7 Summary

In summary, this chapter has presented an evaluation of the Self-Adaptive Authorisation Framework (SAAF) through the simulation of a fictitious case study of insider threat. As part of this evaluation, a malicious changeload has been formally defined in the context of authorisation infrastructures in order to describe scenarios of abuse in access control.

The malicious changeload, relevant to the case study, was then executed to stimulate self-adaptation within a federated authorisation infrastructure. A deployment of the SAAF prototype was then evaluated in mitigating the malicious changeload under various operational conditions. These included changes to the runtime load of the authorisation infrastructure and the SAAF autonomic controller, along with restrictions in available probes and effectors (simulating the presence of a non-cooperating contractor organisation).

The evaluation demonstrated the SAAF prototype's robustness in handling abuse of access under repeatable conditions, where the prototype was shown to consistently mitigate abuse under normal and high loads. In addition, when faced with limitations in enacting adaptation, the prototype was shown to escalate its selection of policy adaptations in order to overcome failures in subject adaptation. Whilst subject adaptation was shown to create minimal impact (in terms of consequence to non-malicious subjects), it was in these conditions that policy adaptation becomes necessary in order to halt the abuse of access. Finally, SAAF has been demonstrated in mitigating the abuse of access in federated environments, where the use of a domain managed effector has been key to enabling adaptation across multiple management domains.

A limitation in evaluating self-adaptive systems through simulation is the inability of dealing with a wide range of changes that are representative of unexpected subject behaviour, and how subjects may react to adaptation. Whilst case studies of insider threat can provide insight to attack scenarios, they do not consider the runtime consequence of mitigation. To evaluate this, Chapter 6 defines a runtime experiment in which real users are invited to carry out malicious behaviour against an organisational resource, protected by a self-adaptive authorisation infrastructure.

Chapter 6

Evaluating SAAF through Gamification

6.1 Introduction

The simulation of insider threat case studies is limited regarding the evaluation of self-adaptive authorisation infrastructures. This is because they would not be able to portray an accurate perception of reality. Simulation has demonstrated partial feasibility of the Self-Adaptive Authorisation Framework (SAAF), including how SAAF mitigates malicious behaviour under prescribed conditions. However, simulation can only evaluate a fraction of the scope of change and types of abuse representative of the real-world.

An important step in evaluating SAAF is demonstrating its ability to mitigate abuse of access when faced with uncertainty. Moreover, it is necessary to evaluate the consequence of self-adaptation in terms of how human users respond to the presence of a feedback loop. In light of a feedback loop, users may change their behaviour, for instance, to mask their malicious activity. Such change is unpredictable, resultant of intelligent user interaction, and therefore challenging to simulate.

Given SAAF's experimental status, to consider deploying SAAF in a real organisation would be inherently risky. Therefore, this chapter presents an approach whereby gamification [60] is used to emulate a real-world environment. Gamification is the use of online games to solve complex problems and generate meaningful data as a consequence of human player participation. It is a crowd sourcing technique to capturing large volumes of data by using the premise of a game to motivate human participation.

For evaluating SAAF, gamification is used for generating diverse and unpredictable data from real user activity. In particular, it enables the observation of SAAF mitigating cases of abuse at runtime, and the observation of user activity post mitigation. As such, the success of mitigation can be validated, along with evaluating the consequence of self-adaptation by analysing user response to mitigation.

The contribution of this chapter is an approach to evaluating self-adaptive systems through gamification [60]. A key feature of the approach is the ability to observe user activity pre- and post-adaptation, in order to evaluate the runtime consequences of self-adaptive systems. Gamification is demonstrated in evaluating the Self-Adaptive Authorisation Framework (SAAF) by way of deploying an online game as a protected resource within an authorisation infrastructure. Human participants are assigned a set of access rights related to the authorisation of actions within the game. Participants of the game are then invited to choose to act honestly or dishonestly. Dishonest activity is viewed as synonymous to malicious behaviour, requiring mitigation.

The rest of this chapter is structured as follows. In Section 6.2 the objectives and scope of the online experiment are presented. Section 6.3 describes the design of an online game, in which diverse and unpredictable behaviour can be observed. Section 6.4 discusses the deployment of the game in a self-adaptive authorisation infrastructure. Section 6.5 describes the phases and execution of the experiment within the game environment. Section 6.6 discusses the results of the experiments. Section 6.7 discusses the evaluation approaches presented in this thesis, identifying limitations. In Section 6.8, a summary of the chapter is provided. Finally, Appendix C contains additional results of the experiment.

6.2 Objective and Scope

The objective of this evaluation is twofold. The first is to demonstrate SAAF in a live deployment, whereby the observations and actions performed by SAAF have a real consequence to human users accessing a resource. The second objective is to generate data that portrays the effectiveness of self-adaptation in mitigating observed attacks, include data relating to the consequences of self-adaptation. Such data is necessary since user behaviour is unpredictable, where simulation alone only provides a constrained view to how users may react to adaptation.

To fulfil these objectives, an experiment is conducted whereby human users are invited to participate in an ethical game of hacking. Users are asked to play an online game based on the classic board game of *Snakes and Ladders* [116]. The use of a game follows the concept of gamification [60], whereby games are used to engage users in order to solve complex problems. For the purpose of the experiment, the game is used as a platform to enable users to perform malicious activity. Users are given the freedom to play the game and to choose to act honestly or dishonestly, such as exploiting vulnerabilities in the game resource or host server.

With this in mind, the experiment conducted within this chapter seeks to answer the following research question: *are self-adaptive authorisation infrastructures capable of mitigating acts of malicious behaviour, and what are the consequences of self-adaptation?* To reflect on this problem statement, the rest of this section identifies a set of hypotheses, as well as the scope of the evaluation.

Hypotheses

(Main) *1. Self-adaptive authorisation will mitigate malicious activity, whilst limiting future attacks.*

(Subsidiary) *2. An experienced subject is capable of carrying out sophisticated and complex attacks.*

(Subsidiary) *3. The behaviour of a malicious subject will change in response to adaptation, in order to circumvent future detection and mitigation.*

Scope of Evaluation

This evaluation is specific to the mitigation of malicious subject activity related to the abuse of access. The proposed experiment will seek to evaluate the effectiveness of subject adaptation, in the context of an online game.

There are some limitations to this evaluation. Notably, human participants are aware of the true nature of the experiment, and as such, it cannot be said that a participant of the game is representative of a ‘true’ malicious insider. Basically, participants are aware that they can be malicious to win the game. However, the game as a whole is representative of the actions of a malicious insider.

A further limitation is that the proposed experiment is not positioned for the evaluation of policy adaptation. Whilst policy adaptation is likely to be successful in mitigating abuse against the game resource, it limits the potential for the amount of data generated. For example, adaptation of a policy may cut off access to the game in its entirety once the game has suffered a certain amount of abuse. In addition, policy adaptation is most effective within the context of a real organisation, where users adhere to a specific role within that organisation. Given the gamification approach and the reliance on anonymous human participants, it would be challenging to verify that a player is indeed adhering to a particular role within an organisation.

6.3 The Game of Snakes and Ladders

Snakes and Ladders is a classic board game which requires players to roll a dice and move their player from a starting square to a finishing square. Players can land on certain squares resulting in them being pushed ahead (i.e., travelling up ladders), or moved backwards (i.e., falling down snakes). The first player to land on the finishing square wins the game, which is purely based on chance.

Considering the objectives of the evaluation, the concept of Snakes and Ladders was chosen for a variety of reasons. These include:

- Familiarity and ease of use;
- The ability to collect a wide range of data from player interaction;
- Contains a clear set of rules that honest players are expected to follow, which can be used to verify the existence of malicious behaviour;
- Has a set of actions that can be protected by an authorisation infrastructure (e.g., game start, roll, move, end).

It can be argued that a game of Snakes and Ladders is not a realistic portrayal of real world resources. However, the game itself represents many of the processes and concepts a real resource would exhibit. These include the ability for a subject to authenticate and gain access to the resource, perform multiple tasks in light of some goal, and have an impact against the resource itself.

The game is a mechanism to enable such concepts, likened to that of a workflow within a resource. Honest players are expected to follow the workflow of the game, and by doing so, the player is eventually able to finish the game. Player activity within the game generates data representative of the player's behaviour. As such, it can be used to capture a number of features of user activity, just as it would be possible to observe in real world resources (e.g., activity within a database).

Whilst Snakes and Ladders presents a narrower scope in the type of changes that can affect the game in comparison to a real world resource (e.g., a database), the rules of the game act as requirements of the user. These requirements provide a base to validate behaviour against. In addition, the game itself will appeal to a wider audience, allowing for a range of attack profiles, including, non-technical opportunist profiles, to technical and informed profiles of attack.

The rest of this section discusses the design of the Snakes and Ladders game as a protected resource. In addition, vulnerabilities are discussed that are purposely left within the game to enable dishonest play.

6.3.1 Game Design

The Snakes and Ladders game is designed in the form of a web application, hosted on an Apache web server, and accessible via any modern web browser. It is built using web based technologies, and has two purposes:

1. Enable participant sign up, whereby a participant becomes a subject of the game's organisation and is given a set of access rights to play the game.

2. Facilitate online play of a game instance of Snakes and Ladders, whilst logging subject interaction with the game.

Figure 6.1 portrays the general activity flow of the web application. To simulate the notion of an 'insider', participants must create an account. Upon signup, each subject is issued with the same level of access (in the form of an X.509 certificate) that initially provides the subject with full access to the game.

Once a participant has been provided subject status, they are capable of authenticating, then requesting and playing instances of the game. Players request access to start a game, in which a game instance is returned to their client. Game logic is handled via both client side and server side processes. The game interface (Figure 6.2) is dynamically updated in order to reflect the subject's actions and state within the game.

Subjects are capable of performing a set of protected actions within the game resource. These actions are expected to be governed by an external ABAC authorisation service, which validates a subject's level of access in relation to a requested

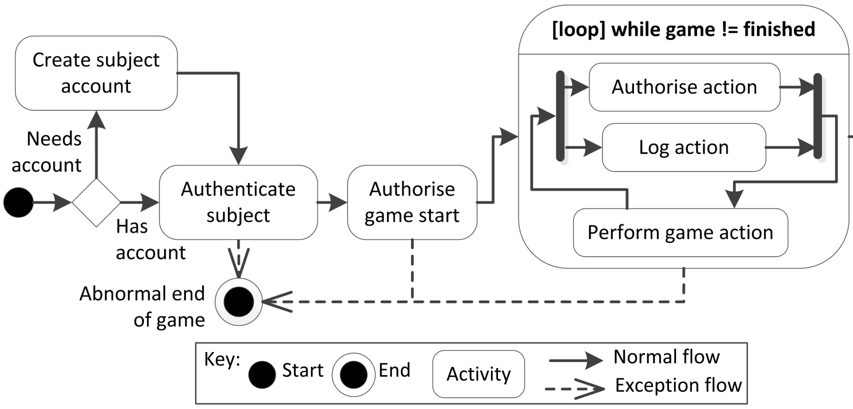


Figure 6.1: Activity flow of the game resource

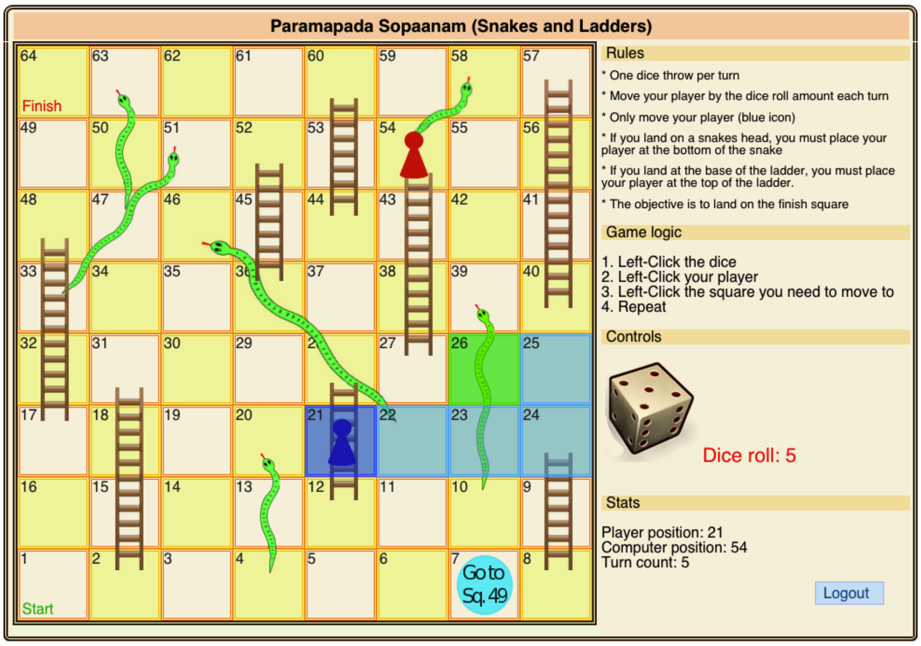


Figure 6.2: Screenshot of the implemented Snakes and Ladders game

action. An authorisation policy is expected to define the criteria of access, and should protect access against the following actions:

- *Start Game*, the ability to request access to a game instance;
- *Roll Dice*, the ability to roll the dice, dictating the amount of squares a player should move on the game board;
- *Move Player*, the ability to move the player within the game board;
- *Use Ladder*, the ability to travel up a ladder should the player land on a square at the base of a ladder;

- *Use Bonus*, the ability to use a bonus move which moves the player towards the end of the game, only if the player lands on the bonus square;
- *End Game*, the ability to finish a game when landing on the 'finish' square.

Once access has been authorised for an action, the player is able to perform the action within the game. The process of authentication and authorisation is enforced by a policy enforcement point (PEP) built into the game resource. Each action the player carries out is then logged (along with metadata) and interpreted in a backend database, providing context to any authorisation request.

To enable competitive play, an online scoreboard is used to monitor the achievements of players over the games they play. As a result, the core objective of the game is for players to attempt to beat the game in as few turns as possible, whilst competing against each other.

6.3.2 Vulnerabilities

The game is designed to facilitate a range of malicious activities. Players are capable of performing malicious activities through exploiting known and unknown vulnerabilities. The game itself is considered a honeypot [132], where a subject that exploits known vulnerabilities within the game is likely to garner some malicious intent (i.e., to complete the game unfairly). These 'known' vulnerabilities exist at the level of the game resource (i.e., the game's interface, the game's code, and the game sessions), and are further discussed as follows.

Game Interface Vulnerabilities

Game interface vulnerabilities symbolise the simplest form of attack, whereby subjects identify bugs within the game logic simply through interaction with the game itself. They showcase attacks to that of an opportunistic attacker, and are easily identifiable through playing the game.

- The dice can be rolled multiple times between moves;
- A player can land on any square within the given dice roll range;
- A player can choose not to go up a ladder;
- A player can choose not go down the snake;
- If the player lands at the base of a snake, the player can travel up the snake.

Code Injection Vulnerabilities

Code injection [72] depicts a more advanced class of attack, where players must have an understanding of how a client operates with a server. Through code injection, the player is capable of modifying the game logic in order to gain an unfair advantage within the game.

To enable code injection exploits, participants must play the game in an environment where they have some access to the code. As a result, through use of JavaScript and PHP, a game instance can be delivered to the participant's client web browser, whereby parts of the game rely on client-side execution. AJAX routines are used to facilitate state changes in the game between the client and the server. This enables the authorisation of subject actions, whereby an AJAX routine communicates with the server, and the server forwards on relevant access requests to an authorisation service.

With the appropriate tools a subject is capable of changing the game logic. For instance, the subject could inject code in order to roll an impossible dice roll value, change the player's starting square position on the game board, move to any square on the game board, or simply trigger the game end conditions.

Given the readability and nature of JavaScript, code injection would be a relatively simple task. In order to ensure there is a reasonable challenge to participants in injecting code, the JavaScript code is obfuscated [80], making it much harder to interpret for a novice programmer.

Session Vulnerabilities

A final scope of vulnerabilities is session poisoning [108]. Session poisoning involves attacks where a client injects data into a session held by a server. Such injection will change the client user's state between requests to the server, potentially overriding the need for authentication and authorisation.

As players progress within the game, their activity is held within a server side session. The session is essential to maintaining transitions of state between a client's HTTP requests to the server, and is required in order to log player activity. Players can therefore perform session poisoning attacks to change the state of play.

Session poisoning attacks are enabled through the exploitation of server side scripts that handle policy enforcement and activity logging. These scripts expect POST data to update player activity within the session. Through monitoring a game's POST data requests, it is possible to identify how a session could be manipulated.

Summary Attack Model

Given the described known vulnerabilities, abuse of access can be modelled as a high level attack tree [91]. Listing 6.1 describes the attack tree of a player abusing their access rights in order to win a game via malicious means. This model of attack defines the scope of malicious behaviour to be mitigated in this evaluation.

```
Goal: Win a game through exploitation of vulnerabilities
Precondition: Attacker is an insider holding a game account
Attack:
  AND:1. Authenticate with identity service
      2. Gain authorisation to start game
        OR:
          1. Exploit glitches within the game's interface
            OR: 1. Roll more than 1 dice roll per turn
                2. Ignore snakes
                3. Ignore ladders
                4. Travel up a snake
                5. Land on any square within dice roll range
          2. Inject code to change game behaviour
            OR: 1. Reduce size of game board
                2. Inject invalid dice roll
                3. Perform an invalid move
                4. Prematurely trigger game end
          3. Poison session to falsify game play
            OR: 1. Exploit AJAX endpoints
                2. Exploit session variables in HTTPS GET requests
                  OR: 1. Falsify roll action
                      2. Falsify move action
                      3. Falsify game end action
Postcondition: Attacker finishes game with unfair advantage
```

Listing 6.1: High Level Attack Tree for Snakes and Ladders

It is recognised that attackers can perform other patterns of attack within the game environment (including the entirety of the authorisation infrastructure). For example, an attacker does not need to rely on their access rights alone to attack the game resource. An attack tree could exist where an attacker bypasses authentication via performing an SQL injection attack, potentially enabling the attacker to falsify game records (i.e., create a fictitious game) or delete game records entirely. Other forms of attack could be made against the infrastructure itself, such as identifying the SAAF controller's endpoints in order to perform a denial of service attack (i.e., disrupt the controller's ability to detect and adapt), or falsify the controller's input or output. These types of attacks, whilst worth investigating in future work, remain out of scope of this evaluation.

Limitations

Several trade-offs were made in order to enable malicious behaviour within the game. In a real-world environment, developing a resource that has known vulnerabilities is inherently insecure. In addition, executing code on the client machine could be considered rare. However, for the purpose of the experiment it was necessary to use client side technologies (i.e., JavaScript) to present an achievable environment for subject's to inject code.

An alternative approach considered was to deliver the game resource in a compiled state (e.g., a Java application). However, this would reduce the scope of players capable in injecting code (due to the technical knowledge required), and ultimately reduce the amount of data generated within the experiment.

Lastly, the fact that subjects are capable of injecting code in the client means that authorisation could be bypassed. A subject could manipulate the game logic to prevent the AJAX routines initiating calls to the resource's policy enforcement point (PEP). As a result, any games that bypass authorisation are out of scope of the evaluation. This is due to the fact that SAAF assumes that resources and the authorisation infrastructure in place are operating in conformance to access control.

6.4 Deployment

The game is deployed into the environment of a fictitious organisation, whereby it is protected by an Attribute-Based Access Control (ABAC) authorisation infrastructure. The following describes the configuration of the authorisation infrastructure, configuration of a SAAF prototype controller, and data to be logged.

6.4.1 Self-Adaptive Authorisation Infrastructure

The infrastructure is comprised of three virtual machines (VMs), as shown in Figure 6.3. Each VM is configured to run Ubuntu v12.04.5 TLS, with 1024MB RAM, whereby a VM is configured to either serving the game resource, managing subject identities, or providing authorisation.

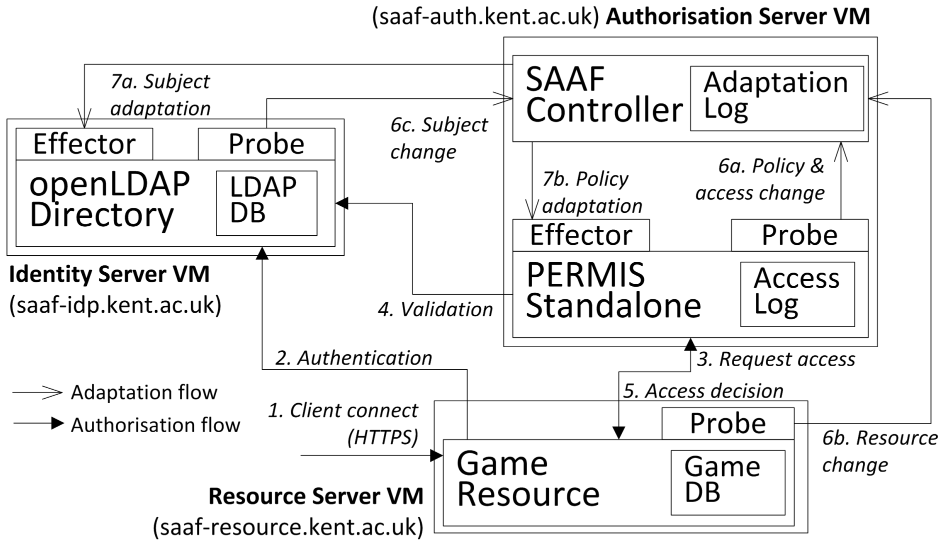


Figure 6.3: Game experiment authorisation infrastructure

Identity Server

The **saaf-idp.kent.ac.uk** VM hosts an openLDAP directory, and a bespoke LDAP probe developed for SAAF. The LDAP directory maintains attribute certificates of each player account within the game. These represent a player's access rights in the form of a set of signed attributes. It is populated as and when subjects create accounts via the Snakes and Ladders game interface. The LDAP probe exists to monitor changes within the LDAP directory. Should a change be identified, the probe notifies the SAAF controller in order to ensure a synchronised model of access.

Authorisation Server

The **saaf-auth.kent.ac.uk** VM hosts an instance of the PERMIS standalone authorisation service, a probe and effector to monitor and adapt PERMIS, and the SAAF controller.

The PERMIS authorisation service is configured to operate in a 'pull' mode, whereby it retrieves the requesting subject's access rights directly from the identity service, per request. A single PERMIS ABAC authorisation policy exists, which defines a hierarchy of attributes. Each level of the hierarchy contains a scope of access, which is relevant to the game. In practice, given a subject's set of attributes, the subject is capable of performing a prescribed set of actions within the game, in conformance to the PERMIS policy.

The SAAF controller is deployed on this server to observe and manage access to the game. It observes data pushed from the resource probe, the LDAP probe, and the PERMIS probe, in order to model access and subject behaviour at runtime.

Resource Server

The **saaf-resource.kent.ac.uk** VM hosts the web application that contains the Snakes and Ladders game, an integral policy enforcement point (PEP), a probe, and a backend database. The resource is served via an Apache web server over a HTTPS connection to requesting client machines.

The probe is designed to identify malicious play interpreted within the game's backend database. The probe itself can be viewed upon as an external detector that informs the SAAF controller of malicious activity. It utilises SQL-based trigger rules to detect log entries that do not conform to the rules of the Snakes and Ladders game, expanding upon SAAF's own detection methods.

The **saaf-resource.kent.ac.uk** VM is the sole point of access for participants that wish to play the game, whereby direct access to **saaf-auth.kent.ac.uk** and **saaf-idp.kent.ac.uk** is prevented through firewall configurations. This means that a player can only indirectly communicate with authentication and authorisation services, via interaction with the game resource itself.

6.4.2 SAAF Controller Configuration

The SAAF controller is configured to maintain (at runtime) a synchronised model of access with the authorisation infrastructure (Figure 6.3). The controller is expected to detect and respond to violations of known malicious behaviour patterns, with the aid of external detectors deployed within the game resource.

The deployed version of the SAAF controller is an earlier version to the one evaluated in Chapter 5, due to the longevity of the game experiment. Notable differences include incompatibility with federated deployments.

Monitoring

The controller observes environment and system changes via the three deployed probes. All probes are configured to 'push' the following changes to the controller:

- *Subject change*: The LDAP probe notifies the creation of subjects, and any changes to subject access rights;

- *Policy change and access change*: The PERMIS probe notifies changes to the PERMIS authorisation policy, as well as logged requests and decisions in regards to authorisation;
- *Resource change*: The resource probe generates signature based patterns that capture malicious activity within authorised sessions of the game.

Upon receipt of change, the SAAF controller either updates its model of access ($ABAC_M$) through the use of model transformation programs, or updates its behaviour model to reflect player authorisation and resource activity.

Behaviour Policy

The controller's behaviour policy is defined in accordance to known game vulnerabilities. The trigger rules contained within the policy are characterised with relation to malicious patterns of access, and malicious patterns of activity within the game resource:

Access related

- `rollMoveViolation` - transaction / pattern based rule requiring that every request to roll should be followed by a request to move, triggering once a subject breaks this transaction more than 3 times within a short interval;
- `fastRollViolation` - pattern based rule that seeks to identify high frequency roll requests beyond human ability (i.e., scripted activity);
- `fastMoveViolation` - pattern based rule that seeks to identify high frequency move requests beyond human ability (i.e., scripted activity);
- `fastStartsViolation` - pattern based rule that seeks to identify a subject persistently restarting a game, typical of a subject aborting games until they receive a beneficial outcome.

Resource related

- `illegalMoveViolation` - signature based rule that triggers a violation if the resource probe indicates a player did not land on a square in accordance to a given dice roll;
- `ignSnakeViolation` - signature based rule that triggers a violation if the resource probe indicates a player ignoring the requirement to travel down a snake;
- `upSnakeViolation` - signature based rule that triggers a violation if the resource probe indicates a player travelling up a snake;
- `rollInjectionViolation` - signature based rule that triggers a violation if the resource probe indicates a player injecting code into the game client, in order to roll an unexpected roll value (e.g., roll value 500);

- `moveInjectionViolation` - signature based rule that triggers a violation if the resource probe indicates a player injecting code into the game client, for moving in an unexpected way (e.g., start square 1, end square 64);
- `bypassAuthsViolation` - signature based rule that triggers a violation if a subject attempts to bypass authorisation within the game resource.

Solution Policy

The controller is deployed with a fixed solution policy, which remains constant throughout the experiment. The tailorable solutions can be categorised by subject adaptation, and policy adaptation. The available solutions are summarised below:

- `S0: noAdaptation` is the default solution for when all other solutions cause greater impact over an observed behaviour;
- `S1: warnSubject` will notify a subject of their behaviour, typical for first offences triggering low impact violations (subject change);
- `S2: lowerSubjectAccess` reduces the level of access a subject has in conformance to the attribute hierarchy contained within the authorisation policy (subject change);
- `S3: removeAllSubjectAttributes` removes all attributes from a subject, typical for when subjects are persistently abusing access (subject change);
- `S4: removeAttributeAssignment` removes trust in an identity provider in issuing a valid attribute (policy change);
- `S5: removeAllAttributeAssignments` removes all trust in an identity provider in issuing valid attributes (policy change);
- `S6: deactivatePolicy` removes all access to all resources (policy change).

The solutions *warnSubject* and *lowerSubjectAccess* were introduced given the context of the game resource, and the use of an attribute hierarchy within the PERMIS policy. Given the extent of a subject's activity in violating the behaviour policy, it is expected that subjects are first warned of their behaviour, before being subjected to increased punitive measures. For example, lowering a subject's level of access within the attribute hierarchy until it is necessary to remove all access.

In regards to policy adaptation, it is expected that should the SAAF controller succeed in mitigating individual malicious subjects, no policy adaptation should occur. However, policy actions are configured should subject mitigation fail (e.g., effector failure within the identity service).

Execution

Lastly, once a solution has been selected, the controller mitigates malicious activity via either the generation and deployment of X.509 certificates or PERMIS authorisation policies.

- *Subject adaptation*: X.509 digital certificates are generated through a process of model transformation and serialisation to define a subject's new level of access, which is then deployed via the LDAP client embedded in the controller's executor component;

- *Policy adaptation*: PERMIS authorisation policies are generated through model transformation and serialisation to create a PERMIS policy document, which is then deployed via a bespoke PERMIS effector.

6.4.3 Logs

Considering the deployment of the game within a self-adaptive authorisation infrastructure, data is logged in regards to the following perspectives.

Game

Player activity is logged by the game resource, which is interpreted within its backend database. This activity provides context to authentication and authorisation requests, and stores the following information:

- Authentication requests via the resource and their corresponding success;

- Authorisation requests via the resource and their corresponding success;

- Roll activity (including contextual data, such as time, rate, roll value);

- Move activity (including starting position, end position, corresponding roll);

- Creation and completion of game sessions;

- An audit log of abnormal game behaviour, created via SQL triggers.

All player activity is linked to a player account (identified by their distinguished name assigned in the LDAP identity service), an authorisation request, and the player's authenticated session. Players can hold multiple accounts (stored within the LDAP identity service), allowing for a player to create a new game account when their current account has been adapted via self-adaptive authorisation.

In addition to the database, server logs are also maintained. Requests sent between clients and the server that hosts the web application are logged via the Apache server. SQL executed directly against the database is also logged, via the game's database server. These logs are necessary to validate that data logged within the database has not been tampered with, as well as enabling the identification of anomalous activity in regards to client / server requests.

Identity Management

The LDAP identity service logs all activity against the LDAP directory in the form of server logs. This includes the retrieval of attribute certificates (as part of PERMIS's credential validation), changes to attributes within an LDAP entry (due to adaptation by SAAF or human administration), the creation of new LDAP entries (when a participant creates an account), and lastly, subject authentication.

Authorisation

From start to finish of a game instance, a player is required to request access to perform specific actions. The PERMIS authorisation service logs all such requests, along with corresponding decisions based on a player's distinguished name within an identity service. These logs contain the subject's distinguished name (DN), the resource they wish to access, and the actions to be carried out.

Adaptation

The SAAF controller maintains two separate log files, along with trace logs that capture the state of access per each adaption made to its access control model. The first log file contains detailed information per cycle of the feedback loop, portraying identification of violations, analysis, planning, and execution. The second log file maintains information specific to the detection and mitigation of subject violations. This includes the identification of subject's that have committed violations, subject impact level at time of violation, the time at which the violation was detected, the time at which the violation was responded to, and the enacted solution.

6.5 Experiments

This section describes the experiments performed within the game environment, conveying data that demonstrates the SAAF controller monitoring and responding to diverse and unpredictable change. In addition, an evaluation of the experiments is given, structured according to the hypotheses previously presented (Section 6.2). The evaluation conveys a detailed analysis of attack data, correlating player pre- and post-behaviour with controller adaptation.

6.5.1 Experiment Execution

The experiment is executed over four phases, whereby human participants attempt to beat the game of Snakes and Ladders in as few turns as possible:

- **Control** - The game is released to a closed set of participants to observe honest play, for validation of detectors;
- **Phase 1** - The game is released within the School of Computing, University of Kent, requesting participants to play the game honestly or dishonestly;
- **Phase 2** - The game is released externally, advertised via academic and research community mailing lists, in addition to external Universities, requesting participants to play the game honestly or dishonestly;
- **Phase 3** - The game is again released internally within the School of Computing, University of Kent, requesting participants to play the game honestly or dishonestly.

In each phase, participants are provided the same guidance (Appendix C.1) in the form of a participant declaration that described the purpose of the experiment, and a brief overview of how the game works. It is expected that participants would interpret the rules of the game as conveyed by the game interface itself.

Each phase is run for a period of time in which the game is accessible to participants. The length of time is dictated by player activity, whereby a phase is complete once a player succeeds against the SAAF controller. Moreover, completion of a phase indicates that a player has been successful in performing an unknown attack that the controller was not able to detect.

At the end of each phase, the SAAF controller is updated to account for any unknown attacks that have been successful in beating the SAAF controller.

This exemplifies SAAF's ability to be extended in order to cope with previously unknown attacks, as well as promote additional challenges for participants within future phases.

Experiment Variables

Each phase is subject to a set of *independent*, *dependent*, and *control* variables. *Independent* variables are indicative of environment change, and driven by human participation. *Dependent* variables measure environment change, which refer to the consequence of human participation. For example, the performance of SAAF, violations detected, unknown attacks performed, the state of the access control, and game usage statistics.

Control variables denote the configuration of the authorisation infrastructure and the SAAF controller. These include the SAAF controller's perception of behaviour (behaviour policy), available solutions to the controller, the availability of probes and effectors, and configuration of the game environment. Control variables remain fixed throughout phases of the experiment. The only exception is that once a phase has ended and an unknown attack has been identified, detectors are updated in order to enable the SAAF controller to detect the unknown attack as a violation in the next phase. Lastly, each phase of the experiment builds upon its predecessor, meaning that the state of access at the end of phase 1 is the initial state of access for phase 2. This is important as it portrays the SAAF controller operating within a live and evolving environment.

Phase Progression

The experiments were conducted over a period of 7 months, as to obtain a wide range of data. Over the course of each experiment phase, violations (known attacks) were detected and mitigated, preventing malicious players from persisting with dishonest play.

A small number of unknown attacks were successful in enabling a player to beat the game in an unexpected way, resulting in the player obtaining what should have been an impossible score (e.g., completing the game in 0 or 1 turns). Post-phase analysis identified how the player was able beat the game without being detected or mitigated by the controller.

- The **control phase** was conducted over a period of 1 week where ten players were observed and asked to play a number of games in conformance to the rules of snakes and ladders. During this time, all players completed their games within a legal number of turns per game (i.e., greater than the least amount of turns possible when playing the game honestly). The SAAF controller was configured to detect violations only, albeit no adaptation would take place in order to identify the extent of unintended violations. As a result, a number of low level violations were detected due to genuine player mistakes.

- **Phase 1** was conducted over a period of 1 month. It resulted in a single player account successfully performing a code injection attack, in which the game's resource probe could not detect and notify the SAAF controller. The player had changed their starting square within the game from square 1 to square 63, allowing the player to finish in one turn after a single dice roll. The resource probe failed to report this activity due to limitations in what it could detect. As such, the trace log of the successful player's activity was analysed and used to create additional detectors within the resource probe.

- **Phase 2** was conducted over a period of 5 months. It resulted in a single player account successfully performing a code injection attack, which the SAAF controller failed to detect. The code injection attack was similar to that of the attack observed in *phase 1.* Despite the probe identifying and logging the attack, two faults were identified through post-phase analysis. This resulted in several false negatives, and ultimately had enabled the successful malicious player to avoid mitigation.

  The first fault identified that a mis-configuration between *phase 1* and *phase 2* had caused the controller to miss 13 potential violations, depicting similar attacks to the one that was successful in *phase 1.* To prevent this from occurring in the following phase, all known attacks were simulated within *phase 3's* production environment, prior to initiation, in order to confirm correct configuration.

  The second fault was associated to prolonged execution (in this case, 5 months), where the resource probe crashed as a result of a critical memory error. This fault was overcome using *microrebooting* [25], whereby sub-components of a system (e.g., the self-adaptive authorisation infrastructure) were scheduled to restart periodically.

- **Phase 3** was conducted over a period of 1 month. It resulted in two player accounts successfully performing a code injection that was not detected by the game resource probe or by the SAAF controller. Post-phase analysis identified that two player accounts beat the game in a single turn via identifying the endpoints of the AJAX routines that handled logging and policy enforcement. Instead of playing the game, the player accounts had sent falsified data directly to these endpoints to mimic player activity. In this case, it was identified that both accounts had triggered a single move with no associated dice roll. The resource probe contained no detectors to identify such a scenario, and as such, could not inform the SAAF controller of the behaviour. To prevent repetition of this type of attack, the resource probe required additional detectors to identify similar trace behaviour.

6.5.2 Observed Environment Change

The following section discusses two aspects of the observed environment change, namely game statistics and trends in player activity.

Game Statistics

Over the course of the experiment phases, 1455 games were played and 366 game accounts were created (Table 6.1). Out of these 366 game accounts, it was observed that account creations stemmed from 264 unique devices (based on a device's IP address). The number of devices provide some indication of the number participants. However, due to restrictions on the collection of personal information (Appendix C.1), a limitation in the game statistics is the inability to accurately identify activity of one individual, as individuals could have used multiple devices and accounts to game the system. As such, it was not possible to identify if participants had created accounts across multiple devices. In addition, this meant that only trends and patterns in the statistics of the game data could be observed.

	Control	P1	P2	P3	Total
Game accounts	20	62	195	89	366
Games played	269	168	692	326	1455
Unique devices	10	34	152	68	264
Unique games played	265	118	422	134	939
Unique turns	1482	329	1007	248	3066
Unique game actions	363	130	216	48	757

Table 6.1: High level statistics of game related data

Of particular importance, was the observation of diverse player interaction. In this instance, out of the 1455 games played, 939 games were unique. A unique game indicates a signature set of actions (i.e, a hash of the game) from start to end, where no other game is the same (excluding the concept of time, such as the time it took to perform an action). In addition, out of all of the games played, 3066 unique game turns were observed, where a unique turn is a signature of a player's turn (e.g., turn number, roll, and move).

When not considering a player's turn number within the game action's signature, there can only be 384 legal move signatures (i.e., number of dice rolls multiplied by the squares in which a player can land on). Despite this, 757 unique game actions were observed. This indicates that there were a number of illegal actions performed as a result of anomalous behaviour.

In addition to game data, a number of authentication and authorisation requests were observed (Table 6.2). The high number of failed authentications is largely due to a number of (unsuccessful) SQL injection attacks against the game's account login page. Whilst SQL injection attacks provide an additional perspective to the types of attacks targeted towards the game, these are viewed as external attacks and therefore out of scope of SAAF.

	Control	P1	P2	P3	Total
Authentication requests	34	175	880	616	1705
Granted authentication	31	104	395	177	707
Failed authentication	3	71	485	439	998
Authorisation requests	6174	2430	9446	3485	21535
Granted access	6174	2262	9109	3292	20837
Denied access	0	168	337	193	698

Table 6.2: Authentication and authorisation statistics

Regarding authorisation, 21,535 requests were observed, evidential of the extent of player activity. A number of these authorisation requests were denied, representative of the SAAF controller modifying subject access rights during game play. Whilst 20,837 requests were granted by the PERMIS authorisation service, the actual number of actions performed within the game are not one-to-one (see Appendix C, Table C.1 and Table C.2). For example, in phase 1 a number of rolls were performed within the game where no authorisation request had been made or was granted. This is evidence of users bypassing authorisation within the game resource.

Player Behaviour

Whilst not a prominent focus of the evaluation, a high level analysis of player behaviour is discussed in Appendix C.3. The analysis identified a number of trends that reflect the controller's perception of malicious behaviour. However, it also demonstrated the challenges in defining malicious behaviour.

For example, comparing player activity from the control phase and other phases demonstrated little correlation in terms of high level activity (e.g., time to perform actions or finish a game, number of actions per game, etc.). Only by observing particular contextual features of player behaviour (such as roll to move ratio) demonstrated clear differences to malicious and non-malicious activity. This emphasises the fact that observation of non-contextual activity (such as rate of access) is limited in detecting wider scopes of malicious behaviour.

6.5.3 Detection and Mitigation

Over the course of experiment phases 1 to 3, 1246 violations were detected (Table 6.3). Out of the violations detected, 1203 violations triggered a resultant mitigation, whereby a solution was enacted by the SAAF controller.

	P1	P2	P3	Total
Violations detected	228	738	280	1246
Violations mitigated	219	717	267	1203
Mitigation failures	9	21	13	43

Table 6.3: Violation statistics

The SAAF controller was shown to respond to 97% to the violations detected. However, 43 mitigation responses had failed for a variety of reasons. The majority of these failures were due to the SAAF controller identifying several violations in a single adaptation cycle. A limitation in the SAAF prototype is that it handles multiple violations in an sequential fashion, meaning that it mitigates the first violation before mitigating the next. Should SAAF have already performed a mitigative act (e.g., remove subject access) for the same subject, and attempted to do so for the next detected violation, it would throw an exception (as access had already been removed). This was the case for 20 of the failed mitigations. The remaining 23 violations occurred in phase 2, where the resource probe failed to report the malicious behaviour due to an error in its configuration.

There were also 50 violations caused by players within the control phase. These violations were representative of genuine player mistakes, and were associated with low severity violations. Moreover, they were typical of early games that control players had played. This identified that despite playing the game honestly, some control players would perform genuine mistakes, and their resultant behaviour would have triggered a mitigative response in a real deployment. The fact that these violations occurred suggests that some of the deployed behaviour rules should have been relaxed, or additional considerations made for essentially 'untrained' subjects.

Violations

Figure 6.4 conveys the percentage of violation types that were detected in phases 1 to 3. Violations `rollMove` and `illegalMove` represent the most common violations. These types of violations are considered to be the two easiest forms of attack, as they require little technical ability, and are opportunistic to the point that malicious subjects can perform the violation in every game turn.

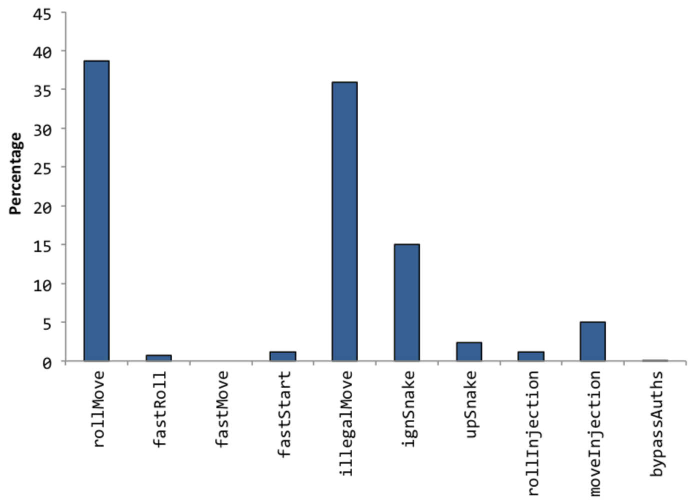


Figure 6.4: Percentage of detected violations by type

Violations `ignSnake` and `upSnake` are not so obvious, but still require little technical ability to perform. However, there was a greater percentage of `ignSnake` violations which is assumed to be because the violation was more obvious to commit (sharing similar characteristics to `illegalMove`). In addition, more sophisticated violations, such as `rollInjection` and `moveInjection` were seen to be rare. This indicated that fewer participants had the technical ability to perform complex attacks, such as code injection or session poisoning.

Lastly, a large proportion of detected violations stemmed from activity directly from the resource. This highlights the fact that observation of access logs alone from an authorisation service limits the detection of not only some of the more severe violations, but reduces the scope of behaviour that can be analysed.

Mitigations

In regards to mitigation, it was expected that the SAAF controller would identify and perform an appropriate adaptation in response to a malicious subject's current and past behaviour. For example, a subject who persistently performs low level violations over time would gradually lose their access, and may be warned about this prospect in the process. In contrast, a subject who performs a severe violation (e.g., code injection) would immediately lose their access.

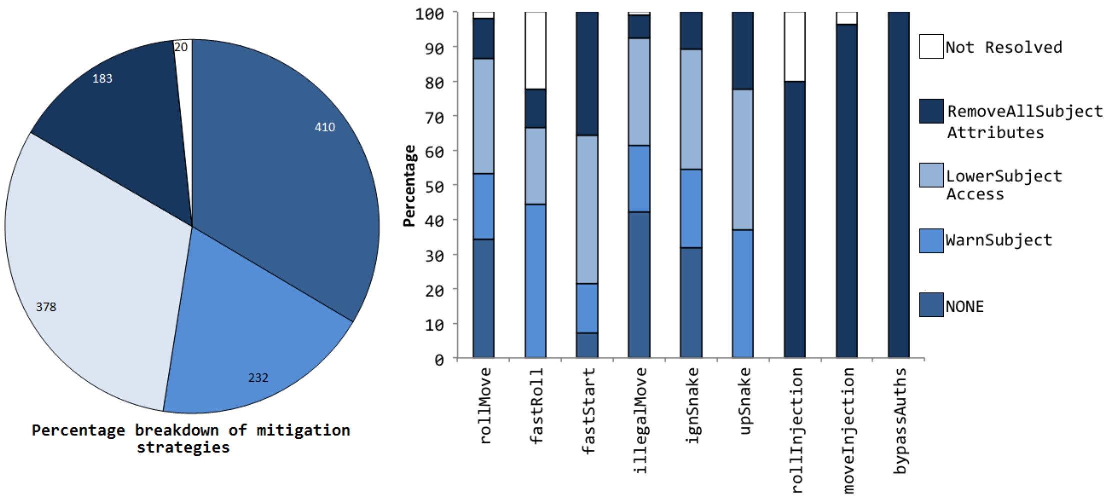


Figure 6.5: Breakdown of violations and mitigations

Figure 6.5 portrays a a complete percentage breakdown of mitigation strategies enacted, and a breakdown of the most common strategies enacted against a particular type of violation. The most common violations, such as `rollMove`, `illegalMove`, and `ignSnake` were typically responded with solution `S0`, where the decision to do nothing was chosen. This is due to the fact that many of these violations were a malicious subject's first time offence, and from the controller's perspective did not warrant adaptation. Mitigation of such violations were followed up with enactment of solution `S1`, where the decision to warn the subject was made, before lowering the malicious subject's level of access. Lastly, the majority of high severity violations, such as `rollInjection` and `moveInjection` were mitigated via an immediate removal of access, preventing malicious subjects from completing a game.

Controller Performance

The performance of the controller was observed in each phase, recording the time it took to decide and act on a detected violation. In addition, snapshots of the $ABAC_M$ access control model were also recorded, as to correlate size of the access control model with the performance of adaptation.

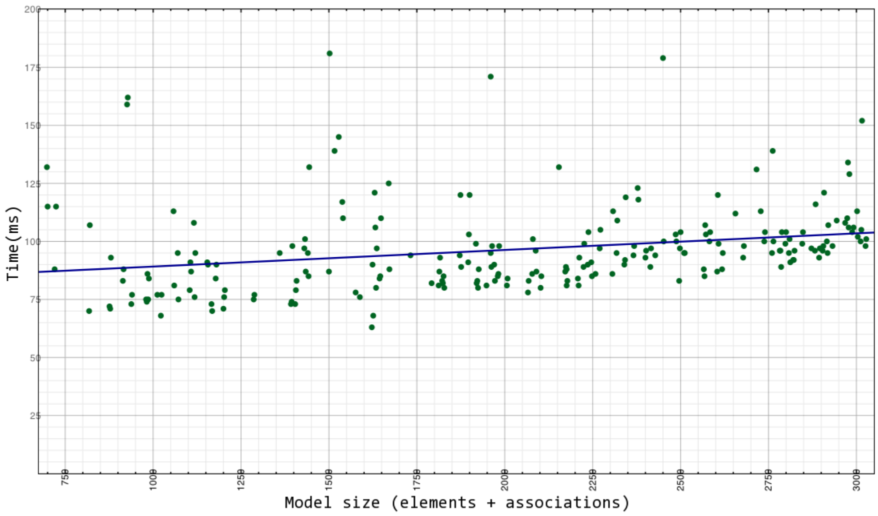


Figure 6.6: Total mitigation time versus model size

Figure 6.6 portrays a snapshot of performance time in enacting solution `S2` (lower subject access) against the size of the controller's $ABAC_M$. Outliers beyond 200ms were removed, which were representative of the problems caused by Java warmup, as discussed in Chapter 5's experiments. However, there were still fluctuations in response time, meaning that as the SAAF controller performed adaptations in quick succession, the performance became more efficient.

Observing the linear interpolation of the results, where the size of the model is 1000 (including all elements and associations), performance is shown to be 89ms. In regards to a model size of 3000, the linear interpolation shows performance at 103ms. As a result, it can be said that as the size the controller's $ABAC_M$ increased (due to new game accounts being created), the time to adapt increased at a linear rate.

6.6 Evaluation of Hypotheses

This section seeks to demonstrate the hypotheses proposed in Section 6.2, by analysing a set of attacks, and discussing dependent variables relevant to each hypothesis.

6.6.1 H1. Adaptation mitigates malicious subject activity

To demonstrate this hypothesis, the following exemplifies three different attack profiles that were observed throughout the experiment phases. They demonstrate how the SAAF controller mitigates malicious behaviour under differing circumstances, but also highlights some of the limitations that self-adaptive authorisation is faced with.

Mitigation of persistent weak violations

Single instances of low level violations (i.e., rollMove, illegalMove, ignSnake) alone do not necessarily warrant adaptation. This is a result of the SAAF controller tolerating a threshold of low level violations before adaptation. However, subjects who persist in committing such violations are faced with adaptation, as it is considered that repeat violations increase the confidence in malicious intent.

Such a profile of activity was typical of malicious subjects committing the simplest types of violations, and also the most prevalent form of activity observed and mitigated by the SAAF controller. Taking a sample of games with more than 5 violations (characterising a persistent attack profile), 229 games were recorded, whereby all players exhibited low level violations leading to the eventual loss of access.

Figure 6.7 captures a game trace that fits this profile, in the form of number of changes observed over time. It portrays the player's changes in terms of requests sent to the authorisation service and corresponding actions made in the game, as well as adaptation as a result of the SAAF controller. Here, the player is repeatedly committing the violation rollMove and ignSnake in order to beat the game. The time at which these violations occurred and the SAAF controller's corresponding mitigation are shown in Table 6.4.

Note that at certain points in this game, the player had clicked on the dice roll action rapidly. This triggered multiple requests to roll before the game had even responded to the initial request. As a result, there are more roll requests

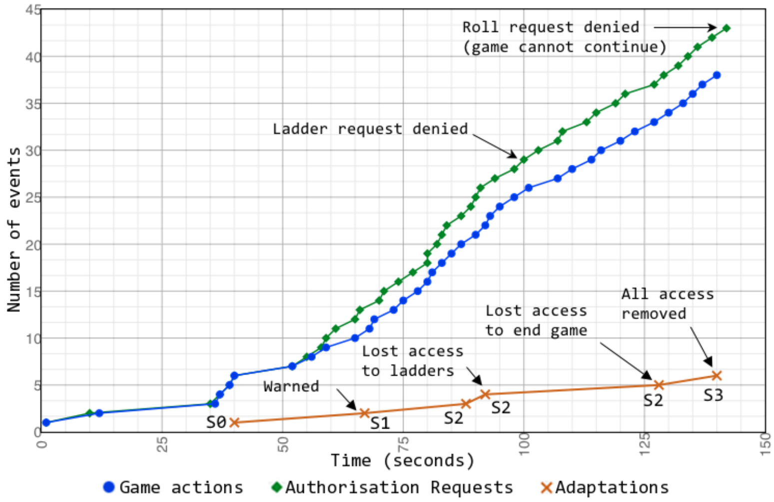


Figure 6.7: Trace of a persistent weak violation profile against mitigation

Step	Game Time (s)	Violation	Enacted Solution	Time (ms)
1	40	rollMove	noAdaptation (S0)	53
2	67	rollMove	warnSubject (S1)	51
3	88	ignSnake	lowerSubjectAccess (S2)	152
4	92	rollMove	lowerSubjectAccess (S2)	105
5	128	ignSnake	lowerSubjectAccess (S2)	100
6	140	rollMove	removeAllSubjectAttributes (S3)	216

Table 6.4: Adaptation trace of an persistent weak attack game

than actual rolls. A normal game would demonstrate a tight correlation between authorisation requests and corresponding game actions.

After each violation, the SAAF controller performs a mitigative decision. Initially the decision to do nothing (S0) is chosen, indicative of low level violations as a first offence. However, as the player persists in committing low level violations, the controller opts to first warn the player (S1), followed by repeatedly lowering the subject's access (S2).

In terms of evidence of mitigation, at 92 seconds into the game, the player lost access to use ladders, due to adaptation. When the player eventually landed at the base of a ladder and requested access to use the ladder, the player's access was denied. In addition, at 140 seconds into the game, after observing 6 low level violations, the SAAF controller removes all access from the subject (S3). As a result, the player's last action (a roll) is denied by the authorisation service, given that the player no longer has the necessary access rights to access the game.

Mitigation of immediate high severity violations

A more severe attack profile is one that contains single or multiple instances of sophisticated violations (i.e., as a consequence of code injection). In these instances, the SAAF controller must mitigate the subject immediately, as such violations are viewed upon as clear evidence of malicious intent.

Throughout the course of phases 1 to 3, 43 games exhibited an attack profile of a single sophisticated violation (whereby the game contained no other violation). This is said to be the profile of a determined attacker, one who is aiming to beat the game in a single turn or less, via the smallest set of changes.

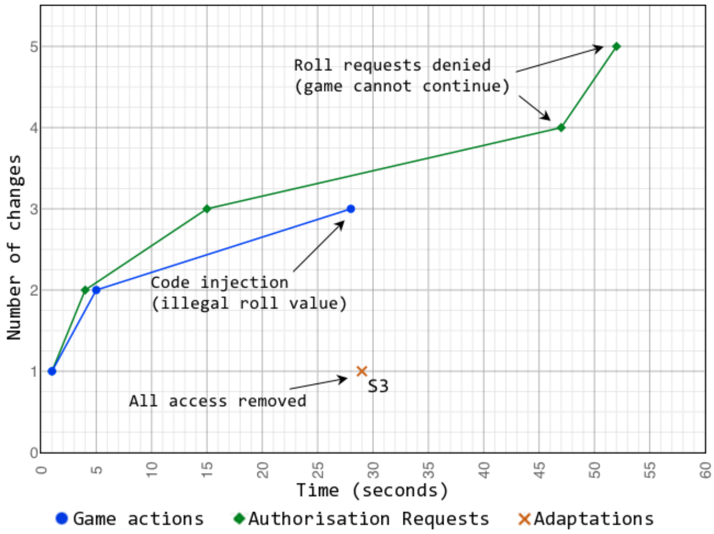


Figure 6.8: Trace of an immediate high severity violation against mitigation

Step	Game Time (s)	Violation	Enacted Solution	Time (ms)
1	29	rollInjection	removeAllSubjectAttributes (S3)	128

Table 6.5: Adaptation trace of an immediate strong attack game

Figure 6.8 and Table 6.5 portray the trace of a game that fits this attack profile. The player performed three authorised actions within the game, being a sequence of 'Start', 'Roll', and 'Roll'. In this instance, the second 'Roll' action was in actual fact a code injection attack, where the player had superficially increased the roll amount to 64 (beyond the legal range of 6). Consequently the illegal roll was identified by the resource probe, and pushed as a signature type violation to the SAAF controller. As a result, the SAAF controller removed all of the subject's access (S3), ensuring that future actions of the player are denied.

An interesting observation of this attack, is that the player who requested (and obtained) access to perform the roll, performed the action after a long delay (13s). This is unusual as a normal game exhibits a near immediate change in response to a granted authorisation request. This suggests the player was executing the game via a debugging tool, where the client code could be paused, updated, and executed, post-authorisation.

Mitigation of violations after game completion

A final attack profile is one considered where mitigation occurs, albeit the malicious player is successful in completing the game. In this instance, a player may be capable in performing a sophisticated attack at the point at which they request access to finish the game. As a result, the SAAF controller may detect and mitigate the player's access only after the game has been completed.

This type of attack profile would have enabled a player to beat the game in zero turns, the lowest possible score. However, during the course of phases 1 to 3, no such attack profile was observed, despite the successful unknown attacks that had triggered the end of each phase.

In order to highlight this behaviour, the trace log conveyed in Figure 6.9 and Table 6.6 has been purposely executed in a controlled environment. In this example trace, the attacker performs two actions, 'Start' and 'End'. Here, a code injection attack exploited a method call that prematurely ended the game before any rolls or moves were made. The resource probe was able to notify the SAAF controller of the behaviour, whereby the controller removed all of the player's access rights (S3), but only after authorisation to end the game had been given.

Step	Game Time (s)	Violation	Enacted Solution	Time (ms)
1	64	moveInjection	removeAllSubjectAttributes (S3)	180

Table 6.6: Adaptation trace of mitigation post game completion

In summary, the adaptation of subject access rights at runtime has shown to be successful in mitigating malicious subject activity. Three different attack profiles have been demonstrated, whereby the SAAF controller has mitigated attacks using a variety of solutions in an appropriate manner. However, in regards to mitigation of violations after game completion, there are some obvious limitations. Here it was possible to demonstrate an attack that could complete a malicious game, prior to detection and mitigation of the attack.

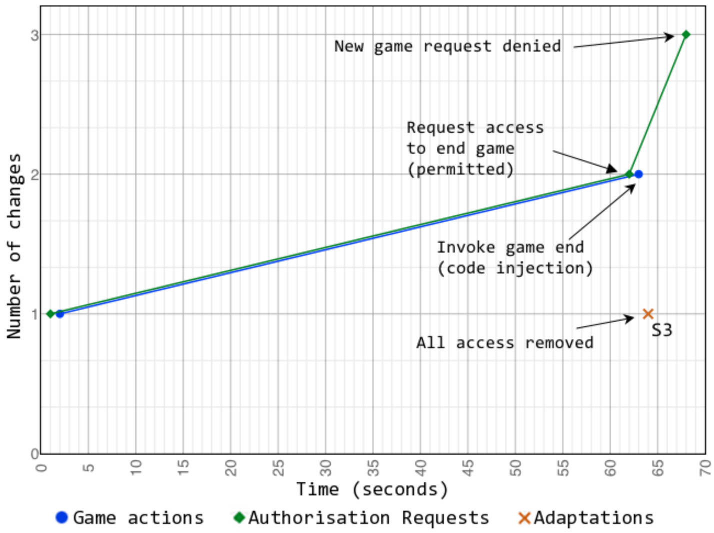


Figure 6.9: Trace of a mitigated malicious player post game completion

This type of outcome is primarily a consequence of the workflow-like nature of the game. However, it also highlights the fact that the SAAF controller can only mitigate a subject's behaviour post authorisation. Moreover, should a subject commit a violation once access has been granted, mitigation can only prevent future access, rather than terminate a subject's current resource session. To compensate for this, it would be necessary for SAAF to view the resource as part of the system as opposed to its environment. Here, a SAAF controller could exhibit control over a resource, and potentially terminate a subject's active session post authorisation. However, this opens up a number of challenges in regards to the integration and deployment of effectors within protected resources.

6.6.2 H2. Experience enables sophisticated attacks

A small percentage of violations detected were sophisticated attacks (Figure 6.4), such as code injection attacks. It was hypothesised that experience enables participants to perform sophisticated attacks within a game.

To demonstrate this, Figure 6.10 portrays statistics pertaining to malicious players prior to the point at which an injection attack was detected and mitigated. Here, the number of games played versus the number of violations detected are shown as evidence of player experience. Each data point represents at least 1 or more players (detected in performing an injection attack) that have played x amount of games, and committed y amount of violations.

Coupled with Table C.3 (Appendix C), only 11 participants (based on devices

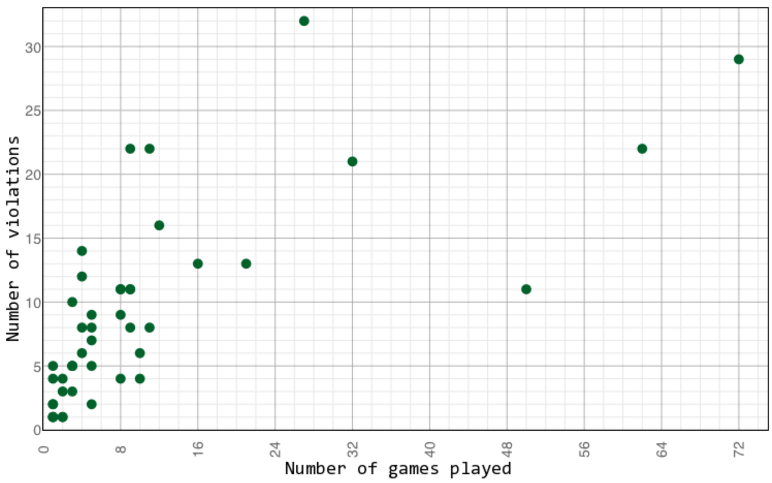


Figure 6.10: Violations and games played prior to injection attacks

used) were able to perform an injection attack within their first couple of games. Four of these were identified as committing an injection attack as their first violation, where these participants are assumed to have prior experience in ethical hacking. By contrast, the majority of participants had played at least 3 games with 3 violations or more, before committing an injection attack.

As a result, some players were quick to perform a sophisticated attack, whereas many players had gained experienced via games played, and were identified in performing a number of simple violations beforehand. In addition, it was observed that 30 of the participants (based on device) who had been identified had also created multiple accounts as a direct result of adaptation (prior to committing an injection attack). In a real deployment, these participants would have been prevented in performing such violations by mitigation in response to previously identified violations.

6.6.3 H3. Subject behaviour changes post adaptation

This hypothesis proposes that a player, aware of adaptation, will change their behaviour as to avoid detection or mitigation. To demonstrate this, taking a sample of players (identified by device), violations are analysed before and after the point at which it becomes known to the player that adaptation has taken place. It is considered that a player is aware of adaptation once they have either been warned of their behaviour (i.e., the controller enacts solution `S1`), or have been denied access to performing an action in a game.

Table 6.7 identifies four types of changes in behaviour observed after a player's

Behaviour changes	Participants
Did not repeat previous violation types, but performed new violation types	37
Repeated previous violation types, but performed no new violation types	26
Repeated previous violation types, and performed new violation types	114
Neither repeated or performed new violation types	0

Table 6.7: Changes in participant behaviour post-warning

first warning in regards to their behaviour. A total of 177 players were identified to have received a warning about their behaviour. It was identified that the majority of these players (64%) went on to continue repeating the same types of violations detected prior to warning, but also were detected as performing new types of violations post warning. However, 21% chose not to repeat previous types of violations, and instead solely performed new types of violations. Lastly, 17% simply chose to persist in repeating the same violations they had previously been warned about.

Behaviour changes	Participants
Did not repeat previous violation types, but performed new violation types	10
Repeated previous violation types, but performed no new violation types	46
Repeated previous violation types, and performed new violation types	37
Neither repeated or performed new violation types	46

Table 6.8: Changes in participant behaviour post-deny of access

Table 6.8 addresses the same four types of change, albeit demonstrating change after a player's first denial of access (e.g., a roll, move, ladder or bonus square has been denied). In this case, 139 players were identified as being aware of having their access denied at some point in their game history. In contrast to receiving a warning, 33% of players continued to persist in performing the same violations that had led to a denial of access, whereas only 6% of players chose not to repeat previous violations. A significant amount of players (33%) also chose to either stop playing the game, or halted their malicious behaviour.

Considering the two perspectives, it can be said that the majority of participants persisted with the same types of violations (i.e., behaviour) post knowledge of adaptation. However, there is evidence to suggest that a small proportion of players had factored in knowledge of adaptation, prior to performing future attacks (due to not repeating violations that lead to adaptation).

An additional viewpoint considers the potential for malicious players to use knowledge of the controller's operation to their advantage. Given that it is possible for a malicious player to finish a game with violations (due to SAAF's tolerance of low severity violations), a scenario exists that allows a malicious behaviour to exploit these circumstances. For example, a player who is aware that a number of rollMove violations would result in their access being removed, could tactfully choose when to perform the violation, and gain a greater advantage in the game.

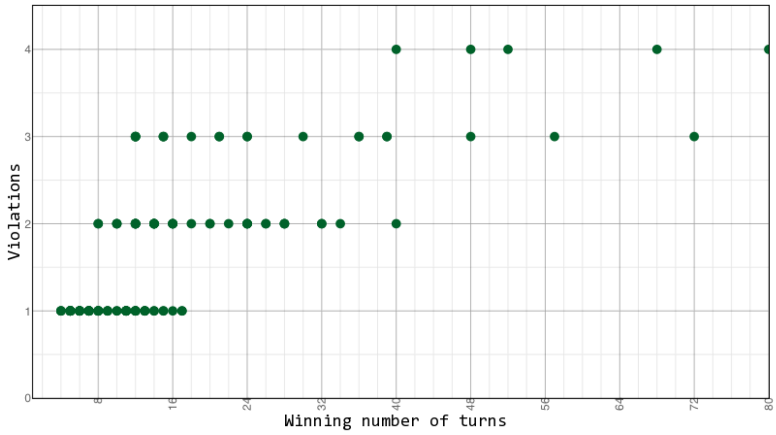


Figure 6.11: Number of detected violations versus number of turns to win a game

Figure 6.11 portrays the number of violations observed within malicious games, against the number of turns it took to win the game. Malicious games that took longer to win (i.e., with a high number of turns) contained a greater number of violations. Whereas, all of the lowest scoring games (i.e., 6 turns of less) had a single violation. In each of these games the player had sufficient experience in terms of number of games played, and previous committed violations. This provides some evidence to suggest that participants tactfully chose when to commit low level violations in order to gain a better advantage in the game, which was indicative of the number of low scoring games with low level violations.

6.6.4 Summary Conclusion on Hypotheses

Each hypothesis set out to evaluate an aspect of the success and limitations of self-adaptive authorisation in mitigating the abuse of access rights by real and unpredictable users.

Hypothesis 1 (adaptation mitigates malicious subject activity) identified that the SAAF controller was capable in mitigating various forms of malicious behaviour, where users adopted different strategies to beat the game. This was necessary to demonstrate the robustness of the SAAF controller in mitigating abuse by opportunistic low severity attackers, versus determined attackers, in regards to escalating appropriate solutions. Given the fact that the experiment has provided evidence that malicious subjects were no longer capable in gaining access, this hypothesis is seen as justified. However, one exception is that adaptation has only been shown to succeed when faced with known violations.

Hypothesis 2 (experience enables sophisticated attacks) analysed player violations over time. This provided insight into the prominence of high severity violations within the game. Whilst players were aware they could carry out malicious activity to beat the game, statistically, many players opted to attempt simple attacks first before carrying more complex attacks (e.g., code injection). In a real deployment, only a small percentage of players would have been capable of first performing a high severity violation, where many players would have already lost their access rights due to prior violations. This indicates that a SAAF deployment is well suited to handling numerous low level attacks, and as a consequence, is able to prevent malicious subjects from gaining enough experience to carry out more complex forms of attack.

Hypothesis 3 (subject behaviour changes post adaptation) evaluated the consequences of self-adaptation through observing change in participant behaviour post adaptation. An important aspect of this hypothesis was to address the deterministic nature of the SAAF controller, where users are capable of exploiting the operation of the SAAF controller given past experiences of adaptation. Whilst many players were statistically seen to persist with the same behaviour, despite adaptation, some players reacted to adaptation (by no longer performing a certain type of violation). Moreover, patterns identified suggested that games completed by players subject to prior adaptation, had tactfully performed violations to a point where they did not lose complete access.

It is worth noting that the evidence to demonstrate **hypotheses 2** and **hypotheses 3** is based on patterns observed within the statistics of the game. These statistics provided evidence that on the whole, more experienced players carried out complex violations, and that players changed their behaviour post adaptation (i.e., in terms of exploiting adaptation, or no longer persisting with a given type of violation). However, given the limitation that player activity was analysed by device address, it is not possible to accurately identify the participants performing violations. Therefore, evidence can only indicate the plausibility of these two hypotheses. To concretely justify these hypotheses, a further experiment is required where specific participant behaviour is assessed under controlled conditions.

6.7 Evaluating SAAF: Success and Limitations

This section discusses the success and limitations in the evaluations of SAAF. In addition, the consequences of SAAF in mitigation at runtime are discussed.

6.7.1 Success and Limitations of Performed Experiments

The Self-Adaptive Authorisation Framework (SAAF) has been evaluated using simulation techniques, as well as the execution of a live environment consisting of real users. This section addresses the success and limitations of each approach.

Preliminary analysis of SAAF in RBAC Authorisations Infrastructures, demonstrating Runtime Model Verification

Chapter 4 conveyed a preliminary experiment into the adaptation of subject access rights and authorisation policies, via simulation. A key focus of this experiment was to demonstrate the SAAF prototype controller managing a simple case of insider threat in a single organisation deployment. Primarily, the experiment showed adaptation in a scaled down RBAC authorisation infrastructure, in order to demonstrate the core concepts of SAAF:

- Runtime modelling of access control in a homogeneous manner, transferable to and from diverse implementations of access control, in order to integrate with legacy based authorisation infrastructures;
- The role of model verification in providing assurances prior to enacting adaptation strategies;
- The mitigation of malicious behaviour via the runtime adaptation, transformation, and deployment of subject access rights and authorisation policies.

This evaluation should be viewed as an introduction to what SAAF can achieve. It is built on simulating characteristics of a real historic case of insider attack. However, there are some limitations. Specifically, self-adaptive authorisation introduces several new concepts that traditional deployments of authorisation infrastructures do not consider. These include a definition of the perception of malicious behaviour, as captured by the controller's behaviour policy, and the perception of adaptation constraints.

Given the use of a historic insider attack, the perception of malicious behaviour was assumed to be that of characteristics that led to the organisation uncovering the attack. In reality the organisation's perception of behaviour may have been less tolerable. Additionally, whilst adaptation constraints are similar to a set of security requirements an organisation may use to verify their access control model, the use of such constraints at runtime is new. The organisation that fell victim to the attack may have defined security requirements at design time, where the requirements cease to be relevant for runtime. As a result, the constraints used were defined in such a way to exemplify what can be verified as part of the adaptation process.

Simulation of Insider Threat in Federated ABAC Authorisation Infrastructures

Chapter 5 furthered the preliminary experiment via the simulation of insider threat, albeit at a larger scale and within a federated ABAC authorisation infrastructure. In this instance, the chapter defined a fictitious case study inspired by historic cases of insider attacks. The case study was then presented as a formal set of environment changes within a federated authorisation infrastructure. These changes were then simulated to stimulate malicious behaviour within a runtime deployment of SAAF. Specifically, the evaluation demonstrated:

- A method in which to formally define and simulate insider attacks in the context of authorisation infrastructures;
- The mitigation of simulated attacks via adaptation of subject access rights and authorisation policies in a federated deployment;
- The escalation of adaptation in light of non-cooperating trusted business partners (i.e., trusted 3rd party identity providers).

The application of changeload [24] provided a formal means to define a trace of malicious behaviour within a target system. The advantages of repeatedly using the same trace of malicious behaviour during simulation has demonstrated the consistency of the SAAF controller's actions, and also what adaptation can be expected after a sequence of change. However, a limitation is that such an approach cannot evaluate how a controller may respond to diverse and unpredictable change. Therefore, as evident in this current chapter, it was necessary to place a deployment of SAAF in a live environment with real users.

Live deployment of SAAF in a real world ABAC Authorisation Infrastructure

In this chapter SAAF has been evaluated in handling the abuse of access within a live and unpredictable environment. The approach was necessary in order demonstrate the following:

- The ability to handle malicious behaviour in a live environment;
- To identify how malicious subjects may react to adaptation in a real deployment.

Gamification has enabled for a real-world like environment in which to capture unpredictable change. This has been critical to evaluating SAAF in as close to a real world deployment as possible. Gamification has many strengths, in particular it presents an environment that participants have familiarity with and can easily understand. In addition, it promotes competitive behaviour amongst participants, and as such the experiment was able to capture a diverse range of data. Lastly, gamification enabled the emulation of real-world resources, capturing concepts that exist in any typical application an organisation aims to protect. For instance, it was possible to emulate application work-flow in a resource (i.e., the game logic and rules), govern access to functions and processes (i.e., actions within the game), and enable participants to cause an impact on the resource (i.e., interaction with the game's leader board).

Of course there are several limitations that exist with this evaluation approach. Firstly, whilst gamification has been shown to be an effective means of generating data, symbolic of subject behaviour, it can only be seen as an emulation of a real-world deployment. A more realistic experiment would have been to deploy a virtual organisation with several resources, such as databases, and web servers, etc. However, given the fact that the experiment relied on human participation, it would have required participants to adhere to a specific role in the virtual organisation. This limits not only the scope of participation, but opens up several challenges in how to define the perception of malicious behaviour (whereas for a real organisation, this perception may already exist).

Additionally, it was decided not to enable model verification at runtime for this experiment, limiting the evaluation of model verification in a live environment. This was due to the fact that participants had no awareness of their position in the fictitious organisation (making adaptation constraints irrelevant). Second to this, was the costly performance times as a consequence of model verification, making it ineffective for the timespan of a game.

6.7.2 Consequences of Self-Adaptive Authorisation

Self-adaptation has been shown to successfully mitigate the abuse of access through the automated management of authorisation infrastructures. With this in mind, it is important to address the side-effects and emergent vulnerabilities.

Subject response to adaptation

The gamification approach provided an interesting perspective into how human subjects may respond to mitigation. It can be surmised that although adaptation had the desired effect in mitigating violations, as a consequence it may spur more severe violations, subject attempts at subterfuge, and exploitation of the controller itself. Notably, the gamification experiment identified that some subjects changed their behaviour post adaptation, and in some cases carried out more extreme forms of attack. This was compounded by SAAF's deterministic nature, demonstrating to some extent that subjects were able to exploit mitigation.

As a result, it is necessary to not only consider the need for adaptation, but also consider the timing of when to perform adaptation. Incorporating a notion of time as part of SAAF's planning mechanisms may enable more appropriate decisions in mitigating abuse. For example, decisions that consider the need to collect further evidence of abuse may well prevent a subject in performing future attacks or exploiting the operation of the controller.

Wrongful adaptation

The SAAF prototype controller has been demonstrated in detecting malicious behaviour using heuristic based rules. In all experiments these rules are assumed to be reflective of an organisation's legitimate perception of malicious behaviour.

Whilst this approach has been successful in demonstrating the closing of the self-adaptive loop, the detection approach is considered limited. The approach is limited as it is unable to identify unknown attacks, and has no ability to maintain a fluid perception of malicious behaviour. Moreover, the perception of malicious behaviour changes at runtime, creating potential scenarios where adaptation triggered by a given state may no longer be appropriate in a similar future state.

In addition, it was identified that during the control phase of the gamification experiments, participants made genuine mistakes that triggered violations. Therefore, considering additional context (e.g., if a subject is untrained) prior to adaptation may become a necessity.

Vulnerabilities of self-adaptive control

Self-adaptation introduces additional vulnerabilities that build on those within traditional systems. These vulnerabilities were initially identified in Chapter 3 (Figure 3.2), where attempts could be made to falsify subject behaviour, imitate the controller to gain indirect control of authorisation, or disrupt the controller's operation through denial of service attacks.

Whilst measures are put in place to protect against some of these attacks (such as through the use of mutual client authentication [7]), many of these areas of vulnerabilities present future research challenges that must be addressed. For example, had a participant within the gamification experiment uncovered the controller's service endpoints, an attacker could have performed a denial of service attack against the controller in order to disrupt observation of their activity, or disrupt adaptation from being performed. As such, a participant could have bypassed mitigation to successfully perform (and continue to perform) known violations to their advantage.

6.8 Summary

In summary, this chapter has demonstrated gamification as an approach for the evaluation of self-adaptive software systems at runtime. Gamification is a technique in which online games are deployed to solve complex problems and generate real meaningful data. It can enable the generation of diverse and unpredictable change representative of intelligent user behaviour, pre- and post-adaptation.

This chapter has demonstrated gamification as a viable approach for the evaluation of self-adaptive software systems. Using gamification, the Self-Adaptive Authorisation Framework (SAAF) was shown to be able to mitigate the abuse of access rights in a diverse and live environment. This was achieved through the deployment of an online game, protected by an authorisation infrastructure. A live experiment captured a wide range of unpredictable change generated through the online game, including malicious behaviour related to the exploitation of known vulnerabilities. This demonstrated SAAF's ability to handle malicious behaviour given the existence of real and intelligent users, in addition to capturing how users responded to adaptation.

Through the live experiment, this chapter has identified some key outcomes and future challenges applicable to self-adaptive authorisation. Notably, a small number of unknown attacks during the live experiment were successful. As a result, additional detectors had to be manually configured in order to detect future instances of the attacks. This is representative of the limitations in the SAAF prototype's current detection techniques, and enforces the need for future approaches to evolve at runtime once an unknown attack has been successful.

In addition, it was observed that malicious subjects may change their behaviour upon awareness of adaptation. In some cases, subjects began committing more sophisticated violations, or chose not to repeat previously detected types of violations. To compound this, there was evidence to suggest that subjects were tactfully choosing when to commit low level violations (to their advantage), as a result of understanding the deterministic nature of the SAAF controller. This poses a challenge that future approaches must consider, which is the fact that self-adaptation could lead to malicious subjects attempting to subvert detection, commit more damaging forms of attack, or exploit the very nature of self-adaptation to their gain.

Chapter 7

Conclusions

This thesis has presented a framework in which self-adaptation is used to enable the automated management of authorisation infrastructures. Through the instantiation of the framework, it has been established that authorisation infrastructures are able to observe, reason, and act at runtime on their configuration of access control. This was achieved through the use of an autonomic controller that is able to mitigate the abuse of access (i.e., insider attacks) through the runtime modelling and adaptation of access control policies and user privileges.

In addition, a prototype implementation of the framework has demonstrated that a self-adaptive authorisation infrastructure is able to detect, analyse, verify, and enact appropriate solutions to mitigate abuse. To this end, the prototype has been instantiated in two authorisation infrastructures, namely, a centralised RBAC [97] deployment, and a federated [93] ABAC [58] deployment.

The framework has been evaluated from a number of perspectives. This includes the simulation of insider attacks in centralised and federated authorisation infrastructures, as well as evaluating the framework in a live user experiment, while mitigating diverse and unpredictable attacks by human participants.

The rest of this section is structured as follows. Section 7.1 provides an overview of the thesis contributions. Section 7.2 provides a discussion of the approach, and its application. Section 7.3 identifies the limitations of the approach. Lastly, Section 7.4 proposes future directions of research.

7.1 Thesis Contributions

The goal of this thesis was to design, and evaluate a framework for the runtime management of access control. The following identifies the main contributions:

1. A framework that enables the automated management of authorisation infrastructures through self-adaptation, where abuse of access is mitigated through runtime adaptation of access control policies and user privileges.

2. An approach that has enabled, at runtime, the automated modelling of the configuration of access control within distributed and complex systems.

3. The automated management of federated identity providers, enabling the mitigation of abuse of access across multiple management domains.

4. The evaluation of self-adaptive software systems using gamification as an approach since it is necessary to trigger self-adaptation under diverse and unpredictable change.

7.2 Discussion

Through the design and evaluation of the Self-Adaptive Authorisation Framework (SAAF), it has been shown that existing access control methodologies and their implementing authorisation infrastructures can be made self-adaptable. By observing and analysing unexpected environmental changes at runtime, it is possible to detect and mitigate malicious user behaviour. Here, the use of autonomic controllers, probes, and effectors, has enabled the observation and control of authorisation infrastructures. Punitive measures are automatically enacted in light of malicious behaviour, whereby a malicious subject's right to access is limited or removed.

In addition, model driven engineering has been shown as an effective means to develop models that provide a homogeneous perception of the runtime configuration of access control. This has enabled autonomic controllers to obtain knowledge of their target system in order to make informed decisions when identifying and mitigating the abuse of access. It has also enabled the observation and adaptation of diverse implementations of a given access control model (e.g., RBAC). Here, autonomic controllers are able to transform between the modelled configuration of access into implementation specific formats of access control policies and user privileges.

To summarise, there are a number of advantages that warrant this framework as an alternative to existing approaches, in management of access control, and mitigation of abuse of access rights:

Mitigates at the point at which access rights are assigned and assessed, whereby abuse of access is mitigated in a persistent fashion via the adaptation of access control policies and user privileges, preventing further abuse of access from continuing in other systems and organisations.

Is reactive by preventing or limiting access once subjects are known to be malicious, as opposed to transient approaches that temporarily restrict access based on a perception of risk and usage, regardless if a subject is malicious or not.

Promotes separation of concerns from the decision of awarding access, to the decision of adapting the configuration of access. By doing so, the definition of access control rules is not subject to the complexity of classifying a perception of behaviour, which could result in errors and vulnerabilities.

Maintains and synchronises a model of the configuration of access causally connected to an authorisation infrastructure's runtime access control rules and assignment of access, in a homogeneous model. This enables adaptation to be relevant to the current state of access within an organisation, whilst considering the impact of varying adaptations over time, as new subjects and access rights are observed.

Provides assurances when mitigating the abuse of access through runtime model verification prior to the enactment of adaptation, safeguarding against inappropriate adaptations from taking place (i.e., adaptations that conflict with an organisation's security requirements).

Deployable in existing authorisation infrastructures thus enabling self-adaptation of any legacy based implementation of access control. This avoids the need to migrate to new authorisation infrastructures and access control models in order to gain the benefits of dynamic access control.

7.3 Limitations

As part of the evaluations of the proposed Self-Adaptive Authorisation Framework (SAAF), several limitations have already been identified. This section presents a summary of some of its limitations.

Using a self-adaptive approach, one of the clear limitations is the introduction of vulnerabilities [29] that a traditional system does not exhibit. Notably, the role of an autonomic controller in this work is to detect malicious behaviour and mitigate such behaviour through the control of authorisation infrastructures. Whilst mutual client authentication can be used to assure the inputs and outputs of the autonomic controller, these entry and exit points are vulnerable to attack. For instance, a denial of service attack on the controller could prevent or limit probe information from being received, therefore allowing abuse to go unnoticed.

A similar limitation concerns the need for probes and effectors. Whilst these components may already exist, their development will become essential in enabling observation and control [7]. If vulnerabilities exist in deployed probes and effectors, it may allow an attacker to gain control of the authorisation infrastructure.

Another limitation is the assumption that an authorisation infrastructure's protected resources operate as intended with respect to access control. The effectiveness of mitigation is dependent on how policy enforcement is achieved within protected resources. For example, if a resource policy enforcement point does not enforce a fine granularity of control (i.e., requesting access per operation the subject is to perform), the extent of control in mitigating abuse is restricted.

Another limitation is that SAAF is unable to reflect on the success of adaptation. SAAF utilises static analysis to decide upon appropriate solutions to malicious behaviour, whereas dynamic analysis would utilise feedback from the success or failure of solutions as part of future mitigation. As such, it can be said that SAAF may result in redundant attempts at adaptation. This was evident within Chapter 5's federated experiment, which demonstrated how SAAF handles the failure of an identity service effector.

Lastly, there are some limitations in the manner that SAAF detects and mitigates malicious behaviour. Whilst SAAF's reactive approach is viewed as a successful means to mitigating malicious behaviour, it cannot predict and thus protect before an attack has begun. In addition, SAAF's reactive approach strives to mitigate malicious behaviour in a timely manner, whereby given an attack, an appropriate solution is identified and enacted. However, mitigating attacks immediately may not be preferable, when it may be necessary to collect further evidence of abuse, before enacting a solution (as observed in Chapter 6).

The above has listed some limitations associated to SAAF. In the following, a number of limitations are discussed related specifically with the SAAF prototype that was built. One of these limitations is related to detection activities. Detection was based on a classic intrusion detection technique [120], whereby heuristics are used to define a fixed perception of malicious behaviour. For now, this technique served its purpose in providing a means to identify violations for a controller to act on. However, in the real-world, the perception of malicious behaviour is rarely fixed, requiring the controller to evolve its perception of malicious behaviour at runtime. In addition, only 'known' violations can be detected, meaning malicious behaviour that has not been identified within the controller's behaviour policy cannot be mitigated. This was evident by the success of some of the attacks observed within Chapter 6's gamification experiment.

Some limitations also exist within the prototype controller's solution selection activity. Whilst this was shown to work well in selecting appropriate solutions to a given case of malicious behaviour, solutions selected are by no means optimal. For instance, solutions are selected using a cost sensitive modelling approach in calculating the trade-offs between enacting a solution and allowing a malicious subject to continue their activity. As a consequence to this, only a small set of dimensions are considered in comparing such trade-offs. Additionally, the notion of cost could be considered artificial, and in some cases difficult to quantify, limiting its application to a diverse range of insider threat scenarios.

Lastly, it was also shown that the prototype's deterministic nature in mitigating malicious behaviour can be exploited. For instance, in Chapter 6's gamification experiment, evidence suggested that players with experience of SAAF's operation were able to gain an advantage in beating the game (dishonestly). This highlights limitations in SAAF's solution selection activity, as well as the simplified planning stage that was necessary to demonstrate closing of the feedback loop.

It is arguable that the risk of exploiting the prototype's deterministic behaviour is relatively low, given it is expected for the prototype to perform persistent adaptation (thus preventing a malicious subject from continuing to gain experience of the controller, and carry out attacks). However, it must not be ruled out that subject(s) may be working collaboratively, or have other access to other identities in which to attack a protected resource with. This is especially significant to federated environments, where a user can maintain a digital identity with multiple identity providers (e.g., Facebook, Twitter, and Google).

7.4 Future Work

Beyond addressing the identified limitations, the following presents future directions of research. Given that this thesis has presented the first steps towards self-adaptive authorisation, further work needs to be done. This entails improvements to detection, solution selection and verification, and the identification of vulnerabilities consequential of self-adaptive authorisation.

Detection of Malicious Behaviour

Mitigation of abuse of access rights has solely been demonstrated through the use of detecting 'known' patterns of malicious behaviour. To expand on this, it is important to adopt anomaly and learning based detection techniques [33] to identify unknown attacks. However, this presents several challenges:

- Dynamic identification and observation of behavioural parameters to observe and detect anomalies within a diverse and evolving environment;
- Evolve the perception of anomalous behaviour in order to maintain relevance and accuracy in detection;
- Classification of impact of anomalous behaviour to enable the selection of appropriate solutions.

Self-adaptive systems are particularly well placed to resolve some of these challenges. For example, a self-adaptive system can observe its environment to gain a clearer understanding of subject behaviour (e.g., monitoring context to legitimise detected anomalies). In addition, dynamic analysis [134] can be used to analyse feedback regarding the success of previous detections in mitigation, and used to strengthen the confidence in reoccurring anomalies warranting adaptation.

Solution Selection and Planning

The experiments identified that the deterministic nature of the Self-Adaptive Authorisation Framework (SAAF) could enable an attacker to exploit SAAF controllers. To deal with this, an element of randomness is required.

For example, planning could consider incorporating random delays [39] before enacting adaptation, as well as implementing a non-deterministic algorithm [52], where no one violation is mitigated in the same way. This may require a move to automating the generation of planning processes [40], whereby diverse alternative plans can be produced to mitigate violations. However, a challenge in regards to instilling non-deterministic self-adaptation concerns the provision of assurances.

A non-deterministic approach risks jeopardising assurance where adaptation may no longer be appropriate (e.g., a random delay prolongs ongoing abuse of access).

Lastly, future approaches in planning will require the use of utility based functions [36] that can consider a wider range of dimensions when selecting solutions. For example, solution selection should consider impact to individual subjects and cost of enactment, but also factor in criticality of attacked resources, a perceived level of threat to the organisation, and risk of failure in enacting adaptation.

Assurances over Adaptation

Verification of adapted models of access demonstrated assurances against inappropriate adaptation. The verification of access control models at runtime is a new problem created by SAAF, where it was shown to be costly in performance. A study of alternative approaches [59, 153] is required to identify a solution to these performance issues, where timely verification of access control models at runtime is essential. For example, impact change analysis [51] has the potential to improve upon performance, whereby only a relevant scope of an adapted model is verified.

Lastly, a flexible approach to verification should be applied, as opposed to the mandatory verification of all adaptation constraints. As such, it would be necessary to incorporate a dynamic approach where requirements can evolve at runtime to ensure relevant and appropriate adaptation [137].

Vulnerabilities as a result of Self-Adaptation

Another direction of research is to identify vulnerabilities that self-adaptive systems introduce. Whilst this thesis has described a high level attack model for SAAF in Chapter 3, clearly the threat from attacks on the self-adaptive system should be addressed further. This is compounded by the evidence of exploitation of the SAAF autonomic controller within Chapter 6's gamification experiment.

As such, there are two areas of future research to consider. The first is the prospect of vulnerabilities introduced as a result of adaptation (e.g., adaptation resulting in weaknesses in access control). The second is vulnerabilities that arise as a result of the existence of self-adaptation (e.g., exploitation of probes, effectors, and controllers). In regards to the former, this is closely related to providing assurances over adaptation. However, the latter is rarely responded to within the literature [150], and viewing self-adaptation as an active solution in mitigating malicious behaviour, should address such vulnerabilities.

In summary, this thesis concludes that self-adaptive authorisation has been shown to be a critical step in enabling access control in handling dynamic risk at runtime, given uncertainty in user behaviour. It has presented the first steps to handling such risk, through the runtime self-adaptation of authorisation infrastructures. As such, this thesis has demonstrated the feasibility of the approach through simulation and live user experiments. In addition, a number of future directions of research have been presented, whereby the emergent challenges of self-adaptive authorisation should be addressed.

Going forward, further details about the research and future publications can be found at **https://saaf-resource.kent.ac.uk**.

Appendix A

SAAF Prototype

A.1 Prototype Code Base

The prototype was developed using a mixture of technologies, predominantly Java 1.7, PHP 5.5, XML, SQL, and ATL. It comprises a standalone autonomic controller, an LDAP probe, a PERMIS probe, a generic resource probe, a SOAP based SimpleSAMLphp identity provider effector, an attribute certificate effector, LDAP client tools, and several model transformation programs. These components have all been used in evaluations of SAAF, through Chatper's 4 to 6.

Autonomic Controller

The autonomic controller was developed in Java 1.7 and is implemented across 11 packages.

- **saaf.com.default (Main):** This package essentially processed the initial configuration parameters of the controller, and initiated the controller. It provided a basic command line interface in order to process instructions at run-time, such as halt adaptation, or restart the authorisation service. These commands were predominantly used during testing.
- **saaf.com.permisLauncher:** Given the limitations of PERMIS, it was necessary for the controller to be capable of restarting the PERMIS authorisation service. This was in order to activate recently adapted policies (as PERMIS had no means to activate polices at runtime).
- **saaf.com.permisClient:** The permisClient is an in-built probe used to detect policy changes and access events within a PERMIS authorisation

service. It operated via the host operating system in order to detect system and environment change exhibited by PERMIS. The package is executed in terms of a 'pull' mode, where it constantly seeks to identify change.

- **saaf.com.ldapClient:** The ldapClient is an in-built probe and effector used to directly observe changes in an LDAP directory as well as carry out changes in an LDAP directory. The package is operated both in a 'push' (to make changes to an LDAP directory) and 'pull' (to identify changegs in an LDAP directory) mode.

- **saaf.com.soapService:** This package implements the prototype's SOAP web service. Upon initiation, the controller would deploy its SOAP service and listen on a specific port so that it may receive data from external probes. It implements a set of web service functions in order to receive different types of changes within an authorisation infrastructure.

- **saaf.com.abac:** The abac package provides an implementation of the ABAC metamodel, defined using Eclipse EMF. The package maintained an ABAC model manager which facilitated the creation, interaction, and identification of change with an ABAC model. This provided the prototype the ability to generate and maintain a modelled state of access.

- **saaf.com.behaviourModel:** The behaviour model provides an implementation of the prototype's behaviour metamodel. It implements a model manager in which to capture behaviour, violations and statistics about subjects derived from gauges. This is a bespoke package developed outside of Eclipse EMF, but makes references to the types maintained in the prototype's ABAC model.

- **saaf.com.monitor:** This package provides an implementation of the prototype's monitoring stage. Specifically, its goal is to process detected changes, either directly from the prototype's internal LDAP and PERMIS client, or via the prototype's SOAP web service. The monitor aims to generate and populate gauges, as well as parsing triggers in order to guide its processing of detected events. In addition, the monitor executes model transformation programs in order to build implementation specific models, and transform them into the relevant ABAC model.

- **saaf.com.decisionEngine:** The decision engine package provides an implementation of both the Analyser and Planner components. Given the fact that the planner component is a relatively simple implementation, it was decided to combine this with the analyser package. The analyser performs a cost sensitive calculation to determine the impact of detected violations, as well as interacts with the behaviour model to maintain a view of subject violations. The planning aspect implements a second cost sensitive calculation in which to identify an appropriate solution. Once completed, it builds an executable plan (in terms of a Java Object) that can be executed by the executor package.

- **saaf.com.executor:** The executor package implements a simple executor which acts as an interface to effectors. It maintains the ability to execute a plan object by transforming attributes of a plan into actions, such as the generation and sending of XML SOAP messages. It maintains the ability to identify the failure of a plan to execute, and can request additional plans from the decisionEngine.

- **saaf.com.utils:** The utils package provides a set of classes to enable common functions within the prototype, such as initiating HTTPS connections, logging mechanisms, and configuration parsers.

In total there are 58 classes across all packages, 654 methods, and 6581 lines of code (not including model transformation programs, probes, effectors, and 3rd party packages). In addition, besides the standard Java packages included in Java 1.7, the controller prototype made use of the following 3rd party Java packages:

- bcprov-jdk16-146 (BouncyCastle: for X.509 certificate generation)
- commons-codec-1.4 (Http Client)
- jnotify-0.94 (Integration with host file system)
- org.eclipse.core (Eclipse modelling framework)
- org.eclipse.emf (Eclipse modelling framework)
- org.eclipse.m2m.atl (ATL transformation language)
- org.eclipse.osgi (Eclipse implementation to generate dynamic component models in Java)

- org.eclipse.uml2 (Eclipse implementation of UML2)
- xom-1.0d8 (XOM XML implementation for Java)

Lastly, a standalone executable of *rbacDSML* was used in conjunction to the prototype's controller solution verification process. This was initiated via the *decisionEngine* package.

Probes and Effectors

Besides the use of the host operating system, LDAP client tools, and internal probes/effectors to the controller, the prototype controller made use of the following external probes and effectors. In total these probes and effectors comprise of 1758 lines of code, developed across Java, and PHP.

1. **LDAP Probe**: The LDAP probe was designed to monitor changes to an LDAP directory via direct observation of LDAP logs. It could be configured to detect specific types of changes, and communicate these changes by pushing change notifications to the prototype controller. This made use of the prototype's web service interface and used as part of the evaluation in Chapter 5. The probe was developed in Java.

2. **Generic Resource Probe**: The Generic Resource Probe could be configured to detect certain types of change events within a resource's log files. The probe is relatively primitive but was capable of instructing the prototype of changes detected in Chapter 6's gamification experiment. The probe was developed in Java and relied on SQL triggers to capture relevant change.

3. **SimpleSAMLphp Effector**: The SimpleSAMLphp effector enabled the adaptation of a federated identity provider's LDAP directory. It was necessary to enable a federated identity provider to control the extent of adaptation from remote controllers. It offered two forms of functionality. One was an automated means to prevent / accept adaptation from controllers. The other was a semi-automated means that allowed the identity provider to manually approve adaptation via the use of a human operator. Adaptation requests could be made via mutually authenticated SOAP messages sent over HTTPS. The effector was developed in PHP and sqlLite, as well as extending modules that already exist within SimpleSAMLphp.

4. **AttributeCertificate Effector** (Effector): Enabled the generation, signing, and deployment of X.509 certificates via an LDAP directory, where a subject's signed access rights could be modified.

Models and Model Transformation Programs

Three Eclipse Modelling Framework metamodels were implemented to enable transformation between the implementation specific (of LDAP and PERMIS), to a generic ABAC model.

- **saaf.emf.abac.metamodel** (Defines what can exist in an ABAC model)
- **saaf.emf.permis.metamodel** (Defines what can exist in a PERMIS policy model)
- **saaf.emf.ldapIdentity.metamodel** (Defines what can exist in terms of an LDAP identity within an LDAP directory)

Alongside these metamodels were automatically generated (with some modification) model management packages (to generate, interrogate, and detect change of instantiated models). These packages were generated by the in-built functionality of Eclipse EMF.

Four model transformation programs were created. A transformation program comprises an executable Java package generated by the ATL plugin creator. Each executable contained a purpose built ATL mapping that describes the mapping from a source metamodel to a target metamodel. In total the transformation programs comprised of 1289 lines of code (not including automatically generated code via the ATL plugin creator).

- **saaf.emf.permis_ldap2ABAC** (Transforms a PERMIS policy model and LDAP model into an ABAC model)
- **saaf.emf.abac2PERMIS** (Transforms and generates a PERMIS policy from an ABAC model)
- **saaf.emf.abac2LDAP** (Transforms an ABAC model into a set of LDAP identities and attributes)
- **saaf.emf.abac2RBACDSML** (Transforms an ABAC model into an RBACDSML model, assuming the ABAC model conforms to RBAC)

A.2 LDAP Metamodel

Figure A.1 depicts the metamodel used to describe the relevant contents of an LDAP directory to modelling subject access rights. It describes an LDAP directory that contains a set of subject entries, a set of attributes, and subject attribute assignments. The enumeration IdentifierType describes a set of LDAP attributes that are used to uniquely identify a subject entry, such as a distinguished name, a unique id (uid), a common name, or eMail address.

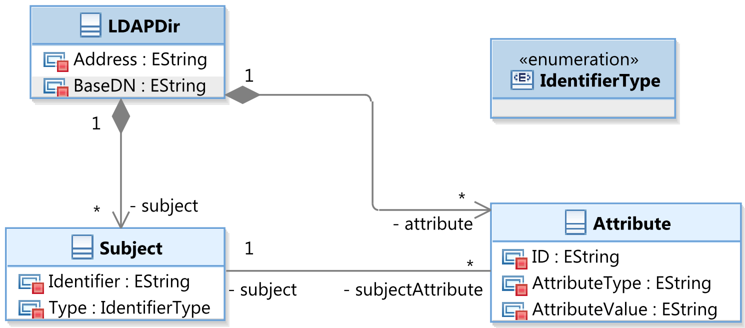


Figure A.1: LDAPIdentity metamodel defined in Ecore

A.3 PERMIS Metamodel

Figure A.2 presents a simplified version of PERMIS's metamodel, described in Ecore. The metamodel is in fact much more expansive, and is used to describe the structure of a PERMIS policy, along with what must exist.

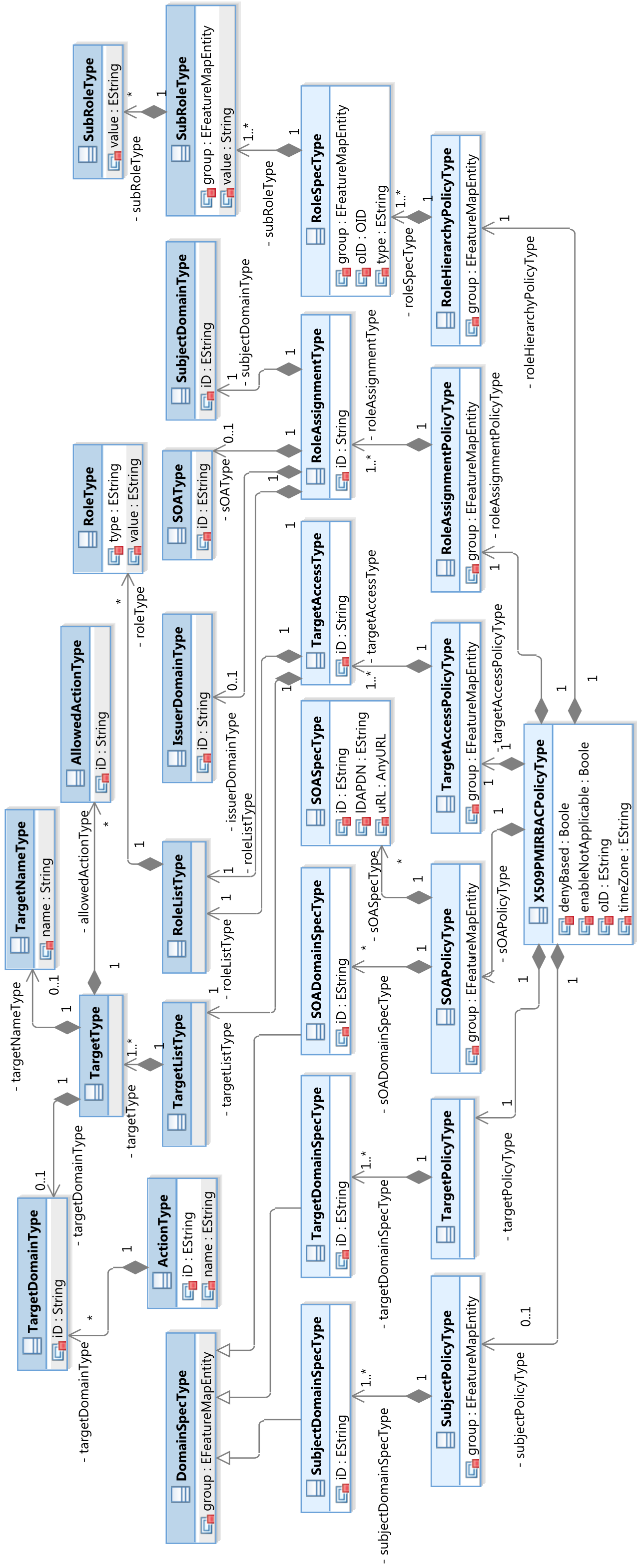


Figure A.2: Simplified PERMIS Metamodel defined in Ecore

A.4 Scope of Observation

The following describes message formats exemplified in XML for each of the detectors applicable to the SAAF controller prototype. They define what is expected by the prototype controller, to detect and understand change in the target system, and its environment.

Access change: Describes a change in an authorisation service, stating a subject's access request and decision.

```
<accessChange>
  <subjectID>cn=bob,ou=saafOrgUnit,o=saafTarget,c=gb</subjectID>
  <issuer>http://issuer.uri</issuer>
  <attributes>
    <attribute>
      <attributeType>role</attributeType>
      <attributeValue>staff</attributeValue>
    </attribute>
  </attributes>
  <resource>http://printer.resource.uri</resource>
  <action>print</action>
  <decision>grant</decision>
  <timestamp>1406800322123</timestamp>
</accessChange>
```

Listing A.1: Access change event (Example)

Resource change: Describes a change in a resource, stating a given subject, attribute type, value, and operation.

```
<resourceChange>
  <subjectID>cn=bob,ou=saafOrgUnit,o=saafTarget,c=gb</subjectID>
  <resource>http://printer.resource.uri</resource>
  <attributeType>printCount</attributeType>
  <attributeValue>1</attributeType>
  <operation>add</operation>
  <timestamp>1406800323123</timestamp>
</resourceChange>
```

Listing A.2: Resource change event (Example)

Policy change: Notifies when a policy has changed and the new contents of the policy. Note that currently this detector can only handle policies from an authorisation service.

```
<policyChange>
  <authService>https://PERMIS.uri</authService>
  <location>../PERMIS/policy.xml</location>
  <policy>
    ... //permis policy document (in XML)
```

```
  </policy>
  <timestamp>1406800343123</timestamp>
</policyChange>
```

Listing A.3: Policy change event (Example)

Subject change: Notifies when a subject's access rights have changed. Note in this case it is implied the controller has access to the identity service.

```
<subjectChange>
  <subjectID>cn=bob,ou=saafOrgUnit,o=saafTarget,c=gb</subjectID>
  <attributeType>role</attributeType> //type of attribute changed
  <attributeValue>staff</attributeValue>
  <operation>add</operation> //implies role=staff has been added
  <timestamp>1406800353123</timestamp>
</subjectChange>
```

Listing A.4: Subject change event (Example)

A.5 Scope of Control

The following describes the format of operator change types, relevant to effectors of a target system. Each change type refers to a change on the target system, containing the necessary information for an effector to act upon.

LDAPClient tools: Utilises common APIs in Java for modifying an LDAP directory. This has been included given the popularity of the LDAP protocol in storing subject access rights.

```
<ldapModify>
  <ldap>ldap://ldap.uri</ldap>
  <subjectID>cn=bob,ou=saafOrgUnit,o=saafTarget,c=gb</subjectID>
  <attributeType>role</attributeType>
  <attributeValue>staff</attributeValue>
  <operation>remove</operation>
</ldapModify>
```

Listing A.5: LDAPModify operation

SOAPSender (change access right): Sends a SOAP message to a given identity service effector requesting the adaptation of a subject's access rights.

```
<requestAdaptation>
  <serviceProvider>http://sp.uri</serviceProvider> //for federated use
  <subjectID>cn=bob,ou=saafOrgUnit,o=saafTarget,c=gb</subjectID>
  <attributeType>role</attributeType>
  <attributeValue>staff</attributeValue>
  <operation>remove</operation>
```

```
  <reason>abused sp.uri.Printer</reason>  //for federated use
</requestAdaptation>
```

Listing A.6: Subject Adaptation Request (SOAP)

SOAPSender (activate policy): Sends a SOAP message to a given authorisation service effector, requesting change in policy.

```
<requestAdaptation>
  <serviceProvider>http://sp.uri</serviceProvider> //for federated use
  <location>../PERMIS/policy.xml</location>
  <policy>
    ... //permis policy document (in XML)
  </policy>
  <reason>multiple attacks on sp.Printer</reason> //for federated use
</requestAdaptation>
```

Listing A.7: Policy Adaptation Request (SOAP)

A.6 Behaviour Policy

Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<behaviourPolicy id="">
  <baseRule id="">
    <abacRelationship>
      <subject></subject>
      <issuer></issuer>
      <attributeType></attributeType>
      <attributeValue></attributeValue>
      <resource></resource>
      <action></action>
    </abacRelationship>
    <conditions>
      <condition type=""></condition>
    </conditions>
    <cost/>
  </baseRule>
  <compRule id="">
    <baseRules>
      <baseRuleID></baseRuleID>
    </baseRules>
    <conditions>
      <condition type=""></condition>
    </conditions>
  </compRule>
</behaviourPolicy>
```

Listing A.8: Behaviour Policy Specification

Behaviour Rule Types

Detecting malicious behaviour follows an event-condition-action approach. System and environment changes (events) are observed, whereby if they meet the conditions of a behaviour rule, behaviour analysis is triggered. When conditions of a behaviour rule are met, a violation is identified. Given the limitations in the detection approach, instantiations of these rules should express the extremes of anomalous activity.

Implemented within the prototype are five types of conditions, which are used to demonstrate the detection of malicious behaviour. Each condition type is conveyed in Table A.1:

Type	Description
Rate of Change	Describes the rate of change over an interval of time
Required Transaction	Defines a required sequence of change (e.g, for a workflow)
Bad Transaction	Defines a malicious sequence of change (e.g., repeating a set of requests in a given sequence)
Deviation of Change	Defines a threshold distance in comparison to past behaviour by the defined ABAC relation
Signature of Change	A single change that matches a signature (e.g., a blacklisted IP address)

Table A.1: Prototype behaviour rule types

Behaviour Policy Example

```
<?xml version="1.0" encoding="UTF-8"?>
<behaviourPolicy id="saaf1">
<!--
   Violation if any subject accesses employeeDB.read at a greater rate of 30
   requests per minute, where requests observed between 8am and 6pm
-->
  <baseRule id="empDB_fast_read_1_min">
    <abacRelationship>
      <subject>*</subject>
      <resource>employeeDB</resource>
      <action>read</action>
    </abacRelationship>
    <conditions>
      <condition type="rate">
        <threshold>30</threshold>
        <interval>1</interval>
        <timeScale>min</timeScale>
      </condition>
```

```
      <condition type="signature">
        <timeAfter>08:00</timeAfter>
        <timeBefore>18:00</timeBefore>
        <source>*</source>
      </condition>
    </conditions>
    <cost>100</cost>
  </baseRule>
<!--
    Violation if any subject from the contractor issuer matches the transaction
    sequence of 'read+modify' of the employeeDB resource, more than 20 times in
    a 10 minute interval.
-->
  <baseRule id="empDB_read_modify_10_min">
    <abacRelationship>
      <subject>*</subject>
      <issuer>contractorOrg</issuer>
    </abacRelationship>
    <conditions>
      <condition type="badTransaction">
        <change>
          <resource>employeeDB</resource>
          <action>read</action>
        </change>
        <change>
          <resource>employeeDB</resource>
          <action>modify</action>
        </change>
      </condition>
      <condition type="rate">
        <threshold>20</threshold>
        <interval>10</interval>
        <timeScale>min</timeScale>
      </condition>
    </conditions>
    <cost>200</cost>
  </baseRule>
<!--
    Violation if a given subject has broken both base behaviour rules
-->
  <compRule id="fast_read_transaction">
    <baseRules>
      <baseRuleID>empDB_fast_read_1_min</baseRuleID>
      <baseRuleID>empDB_read_modify_10_min</baseRuleID>
    </baseRules>
    <conditions>
      <condition type="signature"/>
    </conditions>
    <cost>1000</cost>
  </compRule>
</behaviourPolicy>
```

Listing A.9: Excerpt of a behaviour policy

A.7 Solution Policy

Specification

Listing A.10 describes a general specification of a solution within the SAAF prototype's solution policy.

```
<solutionPolicy id="">
  <solution id="">
    <minImpact></minImpact>
    <action>
      <operation></operation>
      <cost></cost>
    </action>
     ...
    <bRuleID></bRuleID>
  </solution>
   ...
</solutionPolicy>
```

Listing A.10: Solution policy specification

Solution Actions

Actions represent a change that can be performed against the target system (authorisation infrastructure).

There are three types of actions: 1. *Msg* type, where technically no adaptation is made but an action generates a message to a given target, 2. *Sub* type, where adaptations are made against a subject (within an identity service), and 3. *policy* type, where adaptations make changes at a policy level (within an authorisation service). Table A.2 identifies the complete set of actions that can be utilised (in a non-mutually exclusive manner) within solutions.

Solution Policy Example

The following listing describes a basic solution policy. It contains a single solution made up of two parameterised actions. The solution is applicable to an identified violation providing the subject that caused the violation exhibits an impact of 0.3 or greater (as calculated through *behaviour analysis*). Each parameterised action has a weighting of cost, detailing the cost to the deploying organisation should the action be executed. This cost is used as part of the prototype's solution selection process. Finally, a behaviour rule id is used to specify which particular behaviour rules this solution can be applied to. In this example, the solution can be used in any violation of a behaviour rule.

Action	Type	Description
WarnSub	Msg	Warns subject of their behaviour
WarnIssuer	Msg	Warns an issuer of a subject's behaviour
WarnSOA	Msg	Warns resource owner of behaviour
LowerSubAccess	Sub	Lowers a subject access by one in an attribute hierarchy
RmvSubAttribute	Sub	Removes an enabling attribute from a subject
RmvAllSubAttribute	Sub	Removes all enabling attributes from a subject
RmvSubAccessToRsrc	Sub	Removes a subject's access to a resource
RmvIdPValidAttr	Policy	Remove trusted attribute from issuer
RmvAllIdPValidAttr	Policy	Remove all trusted attributes from issuer
RmvAttrPerm	Policy	Remove a attribute permission assignment
RmvAllAttrPerms	Policy	Remove all of an attribute's permission assignments
RmvAccessToRsrc	Policy	Remove all access of all subjects to a resource
DeactivatePolicy	Policy	Shut down all access to all resources

Table A.2: Prototype actions for ABAC authorisation infrastructures

```
<solutionPolicy id="saaf1'>
  <solution="s1">
    <minImpact>0.3</minImpact>
    <action>
      <operation>lowerSubjectAccess</operation>
      <cost>30</cost>
    </action>
    <action>
      <operation>warnIssuer</operation>
      <cost>0</cost>
    </action>
    <bRuleID>*</bRuleID>
  </solution>
</solutionPolicy>
```

Listing A.11: Excerpt of a solution policy

A.8 Preliminary Analysis Results

This section briefly describes the escalation of decisions conveyed within Chapter 4's preliminary analysis. These are included within the appendix in order to provide evidence of the naive implementations of *behaviour analysis* and *solution selection*, however, are not viewed as a critical part of the thesis's contribution.

The simulation of the insider attack contained four attack stages, where the escalation of abuse against the case study's *ElectronicLibrary* can be observed. Two behaviour rules were defined. A base behaviour rule *bt1*, which was assigned a cost value of 50 if violated. And a composite behaviour rule *ct1*, which was assigned a cost value of 150 if violated. *bt1* expressed a condition which would trigger a violation if a subject accessed the *ElectronicLibrary* greater than 48 times within a four hour interval. *ct1* expressed a condition which would trigger a violation if the total number of detected *bt1* violations was greater than 2. Upon each violation detected, a subject's impact to the organisation was calculated.

Table A.3 describes the four stages of the experiment in relation to the impact of each subject in causing violations. Subject impact (S_{Impact}) was calculated in terms of normalising the total cost of violations ($\sum v_w$) multiplied against the total historical violations (V_{count}) that the subject had caused, against a maximum and minimum cost set by the organisation (500 and 0 respectively).

Stage	User	Violations	$\sum v_w$	V_{count}	S_{Impact}
1	Anne	bt1	50	1	0.1
2	John	bt1	50	1	0.1
3	Mary	bt1, ct1	200	2	0.8
4	Bob	bt1, ct1	200	2	0.8

Table A.3: Calculated subject impact per stage of the simulated case study

Given the impact of each subject at each stage, a set of solutions were identified. A solution was deemed applicable if the subject's impact level was greater than the minimum required impact as defined per each solution. To mitigate detected violations, five solutions (Table B.2) were deployed. Each solution contained an aggregate cost ($\sum a_{cost}$) in terms of executable actions against the target system, and a minimum impact (Min_{Impact}). If the subject impact was greater than a given solution's minimum impact, the solution was applicable to mitigating the violation.

Table A.5 describes each stage's applicable solutions, and how each solution's

Solution	Description	$\sum a_{cost}$	Min_{Impact}
S1	Remove all roles from subject	50	0.1
S2	Remove permission from role	200	0.1
S3	Remove all acces to a resource	1000	0.7
S4	Remove all permissions from role	500	0.7
S5	Disable all access	5000	0.9

Table A.4: Solutions with cost and minimum impact

cost was calculated and ranked. Only solution S1, S2, S4 are explored, as solution verification identified that S3 and S5 invalidated the controller's adaptation constraints.

For each solution the number of malicious S_{Mal} and non-malicious S_{NonMal} subjects impacted were calculated. This depends on the current state of SAAF's behaviour model and $ABAC_M$ at time of adaptation, and what adaptation is to be performed. The number of non-malicious subjects reflect the number of malicious subjects that lose access as a result of a given solution. In regards to the number of malicious subjects, this is configurable within SAAF. A malicious subject is identified as either a subject who has committed a malicious act within the past, or is a subject who has performed a violation, but is yet to be mitigated (e.g., due to failed adaptations). For this preliminary analysis, the number of malicious subjects was calculated as the number of subjects identified in causing a violation within the last 30 days.

User	Impact	sl	$\sum a_{cost}$	sl_{impact}	$sl_{goodness}$	S_{Mal}	S_{NonMal}	sl_{cost}	Rank
Anne	0.1	S1	50	0	50	1	0	0	1
John	0.1	S1	50	0	50	1	0	0	1
Mary	0.8	S1	50	0	50	1	0	0	1
		S2	200	200	300	3	4	100	N/A
		S4	500	200	300	3	4	400	N/A
Bob	0.8	S2	200	150	500	4	3	-150	1
		S1	50	0	50	1	0	0	2
		S4	500	150	500	4	3	150	N/A

Table A.5: Solution selection and ranking

The $\sum a_{cost}$ indicates the aggregate cost of the solution sl in terms of its executable actions. The sl_{impact} indicates the impact of the solution to non-malicious subjects S_{NonMal}, identifying the cost implied in removing all access to non-malicious subjects. This is essentially the minimum cost in removing a subject's access (synonymous to S1, i.e., 50) multiplied by S_{NonMal}. The $sl_{goodness}$ indicates the aggregate cost of violations against the abused resource, identified for all subjects in S_{Mal}. The sl_{cost} is calculated as $\sum a_{cost}$ plus sl_{impact} minus the $sl_{goodness}$. If the sl_{cost} is greater than 0, this indicates that the solution is less economical than allowing the detected behaviour to continue. Otherwise, the solution is ranked by lowest sl_{cost}.

Appendix B

Simulating Malicious Changeload

B.1 Change Instantiations

The following details example change instantiations of the defined change types within Chapter 5. These are categorised by environment and system change, denoting a progression of events that enables a subject to request and obtain access.

Environment Change

In this example, the environment change types, provided in Example 7 (Chapter 5), are instantiated into actual changes relevant to the *LGZLogistics* case study.

(i) **Authentication request** change defines the attributes received as part of a request for authentication within identity provider lgzIS.

$$\textbf{emp0003\_auth\_request} = (\mathit{auth\_request\_type}, \mathit{lgzIS}, \langle \mathit{authRequest}(\mathit{emp}0003, \mathit{password}) \rangle, \langle \mathit{event} \rangle, 1373463234, 0)$$

(ii) **Attribute release request** change could be requested by a resource's policy enforcement point at the time of authentication. However, it is also used by authorisation services as part of a credential validation request (depending on its configuration). The request states a set of identity attribute types (i.e., attribute types that can exist with an identity, such as e-mail), and an identity. The identity, shown as a set of numerical and alphabetical characters, is a privacy protected persistent id (PID), however, equally could denote a non-privacy protected identifier (e.g., an e-mail address).

emp0003_permisRole_request =

$$(attr\_release\_request\_type, lgzIS, \langle attrRequest(\langle pid, bxu915810faa4910\rangle, \langle permisRole\rangle, empDB)\rangle, \langle event\rangle, 1373463236, 0)$$

(iii) **Credential validation request**

emp0003_cred_validation_request =

$$(cred\_validation\_request\_type, as, \langle valRequest(\langle PID, bxu915810faa4910\rangle, lgzIS, \langle\langle notOnOrBefore, 1373462240\rangle, \langle notOnOrAfter, 1373473240\rangle\rangle, \langle\langle permisRole, SysAnalyst\rangle\rangle)\rangle, \langle event\rangle, 1373463240, 0)$$

This change portrays a credential validation request being received in authorisation service as. A privacy protected identity is provided, along with the authenticating identity service (lgzIS), in order for the authorisation service as to validate the subject's released attribute $\langle permisRole, SysAnalyst\rangle$. The two conditions (*notOnOrBefore*, *notOnOrAfter*) state the validity of the released attribute.

(iv) **Access request** change captures the subject emp0003, with assigned identity PID: $bxu915810faa4910$ and attribute $\langle permisRole, SysAnalyst\rangle$, requesting to execute 'read' action on the resource empDB (Payroll service). The change is observed as the receipt of request within the authorisation service as.

emp0003_empDB_request =

$$(access\_request\_type, as, \langle accessRequest(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB, read, \langle NULL\rangle, \langle pid, bxu915810faa4910\rangle)\rangle, \langle event\rangle, 1373463245, 0)$$

(v) **Resource action step** is a change that increments the total bandwidth the subject emp0003 has used within an active session, specifically, to the empDB resource. This change indicates a step change to the attribute activeSessions[emp0003].bandwidth, which contains the subject's current used bandwidth for their active session; the change increases 200kb bandwidth to 800kb.

$$\begin{aligned}\textbf{incr\_emp0003\_empDB\_bandwitdh} = &\\ &(resource\_action\_step\_type, empDB,\\ &\langle 200kb\rangle,\\ &\langle \theta_{empDB}(t) = activeSessions[emp0003].bandwidth + 600kb\rangle,\\ &1373465245, 0)\end{aligned}$$

System Change

This example provides instantiations of the system change types identified in Example 8. An instantiation of a change type is defined as (change_type, srcinst, V_A, V_B, time, duration).

(i) **Authentication decision** change indicates the subject emp0003 authenticating themselves against the identity service lgzIS, which is classified by an event. The change is coupled with the attributes of the request, in order to provide the decision. The decision generates a grant and the generation of a new session for the subject within the lgzIS identity service.

$$\begin{aligned}\textbf{emp0003\_auth} = &(auth\_decision\_type, is,\\ &\langle authDecision(emp0003\_auth\_request)\rangle,\\ &\langle event\rangle,\\ &1373463235, 0)\end{aligned}$$

$$\textbf{authDecision(emp0003\_auth\_request)} = success$$

(ii) **Attribute release** change indicates a change observed at the lgzIS identity service, where a resource empDB has requested the attribute release of attribute type 'permisRole' of the subject emp0003. Identity service lgzIS releases a tuple of attributes that match the request from the resource for the required subject. In this case, it releases $\langle permisRole, SysAnalyst\rangle$. The time indicates the time and date of the attribute release, and as this is not associated to any session, the duration is instant (0).

emp0003_permisRole_release $=$

$(attr\_release\_type, is,$
$\langle attrReleasee(emp0003\_permisRole\_request)\rangle,$
$\langle event\rangle,$
$1373463239, 0)$

attrRelease(emp0003_permisRole_request) $=$

$\langle\langle permisRole, SysAnalyst\rangle\rangle$

(iii) **Credential validation** change indicates a change observed within the authorisation service as, being a credential validation. Credential validation either validates the provided attributes in the request, or pulls the subject's attributes from the identity provider. In this example, the authorisation service has validated the pushed attributes, asserting that $\langle permisRole, SysAnalyst\rangle$ is valid.

emp0003_cred_validation $=$

$(cred\_validation\_type, as,$
$\langle valCredentials(emp0003\_cred\_validation\_request)\rangle,$
$\langle event\rangle,$
$1373463240, 0)$

valCredentials(emp0003_cred_validation_request) $=$

$\langle\langle permisRole, SysAnalyst\rangle\rangle$

(iv) **Access decision** change indicates the authorisation servic as receiving a request and generating an authorisation decision based on the attributes of the request. The authorisation service has granted the request. The change is instant and is only relevant for the specific request, therefore there is no duration.

emp0003_empDB_grant $= (access\_decision\_type, as,$
$\langle accessDecision(emp0003\_empDB\_request)\rangle,$
$\langle event\rangle,$
$1373463245, 0)$

accessDecision(emp0003_empDB_request) $= permit$

B.2 Federated Adaptation

This section identifies several challenges to enabling self-adaptation in federated environments. Specifically, it discusses the problems that exist when enabling adaptation across multiple management domains, and provides a solution via the use of domain managed effectors [7].

B.2.1 Automating the Management of Identity Providers

The ability to manage identity providers relies specifically on the trust that an identity provider has in the (SAAF controller of the) requesting service provider. For example, a service provider identifies malicious behaviour associated with a subject belonging to an identity provider. The service provider might request the identity provider to remove the subject's identity attribute(s) which grant the subject access rights at the service provider. However, these identity attributes may give the subject access rights at many service providers, and not only at the abused service provider. In the latter case the identity provider might easily decide to grant the removal request. In the former case the decision is more difficult and hinges partially on whether the identity provider is more concerned about upsetting its subjects or the many service providers that it has trust relationships with (and which their subject might similarly be abusing). If the request is refused the service provider is left with several options:

- allow the malicious activity to continue (for example, when alternative options have a greater cost when compared to malicious activity), or
- ask the identity provider to alter its attribute release / issuing policy so that it does not issue attribute assertions for this subject, or
- remove access rights from this specific subject (challenging, as it depends on how subjects are identified, i.e., through persistent or transient IDs) or
- remove access rights from all subjects who share the same set of identity attributes with the abusive subject, or
- remove all trust from this particular identity provider (for example, the identity provider has refused numerous adaptation requests despite continued abuse exhibited by its subjects).

To avoid the last option being taken, it is in an identity provider's interest to comply with requests for management changes in relation to either its attribute release policy or one of its subject's identity attributes, otherwise service providers may associate too much risk in using the identity provider. It is for these complex reasons that the autonomic management of identity providers is about the identity provider's output i.e., its assertions about a subject's attributes, so that it is independent of the actual internal mechanisms employed by the identity provider to achieve this. The SAAF controller only depends on the final outcome, which is to control the attributes that the identity provider will assert for a particular subject in the future.

The following discusses the enabling of automated management of federated identity providers, including the assumptions on the identity provider domain, service provider domain, and the role of an identity provider effector.

Identity Provider Domain

It is assumed the identity provider is capable of authenticating a user as being one of its subjects, and of providing attribute assertions about an authenticated subject to service providers. The identity provider is capable of utilising supporting technologies that facilitate the storage and access of subject credentials / privilege attributes (identity services), for example, the use of an LDAP directory. These attributes are assumed to be cryptographically secured and provided to trusted service providers as security assertions, following a standard protocol, such as SAML [99]. It is also assumed that identity providers are able to log and audit security assertion assignments, as well as the authentications made through the identity provider authentication services and any random, transient or session identifiers that are assigned to the subjects in the security assertions. Without these auditing capabilities, identity providers are unable to map session usage to actual subjects, in case they need to identify subjects when responding to notifications of malicious activity.

SAAF Controller and Service Provider Domain

In the case of managing identity providers, the SAAF controller is deployed within the service provider domain. The SAAF controller is placed in the service provider domain, as it is intrinsic to identification of malicious behaviour attributed through the subject's direct actions against the service provider's resources and authorisation services. The controller's *behaviour policy* is defined at deployment by sources of authority within the service provider domain, and relevant to the service provider's environment (e.g., academic / government).

The automated adaptation of the identity provider's domain (specifically, the adaptation of subject attribute assignments) is similar to a non-federated authorisation infrastructure. However, adaptation differs in federated environments, whereby: adaptation is not guaranteed, the identity provider has greater control over how to realise an adaptation, and adaptation may not be immediate. Adaptation is not guaranteed due to an identity provider's ownership over the authorisation assets a SAAF controller aims to adapt. The identity provider decides how adaptation requests are realised based on the interpretation of a SAAF controller's adaptation request. Lastly, adaptation may not be immediate as an identity provider may decide to manually review adaptation requests before performing the adaptation (referred to as semi-automated adaptation).

Identity Provider Effector

The identity provider's effector is under full control of the identity provider and enables the processing of adaptations requested by a service provider's SAAF controller, either synchronously, or asynchronously (with human review). Communication flows between the identity provider's effector and identity provider software are made internally and rely on a host's operating system to ensure security. Communication between a SAAF controller and an identity provider's effector are executed via secure communication, such as TLS / SSL, and require mutual authentication.

The effector requires access to issuing policies, attribute repositories and audit logs, within the identity provider. Access to issuing policies is required in order to adapt the policy controlling the subjects' privilege attributes the identity provider is able to assert. Access to logs is required to map between an identifier (persistent or transient) that the service provider has received, and the internal identifier of the subject. Access to attribute repositories is needed to modify a subject's attributes.

The effector supports a set of abstract adaptations that are necessary when managing an identity provider, as described by the prototype SAAF controller's operator change types (Appendix A.5). It is expected to translate these abstract adaptations into concrete adaptions that are supported by the underlying technology. For example, *remove subject's attribute* may be translated into the relevant LDAP modify command in order to be executed against the LDAP directory, or into the appropriate Shibboleth attribute release policy to prevent a SAML attribute assertion being created. This refers to an identity provider's ability to control how an adaptation request should be realised. In addition to controlling how an adaptation request should be realised, the identity provider can decide which adaptations to allow. To enable this, the effector utilises an authorisation service to determine which operations to allow and which to deny, prior to deciding how to implement accepted adaptations.

B.2.2 SimpleSAMLphp Identity Provider Effector

SimpleSAMLphp is the underlying technology utilised in the defined federated authorisation infrastructure (Chapter 5), to enable communication between identity providers and service providers. As an example of a complex effector, the following describes an effector for a SimpleSAMLphp identity provider. The effector is capable of acting on adaptation requests (from the SAAF controller prototype), as well as controlling the extent of adaptation against an identity provider's authorisation assets. In addition, an extension to SimpleSAMLphp is discussed, which is required for the mapping of persistent and transient IDs to a subject's actual entry within the identity provider's LDAP director.

Extending SimpleSAMLphp

To facilitate operations by the identity provider's effector, simpleSAMLphp's logging capabilities are extended. This is to ensure the correct retrieval of a subject's LDAP distinguished name against persistent and transient IDs supplied to service providers. SimpleSAMLphp stores its log information in a relational database (SQLite). In its original configuration, SimpleSAMLphp was only capable of mapping persistent IDs to subject attribute values. Additional information, such as attribute type, LDAP host, and LDAP search base is needed in order to locate the actual subjects' LDAP entries for both transient and persistent IDs. Whilst some of this information (e.g. LDAP host names) is available in the SimpleSAMLphp configuration file, it is not persistent to configuration changes. For this reason it was decided to record this additional information in a database log, so that any SimpleSAMLphp effector is capable of identify the abusive subject's distinguished name. This was achieved through implementing an additional module to SimpleSAMLphp, which extended the current logging module's functionality.

SimpleSAMLphp Effector

The SimpleSAMLphp effector, shown in Figure B.1. It is a PHP web service hosted alongside the SimpleSAMLphp identity provider service. It has access to the log database stored within the SimpleSAMLphp directory, which enables it to map between persistent and transient IDs and a subject's distinguished name. Web service clients, such as the SAAF controller, can access the effector providing they have been issued with a trusted client X.509 certificate. Mutual SSL / TLS authentication is required and the client's certificate distinguished name is used to identify the requesting client.

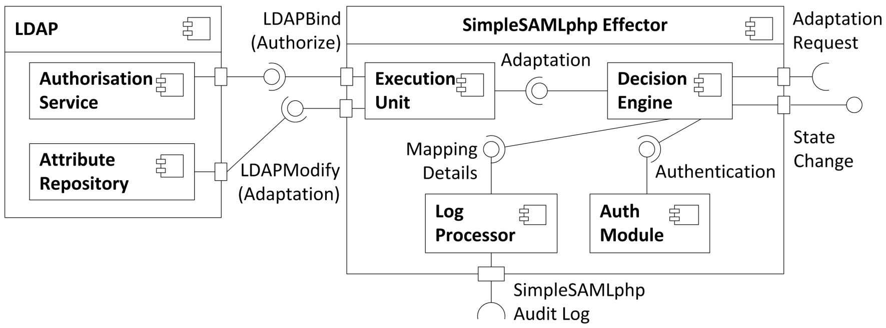


Figure B.1: SimpleSAMLphp Identity Provider Effector

Although the effector component conforms to the conceptual design described in Section B.2.1, it is somewhat restricted due to the limited capabilities of SimpleSAMLphp. SimpleSAMLphp relies upon an attribute repository, such as LDAP, along with an attribute release policy which is represented by a PHP configuration file. The attribute release policy is constrained to stating only which attributes can be released to which service providers, regardless of the individual subject. As a result, the effector adapts subject attributes held in the LDAP repository in order to achieve the per subject granularity.

When operating synchronously, the effector utilises the LDAP access control lists in order to authorise the subject level adaptation requests, notifying requesting clients of failure in case the client is unauthorised. When operating asynchronously, meaning manual review is required, the effector queues requests and notifies administrators via email when new requests are received. Human administrators then review the queued requests before allowing the effector to execute an adaptation and inform the client of success or failure. The effector is initialised once it receives a SOAP message request from a client. From here, SOAP requests are processed in the following manner:

1. Mutually authenticate the requesting client over SSL and obtain the requestor's distinguished name (DN) from its certificate.
2. Verify the requested operation is valid.
3. Retrieve the target subject's unique attribute mapping from the persistent / transient ID stored in the SimpleSAMLphp audit log database.
4. Retrieve the subjects' DN(s) using the relevant LDAP host name and search base;
5. Translate the requested operation into LDAP executable operations.
6. Bind the requestor's DN to the relevant LDAP server.
7. Execute the update operation against LDAP, providing the access control list allows it.
8. Respond to the client with confirmation of the state changes.

B.3 Simulation Experiments

The following section provides supplementary information and data with respect to Chapter 5's simulation experiment. It details the controller's configuration in terms of the behaviour policy and solution policy, along with a break down of solution selection in regards to Exp1 and Exp3. Finally, a high level overview of Exp2 and Exp4 is provided, demonstrating evidence of adaptation under high load.

B.3.1 Controller Configuration

The SAAF controller prototype was configured with a set of behaviour rules and solutions. Behaviour rules denoted the conditions for a violation, where each instance of a violation implied a cost to *LGZLogistics*. With respect to a subject's impact derived from the cost of committing a violation, a maximum and minimum cost of 1500 and 0 are defined for *LGZLogistics* respectively. This provides the upper and lower bounds of subject behaviour. Table B.1 describes the set of behaviour rules as violations, along with the conditions of the violation, and implied cost.

The solutions deployed within the SAAF controller prototype seek to constrain the scope of access in terms of subject adaptation and policy adaptation.

Violation	Description	$\sum a_{cost}$
empDBShortRead	Rate of reads to empDB >20 per min	100
empDBLongRead	Rate of reads to empDB >70/10min	100
empDBShortModify	Rate of modify to empDB >25 per min	150
empDBLongModify	Rate of modify to empDB >50/10min	150
empDBShortDelete	Rate of delete to empDB >5 per min	200
lgtShortAccess	Rate of access to lgT >30 times per min	100
empDBTransaction	Rate of read & modify to empDB >20/10min	200
dueRedundancy	Subject labelled for redundancy in empDB	100

Table B.1: Behaviour rules (violation types)

Table B.2 describes the set of solutions applicable to the simulation experiment. Each solution is defined by a solution ID, description of the actions to be taken, an aggregate cost to *LGZLogistics* as a consequence of enacting the solution, and a minimum subject impact.

Solution	Description	$\sum a_{cost}$	Min_{Impact}
S1	Remove a subject's attribute	50	0.25
S2	Remove all of a subject's attributes	150	0.5
S3	Remove trust in an IdP issuing an attribute	800	0.7
S4	Remove all trust in an IdP	1500	0.7
S5	Disable all access	5000	0.7

Table B.2: Solutions with cost and minimum impact

Lastly, the controller was configured to perceive malicious subjects in terms of subjects yet to be successfully mitigated. For example, if a subject commits a violation and mitigation is successful, the subject is no longer deemed to be malicious. However, if mitigation fails or if the subject commits another violation, the controller considers this subject as malicious, which is necessary to identify an appropriate solution in solution selection. This differs from previous experiments as conveyed in Chapter 4, where the controller's perception of malicious subjects was constrained to all subjects who had committed a violation within the last 30 days.

B.3.2 Adaptation under Normal Load: Solution Selection

To provide evidence of solution ranking, the following describes how solutions were identified, ranked, and selected, with respect to Exp1 and Exp3. Exp2 and

Exp4 are not discussed as they convey the same adaptation scenarios as Exp1 and Exp3, yet under high load.

Exp1 Solution Selection

Table B.3 describes the solution selection and ranking for each stage of simulation within Exp1. It reflects adaptation under normal load where the SAAF controller prototype has full control over the authorisation service `as` and identity provider's `lgzIS` and `conIS`. In this scenario no policy adaptation was observed due to the way in which the controller's perception of malicious subjects was configured (i.e., subjects are only considered malicious if previous mitigation attempts have failed). Note that the calculated S_{Mal} and S_{NonMal} reflect the current state of the $ABAC_M$ and Bhv_M, and the cost of removing a single non-malicious subject's access is synonymous with solution S2 (i.e., 150).

	User	Impact	sl	$\sum a_{cost}$	sl_{impact}	$sl_{goodness}$	S_{Mal}	S_{NonMal}	sl_{cost}	Rank
1	emp3	0.07	S0	-	-	-	-	-	-	-
2	emp4	0.07	S0	-	-	-	-	-	-	-
3	emp5	0.07	S0	-	-	-	-	-	-	-
4	emp6	0.07	S0	-	-	-	-	-	-	-
5	emp3	0.4	S1	50	0	300	1	0	-250	1
6	con2	0.07	S0	-	-	-	-	-	-	-
7	con2	0.27	S1	50	0	200	1	0	-150	1
8	con3	0.07	S0	-	-	-	-	-	-	-
9	con4	0.07	S0	-	-	-	-	-	-	-
10	con3	0.27	S1	50	0	200	1	0	-150	1
11	con5	0.07	S0	-	-	-	-	-	-	-
12	con4	0.27	S1	50	0	200	1	0	-150	1
13	con5	0.27	S1	50	0	200	1	0	-150	1
14	emp3	0.8	S2	150	0	400	1	0	-250	1
			S3	800	149850	400	1	999	150250	N/A
			S4	1500	149400	700	4	996	150200	N/A
			S5	5000	149550	1500	8	997	153050	N/A

Table B.3: Exp1: Solution selection and ranking

Exp3 Solution Selection

Table B.4 describes the solution selection and ranking for each stage of simulation within Exp3. It differs from Exp1 due to the fact that the contractor's identity provider effector was disabled. This prevented adaptation at a subject level from subjects within the contractor's management domain. As a result, subjects from the contractor organisation were able to persist in violations requiring the controller to consider and enact policy adaptation.

A key outcome of this experiment's solution selection is the fact that a solution's goodness ($sl_{goodness}$) increases as subjects persist in performing malicious behaviour. This persistence of malicious behaviour occurs due to prior failures in subject adaptation. As a result, when multiple subjects perform violations, the solution goodness increases. Note that if a solution is capable in mitigating historic violations where mitigation failed in the past, such as with subject adaptation, solution goodness encompasses the cost of these violations as well. This enables the controller to enact solutions with a greater consequence to the organisation.

Note that the calculated S_{Mal} and S_{NonMal} reflect the current state of the $ABAC_M$ and Bhv_M, and that these are dependent on how a solution will adapt the state of access. For example, if a solution removes all access for subjects with the role of *ContractorSupervisor*, S_{Mal} and S_{NonMal} reflect the number of malicious contractor supervisors, and non-malicious contractor supervisors. Lastly, the cost of removing a single non-malicious subject's access is synonymous with solution S2 (i.e., 150).

B.3.3 Adaptation under High Load: Summary of Results

Table B.5 describes the attack steps of Exp2 performed as part of Chapter 5's simulation experiment. It details the persistence of malicious subject behaviour and the SAAF autonomic controller prototype mitigating malicious behaviour under a high load. Malicious behaviour is successfully mitigated through subject adaptation, whereby adaptation requests are sent to identity provider effectors, detailing the removal of subject access.

Table B.6 describes the attack steps of Exp4 performed as part of Chapter 5's simulation experiment. It details the persistence of malicious subject behaviour and the SAAF autonomic controller prototype mitigating malicious behaviour under a high load. Malicious behaviour is successfully mitigated through policy adaptation, whereby access control policies are generated from an adapted modelled state of access, and deployed within an authorisation service. Subject adaptation is shown to fail, whereby the deactivation of the contractor identity provider effector prevented success. This demonstrates the consequences to a non-cooperating trusted business partner.

	User	Impact	sl	$\sum a_{cost}$	sl_{impact}	$sl_{goodness}$	S_{Mal}	S_{NonMal}	sl_{cost}	Rank
1	emp3	0.07	S0	-	-	-	-	-	-	-
2	emp4	0.07	S0	-	-	-	-	-	-	-
3	emp5	0.07	S0	-	-	-	-	-	-	-
4	emp6	0.07	S0	-	-	-	-	-	-	-
5	emp3	0.4	S1	50	0	300	1	0	-250	1
6	con2	0.07	S0	-	-	-	-	-	-	-
7	con2	0.27	S1	50	0	200	1	0	-150	1
8	con3	0.07	S0	-	-	-	-	-	-	-
9	con4	0.07	S0	-	-	-	-	-	-	-
10	con2	0.6	S2	50	0	200	1	0	-150	1
11	con2	1	S2	150	0	400	1	0	-250	1
			S3	800	150	400	1	1	550	N/A
			S4	1500	300	600	3	2	1200	N/A
			S5	5000	149700	1200	7	998	153500	N/A
12	con3	0.27	S1	150	0	200	1	0	-150	1
13	con5	0.07	S0	-	-	-	-	-	-	-
14	con2	1	S2	150	0	500	1	0	-350	1
			S3	800	150	500	1	1	450	N/A
			S4	1500	150	900	4	1	750	N/A
			S5	5000	149550	1500	8	997	1530350	N/A
15	con4	0.27	S1	50	0	200	1	0	-150	1
16	con3	0.6	S2	150	0	300	1	0	-150	1
17	con2	1	S2	150	0	600	1	0	-450	1
			S3	800	150	600	1	1	350	N/A
			S4	1500	150	1200	4	1	450	N/A
			S5	5000	149550	1800	8	997	152750	N/A
18	con5	0.27	S1	50	0	200	1	0	-150	1
19	con3	1	S2	150	0	400	1	0	-250	1
			S3	800	0	800	3	0	0	2
			S4	1500	150	1400	4	1	250	N/A
			S5	5000	149550	2000	8	997	152550	N/A
20	con2	1	S2	150	0	700	1	0	-550	1
			S3	800	150	700	1	1	250	N/A
			S4	1500	150	1500	4	1	150	N/A
			S5	5000	149550	2100	8	997	152450	N/A
21	con2	1	S2	150	0	800	1	0	-650	A
			S3	800	150	800	1	1	150	N/A
			S4	1500	150	1600	4	1	50	N/A
			S5	5000	149550	2200	8	997	152350	N/A
22	con2	1	S2	150	0	900	1	0	-750	1
			S3	800	150	900	1	1	50	N/A
			S4	1500	150	1700	4	1	-50	2
			S5	5000	149550	2300	8	997	152250	N/A
23	emp3	0.8	S2	150	0	400	1	0	-250	1
			S3	800	149850	400	1	999	150250	N/A
			S4	1500	149400	700	4	996	150200	N/A
			S5	5000	149550	2400	8	997	152150	N/A

Table B.4: Exp3: Solution selection and ranking

Step	Subject	Impact	Violation	Identified Solutions	Selected Solution	React Time (ms) (Avg, Std)	Adapt Time (ms) (Avg, Std)	Result
1	emp03	0.07	`dueRedundancy`	`S0`	`S0`	4.8, 0.5	N/A	N/A
2	emp04	0.07	`dueRedundancy`	`S0`	`S0`	1.6, 0.5	N/A	N/A
3	emp05	0.07	`dueRedundancy`	`S0`	`S0`	1.6, 0.5	N/A	N/A
4	emp06	0.07	`dueRedundancy`	`S0`	`S0`	2, 0	N/A	N/A
5	emp03	0.4	`empDBTransaction`	`S1`	`S1`	289.8, 45.9	156, 37.9	1
6	con02	0.07	`empShortRead`	`S0`	`S0`	1.8, 0.4	N/A	N/A
7	con02	0.27	`empShortRead`	`S1`	`S1`	102.4, 25.4	139.4, 20.4	1
8	con03	0.07	`empShortRead`	`S0`	`S0`	5.4, 3.2	N/A	N/A
9	con04	0.07	`empShortRead`	`S0`	`S0`	1.6, 0.5	N/A	N/A
10	con03	0.27	`empShortRead`	`S1`	`S1`	97, 46.5	132.8, 18.9	1
11	con05	0.07	`empShortRead`	`S0`	`S0`	1.8, 0.4	N/A	N/A
12	con04	0.27	`empShortRead`	`S1`	`S1`	127.4, 62.4	152.4, 18.9	1
13	con05	0.27	`empShortRead`	`S1`	`S1`	86, 37.6	114.4, 16.7	1
14	emp03	0.8	`lgTShortAccess`	`S2,S3,S4,S5`	`S2`	273, 80.9	187.8, 93.9	1

Table B.5: Exp2: Adaptation under high load

Step	Subject	Impact	Violation	Identified Solutions	Selected Solution	React Time (ms) (Avg, Std)	Adapt Time (ms) (Avg, Std)	Result
1	emp03	0.07	`dueRedundancy`	`S0`	`S0`	4.6, 0.5	N/A	N/A
2	emp04	0.07	`dueRedundancy`	`S0`	`S0`	1.8, 0.4	N/A	N/A
3	emp05	0.07	`dueRedundancy`	`S0`	`S0`	2, 0.7	N/A	N/A
4	emp06	0.07	`dueRedundancy`	`S0`	`S0`	2.2, 0.4	N/A	N/A
5	emp03	0.4	`empDBTransaction`	`S1`	`S1`	294, 63.8	204.2, 24.8	1
6	con02	0.07	`empShortRead`	`S0`	`S0`	1.6, 0.5	N/A	N/A
7	con02	0.27	`empShortRead`	`S1`	`S1`	147.2, 88.6	129.2, 37.8	0
8	con03	0.07	`empShortRead`	`S0`	`S0`	6, 3.2	N/A	N/A
9	con04	0.07	`empShortRead`	`S0`	`S0`	1.6, 0.5	N/A	N/A
10	con02	0.6	`empShortRead`	`S2`	`S2`	101.2, 35.6	11.8, 48.4	0
11	con02	1	`empLongRead`	`S2,S3,S4,S5`	`S2`	387.8, 80.9	133.8, 54.1	0
12	con03	0.27	`empShortRead`	`S1`	`S1`	86.2, 45.8	110.6, 40.7	0
13	con05	0.07	`empShortRead`	`S0`	`S0`	1.6, 0.5	N/A	N/A
14	con02	1	`empShortRead`	`S2,S3,S4,S5`	`S2`	235, 39.1	119.4, 50.6	0
15	con04	0.27	`empShortRead`	`S1`	`S1`	39.2, 7.5	97.4, 40.8	0
16	con03	0.6	`empShortRead`	`S2`	`S2`	43.8, 19.2	101.2, 53.5	0
17	con02	1	`empShortRead`	`S2,S3,S4,S5`	`S2`	159.8, 13.8	100, 36	0
18	con05	0.27	`empShortRead`	`S1`	`S1`	33.6, 4.4	136, 37.9	0
19	con03	1	`empLongRead`	`S2,S3,S4,S5`	`S2`(F),`S3`	137.6, 34.1	1003.8, 34.3	1
20	con02	1	`empShortRead`	`S2,S3,S4,S5`	`S2`	293, 72.6	97.6, 25.2	0
21	con02	1	`empLongRead`	`S2,S3,S4,S5`	`S2`	233.4, 75.7	94.2, 35.5	0
22	con02	1	`empShortRead`	`S2,S3,S4,S5`	`S2`(F),`S4`	169.8, 57.2	834.4, 34.3	1
23	emp03	0.8	`lgTShortAccess`	`S2,S3,S4,S5`	`S2`	168.8, 39.3	150.6, 50.9	1

Table B.6: Exp4: Adaptation under high load (contractor IdP effector failure)

Appendix C

Gamification Analysis

C.1 Guidance to Players

Participants of the game experiment were provided with the following information (conveyed in Figure C.1 and Figure C.2) in order to have a clear understanding of the experiment, and data collected.

Paramapada Sopaanam (Snakes and Ladders)

Project: Evaluation of self-adaptive authorization in terms of identification and response to malicious usage of online resources

Dear participant,

We invite you to play a game of snakes and ladders.

For this research, we wish to develop tools that better protect programs against malicious attacks, by understanding the different ways that people conduct malicious activity. To help us with this, we encourage you to play a game of snakes & ladders normally (at first) and then attempt to win the game in as few turns as possible, by any means you wish to use (i.e, exploit glitches, inject code).

There are no foreseeable risk or disadvantages to participating within this study. There are no benefits to playing the game, other than to enable us to evaluate our research. No personal data is stored or mapped to participant accounts (making participants personally unidentifiable).

The data collected will be used to solely evaluate our system, and the data along with the results of the evaluation may be submitted for publication to a relevant conference, scientific journal, and or PhD thesis.

Thank you for reading this participant information, and for your consideration in taking part of this study.

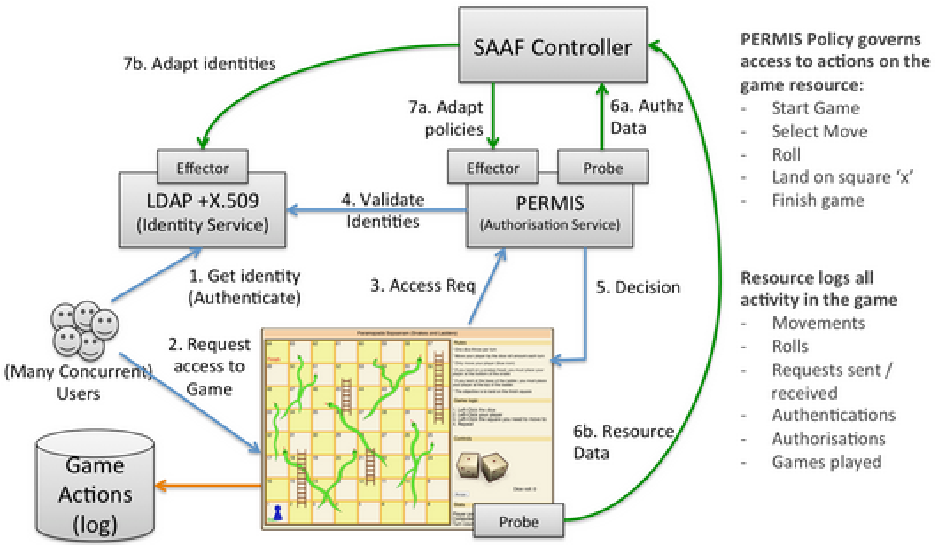
This study is organised by Christopher Bailey (a PhD student at the University of Kent), supervised by Rogerio de Lemos. Should you have any questions, complaints, or wish to withdraw your participation, please contact Christopher Bailey.

Create an account   Login

Figure C.1: Player participation declaration

Game Authorisation Infrastructure

Architecture



Description

The game operates within a protected environment, whereby an authorisation infrastructure exists to protect the game, and the actions executable within the game. As a user, you are utilising an LDAP based identity service to authenticate with, and your access is governed by an external authorisation service in regards to access control.

When you create an account, an LDAP entry is automatically created and stored within this identity service, where by an X.509 certificate is issued to you. Your X.509 certificate contains a set of access rights which is validated by the authorisation service when you attempt to access certain actions within the game. A policy enforcement point within the game ensures that as a user, you must obtain access before carrying out an action within the game.

An autonomic 'SAAF' controller is deployed within the architecture, whereby it is constantly monitoring the state of the authorisation infrastructure, and the behaviour of subjects. It derives a perception of state of access, and state of user behaviour from observation of system and environment change. The controller exists to identify any malicious activity as a result of user interaction with the game. Upon identification, it attempts to mitigate such activity via a variety of strategies.

Figure C.2: Advice to participants

In addition, the only instruction provided to players on how to play the game is conveyed in Figure C.3. This was provided to the player within the game interface.

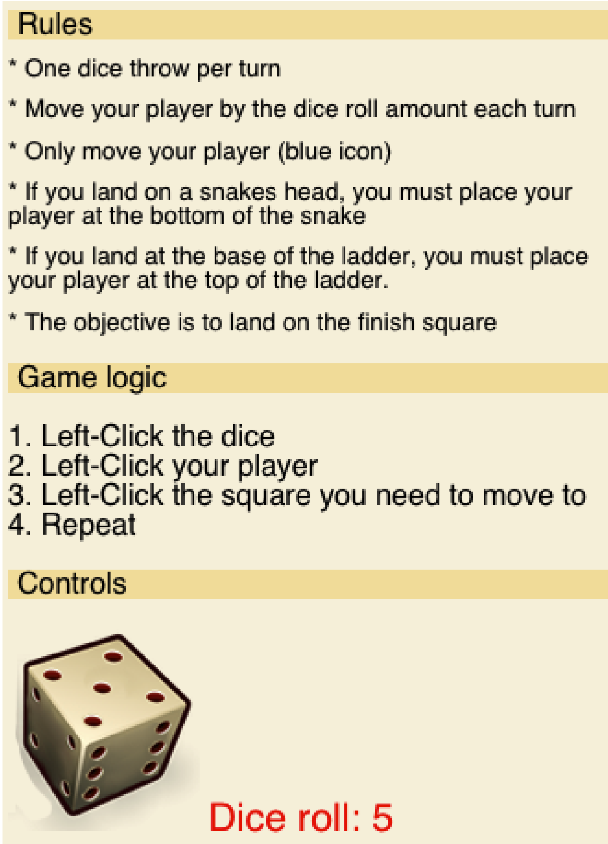


Figure C.3: Games Instructions

C.2 Game Data

This section presents a set of tables reflecting certain aspects of data generated through the gamification experiment.

Table C.1 details the number of granted authorisation requests made per each phase of the gamification experiment. Authorisation requests are categorised in terms of the protected actions within the game (e.g., start, end, roll, etc.). Phase 2, being the longest phase generated the largest amount of authorisation requests, where roll and move actions are seen to be the most common actions requested.

Table C.2 details the number of observed and recorded actions within the game, in order to compare to the number of authorisation requests made. As such, it was noted that there was evidence of attempts to falsify authorisation requests (with no corresponding action), and evidence where users had bypassed authorisation. In addition some users were found to be bypassing authorisation, where actions were observed with no authorisation request.

Permitted Authorisations	Phase 1	Phase 2	Phase 3	Total
Start	168	692	326	1186
End	58	250	98	406
Roll	1165	4412	1582	7159
Move	755	3233	1079	5067
Ladder	87	371	145	603
Bonus	30	151	62	242

Table C.1: Permitted game authorisation requests

Performed Actions	Phase 1	Phase 2	Phase 3	Total
Start	168	692	326	1186
End	58	250	98	406
Roll	1219	4356	1535	7110
Move	757	3235	1081	5073
Ladder	87	371	145	603
Bonus	30	151	62	242

Table C.2: Observed game actions

Table C.3 describes the number of previous games a malicious player has played prior to the SAAF controller detecting the player in performing an injection attack (either a change to the game code resulting in unexpected behaviour, or a change to the game's session). This provides evidence of player experience prior to performing a sophisticated attack.

Games	Violations	Players	Games	Violations	Players
1	1	3	8	4	1
1	2	2	8	9	1
1	4	1	8	11	2
1	5	1	9	8	1
2	1	2	9	11	2
2	3	1	9	22	1
2	4	1	10	4	1
3	3	1	10	6	1
3	5	3	11	8	1
3	10	1	11	22	1
4	6	1	12	16	1
4	8	1	16	13	1
4	12	1	21	13	1
4	14	1	27	32	1
5	2	1	32	21	1
5	7	1	50	11	2
5	8	1	62	22	1
5	9	1	72	29	1

Table C.3: Number of players detected of injection attacks versus number of previous games played and detected violations

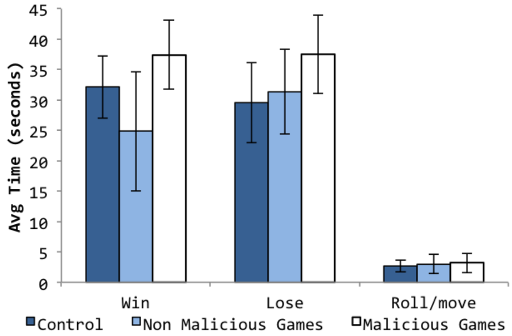
C.3 Player Behaviour

Figure C.4 presents a set of charts that measure aspects of behaviour from differing perspectives. These charts compare games played in the control phase, to non-malicious and malicious games played throughout phases 1 to 3.

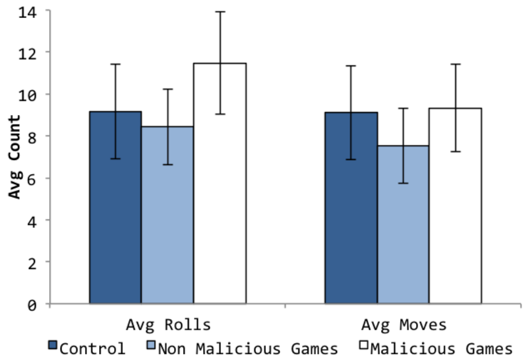
In chart (a), the average time for a game action to be completed is compared, including the time it took for a game to be won, lost, and a turn (i.e., roll then move) to be achieved[1]. Whilst a malicious game is shown to take longer to complete, possibly due to a malicious action requiring more thought than playing the game legitimately, time cannot be considered as a factor in determining malicious behaviour. This is justified by the control games played, where there is little margin of difference.

In regards to chart's (c) and (d), the average number of actions are observed within winning games. Considering rolls and moves alone, the average count shares a similar margin. However, given that the game is based on chance, this

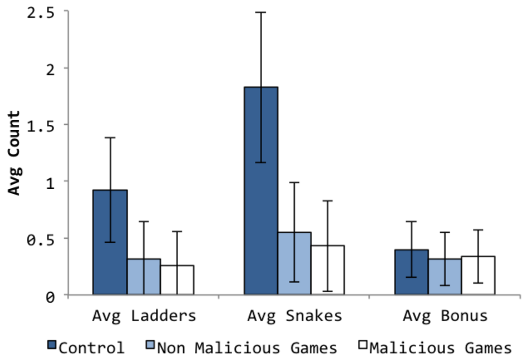
[1]Outliers beyond 60s for time to win and lose, and 20s between turns, were removed to avoid extremities.



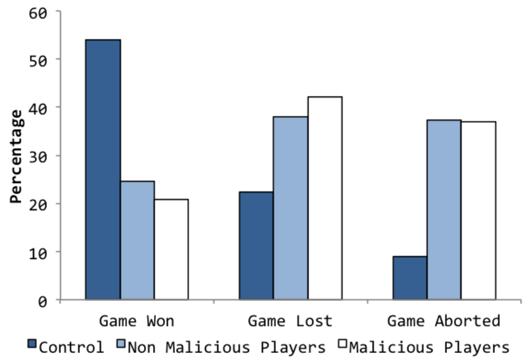
(a) Avg. time per game actions



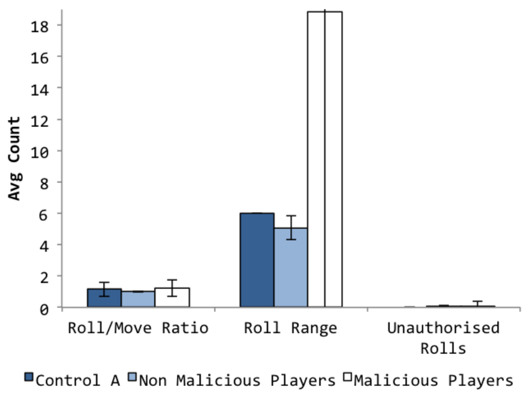
(b) Avg. rolls and moves per winning game type



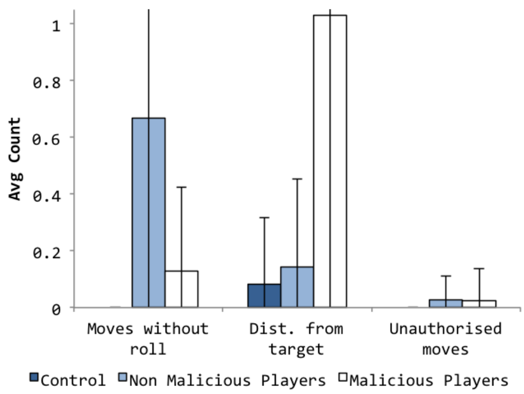
(c) Avg. ladders, snakes, and bonus squares per winning game type



(d) Avg. ratio of games won, lost, and aborted, per player type



(e) Roll features per player type



(f) Move features per player type

Figure C.4: Trends in game and player behaviour is not a useful indication of malicious behaviour. Of particular interest is the average number of snakes that were used throughout a winning game. Control games exhibited a higher average of snakes in comparison to that of malicious games won in phases 1 to 3. This is thought to be a result of malicious games purposely ignoring snakes within the game.

The low number of snakes within non malicious games is significant to the fact that many players from phases 1 to 3 had a greater percentage of aborted games, as shown by chart (d). Observing behaviour from a player perspective, control players aborted an average of 9% of their games, increasing the likelihood of playing a game all the way through and encountering snakes. It is suggested players from phase 1 to 3 simply chose to abort rather than encounter a snake and finish a game.

One can assume that a player who frequently aborts a game is attempting to identify an ideal game of chance, rather than play a game all the way through, and is therefore malicious. However, to follow the view that malicious players are less likely to land on a snake in a game is considered purely circumstantial (as legitimate games can be completed without encountering any snakes).

In order to capture clearer indications of malicious behaviour it is necessary to observe particular features of actions within a game. Chart (e) portrays a set of contextual features related to roll actions (comparing types of player), such as, the ratio of rolls to move, the range of dice rolls, and average number of rolls that were unauthorised. Chart (f) portrays a set of contextual features related to move actions, such as, the average number of moves without a corresponding roll, the distance from the expected target of a move (i.e., where the player should have landed), and average number of moves unauthorised.

These features present a much clearer indication of malicious behaviour. For example, considering the legal roll range of 1 to 6, control players never performed a dice roll beyond 6. However, malicious players from phase 1 to 3 had a far higher average of 18. This was largely increased by a small number of malicious players who rolled high dice values, evidential by a high standard deviation (62.71).

Similarly, considering the average distance from a move's required target, malicious players exhibited a much higher average. Here, an average closer to zero indicates players that landed close to a move's target square. As with the range of rolls, a high standard deviation was observed (9.73), where the average was increased by a small number of malicious players landing on a square with a greater distance from their required target.

Lastly, in chart (f), it was observed that there was a higher average of moves without rolls for non-malicious players in phases 1 to 3. The high standard deviation (1.44) is a result of only a few non-malicious accounts performing moves without a roll. This particular behaviour is significant of the number of unknown attacks that were successful in phases 1 to 3, where the resource game's detectors had not been configured to report such activity.

Bibliography

[1] Andersson, J., de Lemos, R., Malek, S. and Weyns, D. (2009). Modeling dimensions of self-adaptive software systems. In B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi and J. Magee, eds., *Software Engineering for Self-Adaptive Systems*, *Lecture Notes in Computer Science*, vol. 5525, Berlin, Heidelberg: Springer-Verlag, pp. 27–47.

[2] Andersson, J., de Lemos, R., Malek, S. and Weyns, D. (2009). Reflecting on self-adaptive software systems. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Washington, DC, USA: IEEE Computer Society, SEAMS '09, pp. 38–47.

[3] Apache Software Foundation (n.d.). JMeter v.2.11 [Online]. Available from: http://jmeter.apache.org/ [Accessed 10 August 2014].

[4] Axiomatics (n.d.). Axiomatics policy server [Online]. Available from: https://www.axiomatics.com/axiomatics-policy-server.html [Accessed 17 January 2014].

[5] Bailey, C., Chadwick, D. W. and de Lemos, R. (2011). Self-adaptive authorization framework for policy based RBAC/ABAC models. In *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, Washington, DC, USA: IEEE Computer Society, DASC '11, pp. 37–44.

[6] Bailey, C., Chadwick, D. W. and de Lemos, R. (2014). Self-adaptive federated authorization infrastructures. *Journal of Computer and System Sciences*, 80(5), pp. 935–952, Special Issue on Dependable and Secure Computing The 9th {IEEE} International Conference on Dependable, Autonomic and Secure Computing.

[7] Bailey, C., Chadwick, D. W., de Lemos, R. and Siu, K. W. S. (2013). Enabling the autonomic management of federated identity providers. In *Proceedings of the 7th*

IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet - Volume 7943, Berlin, Heidelberg: Springer-Verlag, AIMS'13, pp. 100–111.

[8] Bailey, C., Montrieux, L., de Lemos, R., Yu, Y. and Wermelinger, M. (2014). Run-time generation, transformation, and verification of access control models for self-protection. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, New York, NY, USA: ACM, SEAMS 2014, pp. 135–144.

[9] Baracaldo, N. and Joshi, J. (2013). An adaptive risk management and access control framework to mitigate insider threats. *Comput Secur*, 39, pp. 237–254.

[10] BBC (2014). Credit card details on 20 million South Koreans stolen [Online]. Available from: http://www.bbc.co.uk/news/technology-25808189 [Accessed 5 January 2014].

[11] Bell, D. E. (2005). Looking back at the bell-la padula model. In *Proceedings of the 21st Annual Computer Security Applications Conference*, Washington, DC, USA: IEEE Computer Society, ACSAC '05, pp. 337–351.

[12] Benantar, M. (2005). *Access Control Systems: Security, Identity Management and Trust Models*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

[13] Bertino, E., Catania, B., Damiani, M. L. and Perlasca, P. (2005). Geo-rbac: A spatially aware RBAC. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, New York, NY, USA: ACM, SACMAT '05, pp. 29–37.

[14] Bertino, E., Kamra, A., Terzi, E. and Vakali, A. (2005). Intrusion detection in RBAC-administered databases. In *Proceedings of the 21st Annual Computer Security Applications Conference*, Washington, DC, USA: IEEE Computer Society, ACSAC '05, pp. 170–182.

[15] Bézivin, J. (2004). In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2), pp. 21–24.

[16] Bézivin, J. (2006). Model driven engineering: An emerging technical space. In R. Laemmel, J. Saraiva and J. Visser, eds., *Generative and Transformational Techniques in Software Engineering*, *Lecture Notes in Computer Science*, vol. 4143, Springer Berlin Heidelberg, pp. 36–64.

[17] Bishop, M., Conboy, H. M., Phan, H., Simidchieva, B. I., Avrunin, G. S., Clarke, L. A., Osterweil, L. J. and Peisert, S. (2014). Insider threat identification by process analysis. In *Security and Privacy Workshops (SPW)*, IEEE, pp. 251–264.

[18] Bistarelli, S., Martinelli, F. and Santini, F. (2010). A formal framework for trust policy negotiation in autonomic systems: Abduction with soft constraints. In *Proceedings of the 7th International Conference on Autonomic and Trusted Computing, ATC'10*, vol. 6407, Berlin, Heidelberg: Springer-Verlag, pp. 268–282.

[19] Böhm, K., Etalle, S., Den Hartog, J., Hütter, C., Trabelsi, S., Trivellato, D. and Zannone, N. (2010). A flexible architecture for privacy-aware trust management. *J Theor Appl Electron Commer Res*, 5(2), pp. 77–96.

[20] Booth, R., Brooke, H. and Moriss, S. (2010). WikiLeaks cables: Bradley Manning faces 52 years in jail [Online]. Available from: http://www.theguardian.com/world/2010/nov/30/wikileaks-cables-bradley-manning [Accessed 5 January 2014].

[21] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. and Winer, D. (2000). Simple object access protocol (SOAP) 1.1.

[22] Brun, Y., Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M. and Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi and J. Magee, eds., *Software Engineering for Self-Adaptive Systems*, *Lecture Notes in Computer Science*, vol. 5525, Berlin, Heidelberg: Springer-Verlag, pp. 48–70.

[23] Budinsky, F. (2004). *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional.

[24] Cámara, J., Correia, P., de Lemos, R. and Vieira, M. (2014). Empirical resilience evaluation of an architecture-based self-adaptive software system. In *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, New York, NY, USA: ACM, QoSA '14, pp. 63–72.

[25] Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G. and Fox, A. (2004). Microreboot - a technique for cheap recovery. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, Berkeley, CA, USA: USENIX Association, OSDI'04, pp. 31–45.

[26] Cappelli, D., Desai, A., Moore, A., Shimeall, T., Weaver, E., Willke, B. et al. (2006). Management and education of the risk of insider threat (MERIT): System dynamics modeling of computer system sabotage. *Joint CERT Coordination Center/SEI and CyLab at Carnegie Mellon University Report, Pittsburgh, PA*, pp. 1–34.

[27] Cappelli, D. M., Moore, A. P. and Trzeciak, R. F. (2012). *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes*. Addison-Wesley Professional, 1st edn.

[28] Caputo, D., Maloof, M. and Stephens, G. (2009). Detecting insider theft of trade secrets. *IEEE Security and Privacy*, 7(6), pp. 14–21.

[29] Cardenas, A., Amin, S. and Sastry, S. (2008). Secure control: Towards survivable cyber-physical systems. In *First International Workshop on Cyber-Physical Systems (WCPS2008)*, pp. 495–500.

[30] Carver Jr, C. A. (2001). Adaptive agent-based intrusion response. Tech. rep., DTIC.

[31] Chadwick, D. W. and Otenko, A. (2002). The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, New York, NY, USA: ACM, SACMAT '02, pp. 135–140.

[32] Chadwick, D. W., Zhao, G., Otenko, S., Laborde, R., Su, L. and Nguyen, T. A. (2008). PERMIS: A modular authorization infrastructure. *Concurr Comput : Pract Exper*, 20(11), pp. 1341–1357.

[33] Chandola, V., Banerjee, A. and Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput Surv*, 41(3), pp. 15:1–15:58.

[34] Cheng, B. H., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D. and Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi and J. Magee, eds., *Software Engineering for Self-Adaptive Systems*, *Lecture Notes in Computer Science*, vol. 5525, Berlin, Heidelberg: Springer-Verlag, pp. 1–26.

[35] Cheng, P., Rohatgi, P., Keser, C., Karger, P. A., Wagner, G. M. and Reninger, A. S. (2007). Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Washington, DC, USA: IEEE Computer Society, SP '07, pp. 222–230.

[36] Cheng, S.-W. and Garlan, D. (2012). Stitch: A language for architecture-based self-adaptation. *J Syst Softw*, 85(12), pp. 2860–2875.

[37] Chung, C. Y., Gertz, M. and Levitt, K. (2000). DEMIDS: A misuse detection system for database systems. In M. E. van Biene-Hershey and L. Strous, eds., *Integrity and Internal Control Information Systems*, Norwell, MA, USA: Kluwer Academic Publishers, pp. 159–178.

[38] Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, London, UK, UK: Springer-Verlag, CAV '02, pp. 359–364.

[39] Cunningham, P., Anderson, R., Mullins, R., Taylor, G. and Moore, S. (2002). Improving smart card security using self-timed circuits. In *Proceedings of the 8th International Symposium on Asynchronus Circuits and Systems*, Washington, DC, USA: IEEE Computer Society, ASYNC '02, pp. 211–219.

[40] da Silva, C. E. and de Lemos, R. (2011). Dynamic plans for integration testing of self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, New York, NY, USA: ACM, SEAMS '11, pp. 148–157.

[41] Damianou, N., Dulay, N., Lupu, E. and Sloman, M. (2001). The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, London, UK, UK: Springer-Verlag, POLICY '01, pp. 18–38.

[42] de Lemos, R., Giese, H., Müller, H., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Gschka, K., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezz, M., Prehofer, C., Schfer, W., Schlichting, R., Smith, D., Sousa, J., Tahvildari, L., Wong, K. and Wuttke, J. (2013). Software engineering

for self-adaptive systems: A second research roadmap. In R. de Lemos, H. Giese, H. Müller and M. Shaw, eds., *Software Engineering for Self-Adaptive Systems II*, *Lecture Notes in Computer Science*, vol. 7475, Springer Berlin Heidelberg, pp. 1–32.

[43] Debar, H., Dacier, M. and Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. *Comput Networks*, 31(9), pp. 805–822.

[44] Demchenko, Y., Gommans, L. and Laat, C. (2007). Extending role based access control model for distributed multidomain applications. In H. Venter, M. Eloff, L. Labuschagne, J. Eloff and R. Solms, eds., *New Approaches for Security, Privacy and Trust in Complex Environments*, *IFIP International Federation for Information Processing*, vol. 232, Springer US, pp. 301–312.

[45] Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N. and Zambonelli, F. (2006). A survey of autonomic communications. *ACM Trans Auton Adapt Syst*, 1(2), pp. 223–259.

[46] Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Mach Learn*, 29(2-3), pp. 103–130.

[47] Eberle, W. and Holder, L. (2009). Graph-based approaches to insider threat detection. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, New York, NY, USA: ACM, CSIIRW '09, pp. 44:1–44:4.

[48] Enterprise XACML (n.d.). [Online]. Available from: http://code.google.com/p/enterprise-java-xacml/ [Accessed 17 January 2014].

[49] Ferraiolo, D., Cugini, J. and Kuhn, D. R. (1995). Role-based access control (rbac): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pp. 241–48.

[50] Fishburn, P. C. (1970). Utility theory for decision making. Tech. rep., DTIC.

[51] Fisler, K., Krishnamurthi, S., Meyerovich, L. A. and Tschantz, M. C. (2005). Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, New York, NY, USA: ACM, ICSE '05, pp. 196–205.

[52] Floyd, R. W. (1967). Nondeterministic algorithms. *J ACM*, 14(4), pp. 636–644.

[53] Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B. and Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10), pp. 46–54.

[54] Greenemeier, L. (2007). Massive insider breach at DuPont [Online]. Available from: http://www.informationweek.com/massive-insider-breach-at-dupont/d/d-id/1052047 [Accessed 5 January 2014].

[55] Hellerstein, J. L., Diao, Y., Parekh, S. and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*. John Wiley & Sons.

[56] Horn, P. (2001). Autonomic computing: IBM's perspective on the state of information technology. Tech. rep., IBM.

[57] Hu, V. C., Kuhn, D. R., Xie, T. and Hwang, J. (2011). Model checking for verification of mandatory access control models and properties. *International Journal of Software Engineering and Knowledge Engineering*, 21(01), pp. 103–127.

[58] Hu, V. C., Schnitzer, A., Sandlin, K. and Scarfone, K. (2013). Guide to attribute based access control (abac) definition and considerations. NIST Special Publication.

[59] Hughes, G. and Bultan, T. (2008). Automated verification of access control policies using a SAT solver. *Int J Softw Tools Technol Transf*, 10(6), pp. 503–520.

[60] Huotari, K. and Hamari, J. (2012). Defining gamification: A service marketing perspective. In *Proceeding of the 16th International Academic MindTrek Conference*, New York, NY, USA: ACM, MindTrek '12, pp. 17–22.

[61] IBM (n.d.). IBM Security Intelligence with Big Data [Online]. Available from: http://www-03.ibm.com/security/solution/intelligence-big-data/ [Accessed 20 July 2014].

[62] ID Analytics, Inc. (2008). Analysis of internal data theft. Tech. rep., ID Analytics, Inc.

[63] ITU (2009). NGN Identity Management Framework, Y.2720.

[64] ITU-T Rec. X.509 (2000). The Directory: Authentication Framework. ISO/IEC 9594-8.

[65] Jakobwitz, S. and Egan, V. (2006). The dark triad and normal personality traits. *Personality and Individual Differences*, 40(2), pp. 331–339.

[66] Janicke, H., Cau, A., Siewe, F. and Zedan, H. (2013). Dynamic access control policies. *Comput J*, 56(4), pp. 440–463.

[67] Jouault, F., Allilaire, F., Bézivin, J. and Kurtev, I. (2008). ATL: A model transformation tool. *Sci Comput Program*, 72(1-2), pp. 31–39.

[68] JTC1/SC7, I. (2011). ISO/IEC 20000-1:2011 Service management system requirements.

[69] Kalam, A. A. E., Benferhat, S., Miège, A., Baida, R. E., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y. and Trouessin, G. (2003). Organization based access control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, IEEE Computer Society, POLICY '03, pp. 120–131.

[70] Kandala, S., Sandhu, R. and Bhamidipati, V. (2011). An attribute based framework for risk-adaptive access control models. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, IEEE, pp. 236–241.

[71] Kandias, M., Mylonas, A., Virvilis, N., Theoharidou, M. and Gritzalis, D. (2010). An insider threat prediction model. In *Trust, Privacy and Security in Digital Business*, Springer, pp. 26–37.

[72] Kc, G. S., Keromytis, A. D. and Prevelakis, V. (2003). Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, New York, NY, USA: ACM, CCS '03, pp. 272–280.

[73] Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1), pp. 41–50.

[74] Kephart, J. O. and Das, R. (2007). Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1), pp. 40–48.

[75] Klein, M. H. (1983). Department of Defense Trusted Computer System Evaluation Criteria.

[76] Koutsonikola, V. and Vakali, A. (2004). LDAP: Framework, practices, and trends. *IEEE Internet Computing*, 8(5), pp. 66–72.

[77] Kramer, J. and Magee, J. (2007). Self-managed systems: An architectural challenge. In *2007 Future of Software Engineering*, Washington, DC, USA: IEEE Computer Society, FOSE '07, pp. 259–268.

[78] Kurtev, I., Bézivin, J. and Akşit, M. (2002). Technological spaces: An initial appraisal. *Computing & Control Engineering Journal*.

[79] Legg, P. A., Moffat, N., Nurse, J. R., Happa, J., Agrafiotis, I., Goldsmith, M. and Creese, S. (2013). Towards a conceptual model and reasoning structure for insider threat detection. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 4(4), pp. 20–37.

[80] Linn, C. and Debray, S. (2003). Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, New York, NY, USA: ACM, CCS '03, pp. 290–299.

[81] Lopez, J., Oppliger, R. and Pernul, G. (2004). Authentication and authorization infrastructures (aais): A comparative survey. *Comput Secur*, 23(7), pp. 578–590.

[82] Maes, P. (1987). Concepts and experiments in computational reflection. *SIGPLAN Not*, 22(12), pp. 147–155.

[83] McGraw, R. (2009). Risk-adaptable access control (RADac). Tech. rep., National Institute of Standards and Technology (NIST).

[84] McGuinness, D. L., Van Harmelen, F. et al. (2004). Owl web ontology language overview. *W3C recommendation*, 10(10), p. 2004.

[85] Mens, T. and Van Gorp, P. (2006). A taxonomy of model transformation. *Electron Notes Theor Comput Sci*, 152, pp. 125–142.

[86] Mohri, M., Rostamizadeh, A. and Talwalkar, A. (2012). *Foundations of machine learning*. MIT Press.

[87] Montrieux, L. (2013). *Model-Based Analysis of Role-Based Access Control*. Ph.D. thesis, The Open University.

[88] Montrieux, L. (2014). rbacDSML tool [Online]. Available from: http://computing-research.open.ac.uk/rbac/ [Accessed 5 January 2014].

[89] Moore, A. P., Cappelli, D., Caron, T. C., Shaw, E. D., Spooner, D. and Trzeciak, R. F. (2011). A preliminary model of insider theft of intellectual property. Tech. Rep. CMU/SEI-2011-TN-013, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[90] Moore, A. P., Cappelli, D. M. and Trzeciak, R. F. (2008). *The big picture of insider IT sabotage across US critical infrastructures*. Springer.

[91] Moore, A. P., Ellison, R. J. and Linger, R. C. (2001). Attack modeling for information security and survivability. Tech. Rep. CMU/SEI-2001-TN-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[92] Moore, A. P., Hanley, M. and Mundie, D. (2012). A pattern for increased monitoring for intellectual property theft by departing insiders. Tech. Rep. CMU/SEI-2012-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[93] Morgan, R. L., Cantor, S., Carmody, S., Hoehn, W. and Klingenstein, K. (2004). Federated security: The Shibboleth approach. *EDUCAUSE Quarterly*, 27(4), pp. 12–17.

[94] Morin, B., Mouelhi, T., Fleurey, F., Le Traon, Y., Barais, O. and Jézéquel, J.-M. (2010). Security-driven model-based dynamic adaptation. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, New York, NY, USA: ACM, ASE '10, pp. 205–214.

[95] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, ACM, pp. 530–535.

[96] Mu, C. and Li, Y. (2010). An intrusion response decision-making model based on hierarchical task network planning. *Expert systems with applications*, 37(3), pp. 2465–2472.

[97] NIST (2004). INCITS 359-2004 - Role Based Access Control.

[98] Nurse, J. R., Buckley, O., Legg, P. A., Goldsmith, M., Creese, S., Wright, G. R. and Whitty, M. (2014). Understanding insider threat: A framework for characterising attacks. In *Workshop on Research for Insider Threat (WRIT) held as part of the IEEE Computer Society Security and Privacy Workshops (SPW14), in conjunction with the IEEE Symposium on Security and Privacy (SP)*, IEEE, pp. 214–228.

[99] OASIS (2005). Security Assertion Markup Language (SAML) Version 2.0.

[100] OASIS (2011). XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile. Tech. rep., OASIS.

[101] OASIS (2013). eXtensible Access Control Markup Language (XACML) v3.0.

[102] O'Conner, A. C. and Loomis, R. J. (2010). 2010 economic analysis of role-based access control. Tech. rep., RTI International, NIST.

[103] Ogata, K. (1990). *Modern Control Engineering*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd edn.

[104] Oltsik, J. (2013). The 2013 Vormetric insider threat report [Online]. Available from: http://www.vormetric.com/sites/default/files/vormetric-insider-threat-report-oct-2013.pdf [Accessed 12 June 2014].

[105] OMG (2006). Meta Object Facility (MOF) 2.0.

[106] OMG (2010). Unified Modeling Language (UML) 2.3.

[107] Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S. and Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3), pp. 54–62.

[108] Palomo, I. (2001). Serious security hole in Mambo site server version 3.0.X [Online]. Available from: http://seclists.org/bugtraq/2001/Jul/569 [Accessed 17 June 2014].

[109] Park, J. and Sandhu, R. (2004). The UCONABC usage control model. *ACM Trans Inf Syst Secur*, 7(1), pp. 128–174.

[110] Pashalidis, A. and Mitchell, C. J. (2003). A taxonomy of single sign-on systems. In *Proceedings of the 8th Australasian Conference on Information Security and Privacy*, Berlin, Heidelberg: Springer-Verlag, ACISP'03, pp. 249–264.

[111] Pasquale, L., Menghi, C., Salehie, M., Cavallaro, L., Omoronyia, I. and Nuseibeh, B. (2012). Securitas: A tool for engineering adaptive security. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, New York, NY, USA: ACM, FSE '12, pp. 19:1–19:4.

[112] Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S. (2002). A community authorization service for group collaboration. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, Washington, DC, USA: IEEE Computer Society, POLICY '02, pp. 50–59.

[113] Perkins, J. H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., Pacheco, C., Sherwood, F., Sidiroglou, S., Sullivan, G., Wong, W.-F., Zibin, Y., Ernst, M. D. and Rinard, M. (2009). Automatically patching errors in deployed software. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, New York, NY, USA: ACM, SOSP '09, pp. 87–102.

[114] PERMIS Standalone Authorisation Server (n.d.). [Online]. Available from: http://sec.cs.kent.ac.uk/permis/ [Accessed 5 January 2014].

[115] Perrin, C. (2008). The CIA triad. Available from: http://www.techrepublic.com/blog/security/the-cia-triad/488 [Accessed 03 Sep 2015.

[116] Pritchard, D. B. (1994). *'Snakes and Ladders', The Family Book of Games*. Brockhampton Press, 1st edn.

[117] Ratha, N. K., Bolle, R. M., Pandit, V. D. and Vaish, V. (2000). Robust fingerprint authentication using local structural similarity. In *Applications of Computer Vision, 2000, Fifth IEEE Workshop on.*, IEEE, pp. 29–34.

[118] Ray, I. and Poolsapassit, N. (2005). Using attack trees to identify malicious attacks from authorized insiders. In *Proceedings of the 10th European Conference on Research in Computer Security*, *Lecture Notes in Computer Science*, vol. 3679, Berlin, Heidelberg: Springer-Verlag, pp. 231–246.

[119] Robertson, P. and Laddaga, R. (2005). Model based diagnosis and contexts in self adaptive software. In *Self-star Properties in Complex Information Systems*, *Lecture Notes in Computer Science*, vol. 3460, Springer, pp. 112–127.

[120] Roesch, M. et al. (1999). Snort: Lightweight intrusion detection for networks. In *LISA*, vol. 99, pp. 229–238.

[121] Salehie, M. and Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Trans Auton Adapt Syst*, 4(2), pp. 14:1–14:42.

[122] Sandhu, R. (2012). The authorization leap from rights to attributes: maturation or chaos? In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, ACM, SACMAT '12, pp. 69–70.

[123] Sandhu, R. S., Coyne, E. J., Feinstein, H. L. and Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2), pp. 38–47.

[124] Schneider, F. B. (2000). Enforceable security policies. *ACM Trans Inf Syst Secur*, 3(1), pp. 30–50.

[125] Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5), pp. 26–32.

[126] Sendall, S. and Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5), pp. 42–45.

[127] Serrano, M., Meer, S., Strassner, J., Paoli, S., Kerr, A. and Storni, C. (2009). Trust and reputation policy-based mechanisms for self-protection in autonomic communications. In *Proceedings of the 6th International Conference on Autonomic and Trusted Computing*, Berlin, Heidelberg: Springer-Verlag, ATC '09, pp. 249–267.

[128] Sharma, A., Kalbarczyk, Z., Iyer, R. and Barlow, J. (2010). Analysis of credential stealing attacks in an open networked environment. In *Proceedings of the 2010 Fourth International Conference on Network and System Security*, Washington, DC, USA: IEEE Computer Society, NSS '10, pp. 144–151.

[129] Shi, L. and Chadwick, D. W. (2011). A controlled natural language interface for authoring access control policies. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, New York, NY, USA: ACM, SAC '11, pp. 1524–1530.

[130] Shirey, R. (2007). Internet RFC 4949.

[131] SimpleSAMLphp (n.d.). [Online]. Available from: http://simplesamlphp.org/ [Accessed 5 January 2014].

[132] Spitzner, L. (2003). Honeypots: Catching the insider threat. In *Proceedings of the 19th Annual Computer Security Applications Conference*, IEEE, pp. 170–179.

[133] Spivey, J. M. (1989). *The Z Notation: A Reference Manual.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

[134] Stakhanova, N., Basu, S. and Wong, J. (2007). A cost-sensitive model for preemptive intrusion response systems. In *AINA*, vol. 7, pp. 428–435.

[135] Stakhanova, N., Basu, S. and Wong, J. (2007). A taxonomy of intrusion response systems. *Int J Inf Comput Secur*, 1(1/2), pp. 169–184.

[136] Strasburg, C., Stakhanova, N., Basu, S. and Wong, J. S. (2009). A framework for cost sensitive assessment of intrusion response selection. In *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01*, Washington, DC, USA: IEEE Computer Society, COMPSAC '09, pp. 355–360.

[137] Tamura, G., Villegas, N. M., Müller, H. A., Duchien, L. and Seinturier, L. (2013). Improving context-awareness in self-adaptation using the dynamico reference model. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Piscataway, NJ, USA: IEEE Press, SEAMS '13, pp. 153–162.

[138] Team, T. C. I. T. (2013). Unintentional insider threats: A foundational study. Software Engineering Institute Technical Report.

[139] Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K. and Essiari, A. (1999). Certificate-based access control for widely distributed resources. In *Proceedings of the 8th Conference on USENIX Security Symposium*, Berkeley, CA, USA: USENIX Association, SSYM'99, pp. 17–30.

[140] UK Access Management Federation (n.d.). [Online]. Available from: http://www.ukfederation.org.uk/ [Accessed 17 January 2014].

[141] Vijayan, J. (2009). Former DuPont researcher hit with federal data theft charges [Online]. Available from: http://www.computerworld.com/s/article/9139014/Former_DuPont_researcher_hit_with_federal_data_theft_charges [Accessed 10 May 2014].

[142] Villegas, N. M., Müller, H. A., Tamura, G., Duchien, L. and Casallas, R. (2011). A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, New York, NY, USA: ACM, SEAMS '11, pp. 80–89.

[143] Walsh, C. (2006). New data theft scandal rocks subcontinent's call centres [Online]. Available from: http://www.theguardian.com/money/2006/sep/03/business.india [Accessed 5 January 2014].

[144] Wang, K. and Stolfo, S. (2004). Anomalous payload-based network intrusion detection. In E. Jonsson, A. Valdes and M. Almgren, eds., *Recent Advances in Intrusion Detection*, *Lecture Notes in Computer Science*, vol. 3224, Springer Berlin Heidelberg, pp. 203–222.

[145] Warmer, J. B. and Kleppe, A. G. (2003). *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional.

[146] Weyns, D., Malek, S. and Andersson, J. (2010). Forms: A formal reference model for self-adaptation. In *Proceedings of the 7th International Conference on Autonomic Computing*, New York, NY, USA: ACM, ICAC '10, pp. 205–214.

[147] Weyns, D., Malek, S. and Andersson, J. (2012). Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Trans Auton Adapt Syst*, 7(1), pp. 8:1–8:61.

[148] XACMLight (n.d.). [Online]. Available from: http://xacmllight.sourceforge.net/ [Accessed 17 January 2014].

[149] Yuan, E., Esfahani, N. and Malek, S. (2014). A systematic survey of self-protecting software systems. *ACM Trans Auton Adapt Syst*, 8(4), pp. 17:1–17:41.

[150] Yuan, E. and Malek, S. (2012). A taxonomy and survey of self-protecting software systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, IEEE, pp. 109–118.

[151] Yuan, E., Malek, S., Schmerl, B., Garlan, D. and Gennari, J. (2013). Architecture-based self-protecting software systems. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, ACM, pp. 33–42.

[152] Yuan, E. and Tong, J. (2005). Attributed based access control (ABAC) for web services. In *Proceedings of the IEEE International Conference on Web Services*, Washington, DC, USA: IEEE Computer Society, ICWS '05, pp. 561–569.

[153] Zhang, N., Ryan, M. and Guelev, D. P. (2008). Synthesising verified access control systems through model checking. *J Comput Secur*, 16(1), pp. 1–61.