Evaluating Self-Adaptive Authorisation Infrastructures through Gamification

(Regular Paper)

***Abstract*—Self-adaptive systems are able to modify their behaviour and/or structure in response to changes that occur to the system itself, its environment, or even its goals. In terms of authorisation infrastructures, self-adaptation has been shown to provide runtime capabilities for specifying and enforcing access control policies and subject access privileges, with a goal to mitigate insider threat. The evaluation of self-adaptive authorisation infrastructures, particularly, in the context of insider threats, is challenging because simulation of malicious behaviour can only demonstrate a fraction of the types of abuse that is representative of the real-world. In this paper, we present an innovative approach based on an ethical game of hacking, protected by an authorisation infrastructure. A key feature of the approach is the ability to observe user activity pre- and post-adaptation when evaluating runtime consequences of self-adaptation. Our live experiments captured a wide range of unpredictable changes, including malicious behaviour related to the exploitation of known vulnerabilities. As an outcome, we demonstrated the ability of our self-adaptive authorisation infrastructure to handle malicious behaviour given the existence of real and intelligent users, in addition to capturing how users responded to adaptation.**

***Index Terms*—self-adaptive systems, authorisation infrastructures, insider threats, gamification**

I. Introduction

Self-adaptive systems are able to modify their behaviour and/or structure in response to changes that occur to the system itself, its environment, or even its goals [5]. Based on this, a self-adaptive authorisation infrastructure refers to the run-time adaptation of access control policies and their enforcement.

An important aspect when evaluating a self-adaptive authorisation infrastructure is to demonstrate its ability to mitigate abuse of access when faced with uncertainty. The simulation of insider threat scenarios is limited because they would not be able to portray an accurate perception of reality. Moreover, when considering self-adaptive authorisation infrastructures the usage of existing data has little value because of the dynamic aspects of the infrastructure that continuously adapts itself in response to changes. In terms of insider threats, it is necessary to evaluate the consequence of self-adaptation in terms of how human users respond to automated mitigation. 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.

This paper presents an approach whereby gamification [7] 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. Gamification allows generating diverse and unpredictable data from real user activity. In particular, it enables the observation of 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.

In this paper, we employ the Self-Adaptive Authorisation Framework (SAAF) for evaluating the effectiveness of self-adaptive authorisation infrastructures. The objective of evaluating SAAF in a live deployment is twofold. First, to demonstrate that the observations and actions performed by SAAF have a real consequence to human users accessing a resource. Second, to generate data that portrays the effectiveness of self-adaptation in mitigating observed attacks, including data related to the consequences of self-adaptation. To fulfil these objectives, an experiment was conducted whereby human users were invited to participate in an ethical game of hacking. Users were asked to play an online game based on the classic board game of *Snakes and Ladders* [20]. 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.

The experiments conducted seek to answer the following two research questions: *are self-adaptive authorisation infrastructures capable of mitigating acts of malicious behaviour? What are the consequences of self-adaptation?* To reflect on this problem statement, we identify three key hypotheses:

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

Hypothesis 2. *An experienced subject is capable of carrying out sophisticated and complex attacks.*

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

This evaluation is specific to the mitigation of malicious subject activity related to the abuse of access. However, there are some limitations associated this exercise. 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, though, the game as a whole is representative of the actions of a malicious insider.

The contribution of this paper is an approach to evaluating self-adaptive systems through gamification [7]. 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. The effectiveness of Self-Adaptive Authorisation Framework (SAAF) is evaluated by way of deploying an online game as a protected resource within an authorisation infrastructure.

The rest of this paper is structured as follows. In Section II, we present some basic concepts related to self-adaptive authorisation infrastructures and insider threats. Section III describes the design of an online game in which diverse and unpredictable behaviour can be observed. Section IV discusses the deployment of the game in a self-adaptive authorisation infrastructure. Section V describes the phases and execution of the experiment within the game environment, and discusses the results of the experiments. In Section VIII, a summary of the paper is provided in addition to some insights regarding future work.

II. Background

A. Self-adaptive Authorisation Framework

The goal of Self-adaptive Authorisation Framework (SAAF) is to make existing authorisation infrastructures self-adaptable, where an organisation can benefit from the properties of dynamic access control without the need to adopt new access control models [1], [2]. SAAF is based on the MAPE-K [11] feedback loop, which monitors the distributed services of an authorisation infrastructure to build a modelled state of access at runtime (i.e., deployed access control rules, assigned subject privileges, and protected resources). Malicious user behaviour observed by a SAAF controller is mitigated through the generation and deployment of access control policies at runtime, preventing any identified abuse from continuing. Adaptation at the model layer enables assurances and verification that abuse can no longer continue. In addition, model transformation has been shown to generate access control policies from an abstract model of access. This has the potential to enable the generation of policies specific to many different implementations of access control.

Figure 1 presents a conceptual view of SAAF in which an autonomic controller monitors and adapts multiple systems within an authorisation infrastructure. This presents a challenge since no single system provides a complete view of access in terms of what users own in access rights, what access control rules exist, and finally, how users are utilising access rights. In its current form, SAAF ensures that whatever adaptations take place will not break conformance to the service's implemented access control methodology (ABAC), nor conflict with application domain requirements (e.g., ensure access to business critical systems). To implement ABAC, we provide an identity service referred to as LDAP [12], which is a directory service commonly used to hold information

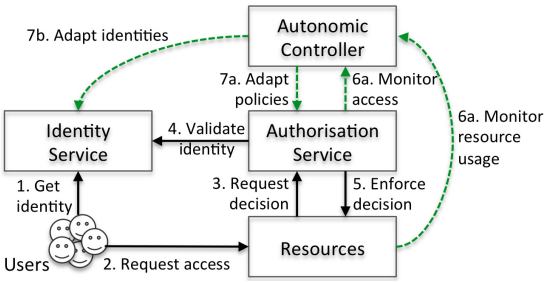

Fig. 1. SAAF conceptual design

(including user roles) about users within an organisation, and a standalone service authorisation service, known as PERMIS [4], used to generate ABAC access control decisions based on roles owned by users.

B. Insider Threats

Insider threat refers to an organisation's risk of attack by their own users or employees. This is particularly relevant to access control, where the active management of authorisation has the potential to mitigate and prevent users from abusing their own access rights to carry out attacks.

A common characteristic of insider threat is that malicious insiders utilise their knowledge of their organisation's systems, and their assigned access rights, to conduct 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 [3]. Such form of attack is 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 [17]. Unless additional measures are put into place, malicious insiders can abuse existing security measures, where current approaches fail to robustly adapt and respond to the unpredictable nature of users. Whilst there are a number of novel techniques that enable the detection of insider threat [8], [16], [21], there is little research that utilises such techniques within an automated setting.

III. 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.

Whilst Snakes and Ladders presents a narrower scope in the type of malicious behaviour 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.

A. 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. Figure 2 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.

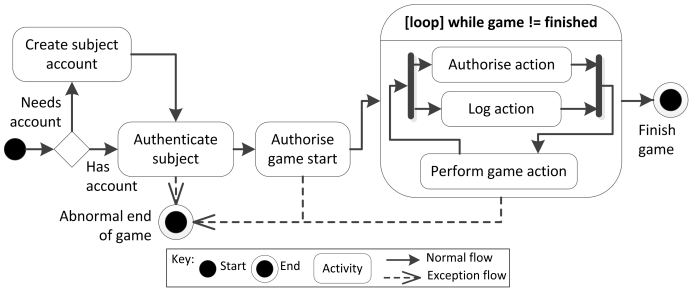

Fig. 2. Activity flow of the game resource

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 3) 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 action. An authorisation policy is expected to define the criteria of access, and should protect access against the following actions: start game, roll dice, move player, use ladder, end game, and use bonus (an added feature to the original game, which moves the player towards the end of the game).

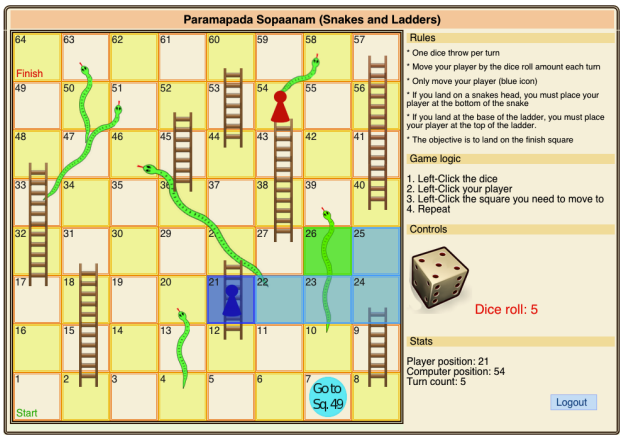

Fig. 3. Screenshot of the implemented Snakes and Ladders game

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.

B. Vulnerabilities

The design of the game facilitates players performing malicious activities through exploiting known and unknown vulnerabilities. The game itself is considered a honeypot [21], 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.

1) Game Interface Vulnerabilities: These symbolise the simplest form of attack, whereby subjects identify bugs within the game logic simply through interaction with the game itself. For example, the dice can be rolled multiple times, or the player can land on any square within the given dice role range.

2) Code Injection Vulnerabilities : Code injection [10] 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 the use of obfuscated [13] 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.

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.

3) Session Vulnerabilities: Session poisoning involves attacks where a client injects data into a session held by a server [18]. 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.

4) Summary Attack Model: Given the described known vulnerabilities, abuse of access can be modelled as a high level attack tree [15]. Listing 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.

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

Listing 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. These types of attacks, whilst worth investigating in future work, remain out of scope of this evaluation.

C. 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.

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 bypass the resource's policy enforcement point (PEP). As a result, any games that bypass authorisation are out of scope of the evaluation.

IV. 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.

A. Self-Adaptive Authorisation Infrastructure

The infrastructure is comprised of three virtual machines (VMs), as shown in Figure 4, with each VM is configured to run Ubuntu v12.04.5 TLS, with 1024MB RAM.

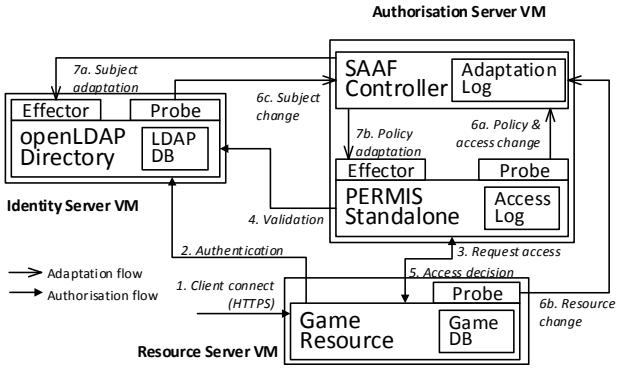

Fig. 4. Game experiment authorisation infrastructure

1) Identity Server: The Identity Server 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. 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.

2) Authorisation Server: The Authorisation Server VM hosts an instance of the PERMIS standalone authorisation service [4], a probe and effector to monitor and adapt PERMIS, and the SAAF controller.

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.

3) Resource Server: The Resource Server 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.

B. SAAF Controller Configuration

The SAAF controller is configured to maintain (at runtime) a synchronised model of access within the authorisation infrastructure (Figure 4). 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.

1) 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 [9], or updates its behaviour model to reflect player authorisation and resource activity.

2) 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.

3) 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.

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).

4) Execution: 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.

C. Logs

Considering the deployment of the game and SAAF, data is logged in regards to the following perspectives.

1) Game: Player activity is logged by the game resource, which is interpreted within its backend database. 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.

Player 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.

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.

2) 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.

3) 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.

4) 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.

V. Experiments

This section describes the experiments performed within the game environment, conveying data that demonstrates the SAAF controller monitoring and responding to malicious behaviour.

A. 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. 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).

- **Control** - The game is released to a closed set of participants to observe honest play, for validation of detectors. It 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. It 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;
- **Phase 1** - The game is released within the School of Computing, University of Kent, requesting participants to play the game honestly or dishonestly. It 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;
- **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. It 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;
- **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. It 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.

In each phase, participants were provided the same guidance in the form of a participant declaration that described the purpose of the experiment, and a brief overview of how the game works. 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.

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.

B. Observed Environment Change

The following section discusses two aspects of the observed environment change, namely game statistics and trends in player activity.

1) Game Statistics: Over the course of the experiment phases, 1455 games were played and 366 game accounts were created (Table I). 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.

TABLE I
HIGH LEVEL STATISTICS OF GAME RELATED DATA

	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

Of particular importance, was the observation of diverse player interaction. In this instance, out of the 1455 games played, 939 games were unique. 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). The 757 unique game actions observed indicate 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 II). The high number of failed authentications is largely due to a number of (unsuccessful) attacks against the game's account login page.

TABLE II
AUTHENTICATION AND AUTHORISATION STATISTICS

	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

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. This is evidence of users bypassing authorisation within the game resource.

2) Player Behaviour: A high level analysis of player has 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.

C. Detection and Mitigation

Over the course of experiment phases 1 to 3, 1246 violations were detected (Table III). Out of the violations detected, 1203 violations triggered a resultant mitigation, whereby a solution was enacted by the SAAF controller.

TABLE III
VIOLATION STATISTICS

	P1	P2	P3	Total
Violations detected	228	738	280	1246
Violations mitigated	219	717	267	1203
Mitigation failures	9	21	13	43

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. 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.

1) Violations: Figure 5 conveys the percentage of violation types that were detected in phases 1 to 3 (see Section IV-B). Violations `rollMove` and `illegalMove` represent the most common violations.

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

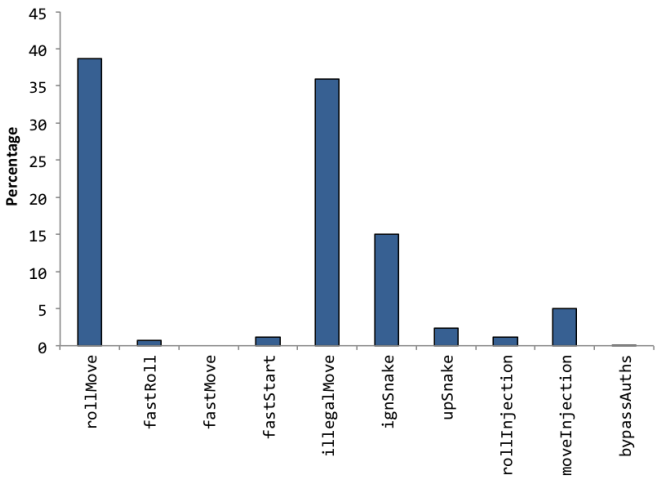

Fig. 5. Percentage of detected violations by type

more obvious to commit (sharing similar characteristics to `illegalMove`). In addition, more sophisticated violations, such as `rollInjection` and `moveInjection` were seen to be rare.

2) Mitigations: Regarding 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.

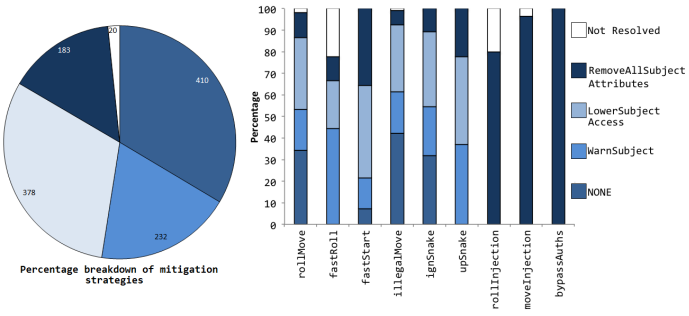

Fig. 6. Breakdown of violations and mitigations

Figure 6 portrays 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.

3) 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.

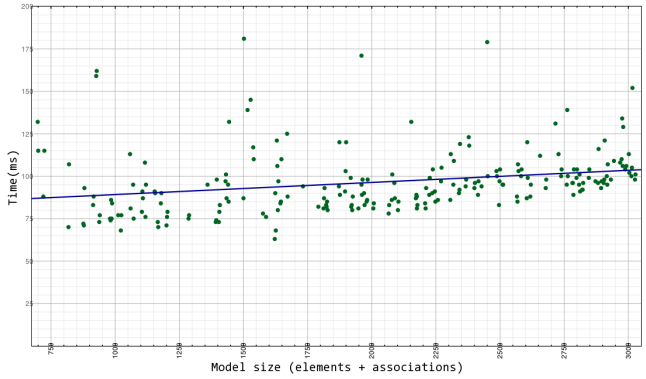

Fig. 7. Total mitigation time versus model size

Figure 7 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.

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.

VI. Evaluation

In this section we demonstrate the hypotheses proposed in Section I by analysing a set of attacks, and discussing dependent variables relevant to each hypothesis.

A. Hypothesis 1 - Self-adaptation mitigates malicious activity

To demonstrate this hypothesis, the following exemplifies three different attack profiles that were observed throughout the experiment phases.

1) 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. 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.

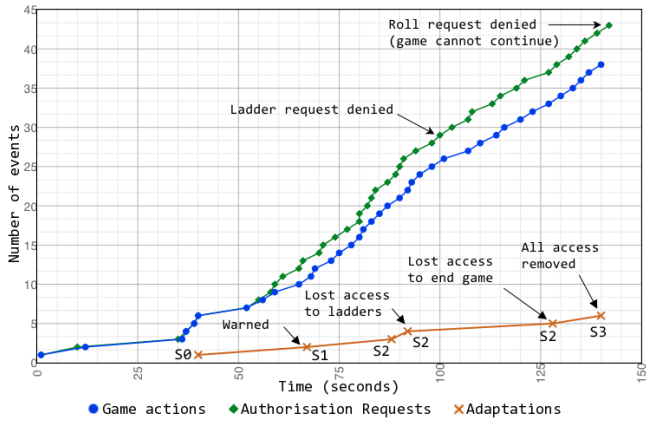

Fig. 8. Trace of a persistent weak violation profile against mitigation

Figure 8 portrays the player's changes, over time, 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 IV.

TABLE IV
ADAPTATION TRACE OF AN PERSISTENT WEAK ATTACK GAME

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

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 140 seconds into the game, after observing 6 low level violations, the SAAF controller removes all access from the subject (S3).

2) 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.

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.

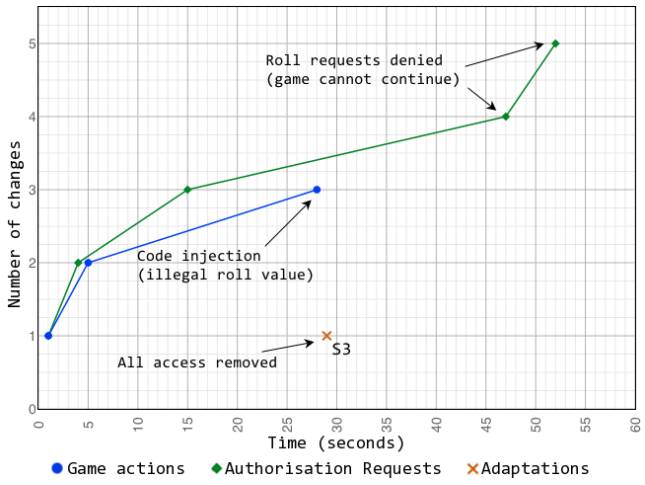

Fig. 9. Trace of an immediate high severity violation against mitigation

TABLE V
ADAPTATION TRACE OF AN IMMEDIATE STRONG ATTACK GAME

Step	Game Time (s)	Violation	Enacted Solution	Time (ms)
1	29	rollInjection	removeAllSubjectAttributes (S3)	128

Figure 9 and Table V 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, resulting in the SAAF controller removing 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.

B. Hypothesis 2 - Experience subject is able to carry sophisticated attacks

A small percentage of violations detected were sophisticated attacks (Figure 5), such as code injection attacks. It was hypothesised that experience enables participants to perform sophisticated attacks within a game.

To demonstrate this, Figure 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

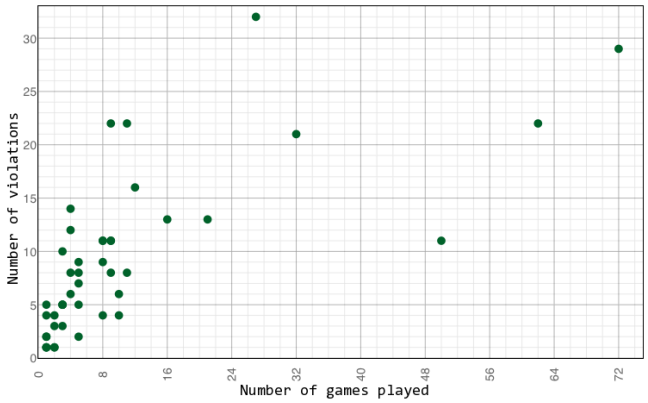

Fig. 10. Violations and games played prior to injection attacks

attack) that have played x amount of games, and committed y amount of violations.

Only 11 participants (based on devices 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.

C. Hypothesis 3 - 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 VI
CHANGES IN PARTICIPANT BEHAVIOUR POST-WARNING

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 VI identifies four types of changes in behaviour observed after a player's 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.

TABLE VII
CHANGES IN PARTICIPANT BEHAVIOUR POST-DENY OF ACCESS

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 VII 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).

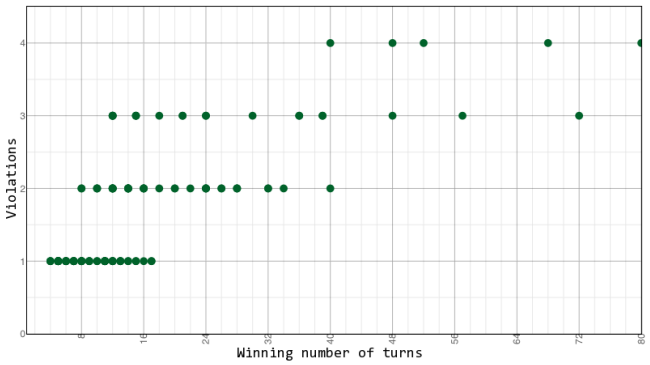

Fig. 11. Number of detected violations versus number of turns to win a game

Figure 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.

D. Summary

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 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 experiments have provided evidence that malicious subjects were no longer capable in gaining access, this hypothesis has been confirmed. However, one exception is that adaptation has only been shown to succeed when faced with known violations.

Hypothesis 2 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 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 self-adaptation. Whilst many players were statistically seen to persist with the same behaviour, despite self-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 self-adaptation, had tactfully performed violations to a point where they did not lose complete access.

It is worth noting that the evidence to demonstrate **hypothesis 2** and **hypothesis 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.

A lesson learned from using gamification to evaluate novel approach for protecting a system against insider threats is that gamification was quite effective in identifying vulnerabilities that other conventional evaluation techniques would not be able to identify. Evidence of this is that by fixing the identified vulnerabilities in the earlier phases of evaluation lead to a more robust SAAF. During the phase 3 of the experiment, the only attack not identified was when a player accounts were falsified without any attempt of playing the game. This was done by manipulating the endpoints of the AJAX routines that handled logging and policy enforcement for mimicking player activity.

VII. Related Work

Self-protecting systems are a specialisation of self-adaptive systems with a goal to mitigating malicious behaviour, and SAAF can be considered a self-protecting system. In the following, we discuss the few works that have demonstrated self-protection within the context of mitigating insider attacks. In particular, we discuss two self-protection approaches based on the state of access control, and one approach based on the state system architecture.

One of the approaches to self-protection via access control is SecuriTAS [19]. SecuriTAS is a tool that enables dynamic decisions in awarding access, which is based on a perceived state of the system and its environment. SecuriTAS is similar to dynamic access control approaches, such as RADac [14], in that it has a notion of risk (threat) to resources, and changes in threat leads to a change in access control decisions. However, it furthers the concepts in RADac to include the notion of utility. The main difference between SecuriTAS and SAAF, is that SecuriTAS positions its own bespoke access control model and authorisation infrastructure that incorporates self-adaptation by design. SAAF, on the other hand, is a framework that describes how existing access control models and authorisation infrastructures can be made self-adaptive, and as such, configured to actively mitigate insider threat. With that said, both approaches demonstrate an authorisation infrastructure's robustness in mitigating insider attacks, by ensuring that authorisation remains relevant to system and environment states (and preventing continuation of attacks by adaptation of security controls).

In contrast to self-protection via access control, architectural-based self-protection (ABSP) [22] presents a general solution to detection and mitigation of security threats, via runtime structural adaptation. Rather than reason at the contextual layer of 'access control', ABSP utilises an architectural model of the running system to identify the extent of impact of identified attacks. Once attacks or security threats have been assessed, a self-adaptive architectural

manager (Rainbow [6]) is used to perform adaptations to mitigate the attack. ABSP shares a number of similarities with intrusion response and prevention systems, particularly with the scope of adaptations that ABSP can perform (e.g., structural adaptation against network devices and connections). However, because ABSP maintains a notion of 'self', it is able to reason about the impact of adaptations and provide assurance over adaptation before adapting its target system.

VIII. Conclusions

This paper has demonstrated gamification as a viable approach for the evaluation of self-adaptive authorisation infrastructures. 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 malicious behaviour representative of intelligent user behaviour, pre- and post-adaptation.

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 malicious behaviours related to the exploitation 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 paper 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, i.e., attacks models need to be reevaluated.

References

[1] Self-adaptive federated authorization infrastructures. *Journal of Computer and System Sciences*, 80(5):935–952, 2014.

[2] C. Bailey, D. W. Chadwick, and R. de Lemos. 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*, DASC '11, pages 37–44, Washington, DC, USA, 2011. IEEE Computer Society.

[3] D. Caputo, M. Maloof, and G. Stephens. Detecting insider theft of trade secrets. *IEEE Security and Privacy*, 7(6):14–21, Nov. 2009.

[4] D. W. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, and T. A. Nguyen. PERMIS: A modular authorization infrastructure. *Concurr. Comput. : Pract. Exper.*, 20(11):1341–1357, Aug. 2008.

[5] R. de Lemos and et al. Software engineering for self-adaptive systems: A second research roadmap. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013.

[6] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, Oct. 2004.

[7] K. Huotari and J. Hamari. Defining gamification: A service marketing perspective. In *Proceeding of the 16th International Academic MindTrek Conference*, MindTrek '12, pages 17–22, New York, NY, USA, 2012. ACM.

[8] IBM. IBM Security Intelligence with Big Data [Online], n.d. Available from: http://www-03.ibm.com/security/solution/intelligence-big-data/ [Accessed 20 July 2014].

[9] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, June 2008.

[10] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 272–280, New York, NY, USA, 2003. ACM.

[11] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[12] V. Koutsonikola and A. Vakali. LDAP: Framework, practices, and trends. *IEEE Internet Computing*, 8(5):66–72, Sept. 2004.

[13] C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 290–299, New York, NY, USA, 2003. ACM.

[14] R. McGraw. Risk-adaptable access control (RADac). Technical report, National Institute of Standards and Technology (NIST), 2009.

[15] A. P. Moore, R. J. Ellison, and R. C. Linger. Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2001.

[16] J. R. Nurse, O. Buckley, P. A. Legg, M. Goldsmith, S. Creese, G. R. Wright, and M. Whitty. 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)*, pages 214–228. IEEE, 2014.

[17] J. Oltsik. The 2013 Vormetric insider threat report [Online], 2013. Available from: http://www.vormetric.com/sites/default/files/vormetric-insider-threat-report-oct-2013.pdf [Accessed 12 June 2014].

[18] I. Palomo. Serious security hole in Mambo site server version 3.0.X [Online], 2001. Available from: http://seclists.org/bugtraq/2001/Jul/569 [Accessed 17 June 2014].

[19] L. Pasquale, C. Menghi, M. Salehie, L. Cavallaro, I. Omoronyia, and B. Nuseibeh. Securitas: A tool for engineering adaptive security. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 19:1–19:4, New York, NY, USA, 2012. ACM.

[20] D. B. Pritchard. *'Snakes and Ladders', The Family Book of Games*. Brockhampton Press, 1st edition, 1994.

[21] L. Spitzner. Honeypots: Catching the insider threat. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 170–179. IEEE, 2003.

[22] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari. Architecture-based self-protecting software systems. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 33–42. ACM, 2013.