NANE: Identifying Misuse Cases Using Temporal Norm Enactments

Özgür Kafalı, Munindar P. Singh, and Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
{rkafali,singh,laurie_williams}@ncsu.edu

***Abstract*—Recent data breaches in domains such as healthcare where confidentiality of data is crucial indicate that breaches often originate from misuses, not only from vulnerabilities in the technical (software or hardware) architecture. Current requirements engineering (RE) approaches determine what access control mechanisms are needed to protect sensitive resources (assets). However, current RE approaches inadequately characterize how a user is expected to interact with others in relation to the relevant assets. Consequently, a requirements analyst cannot readily identify misuses by legitimate users. We adopt social norms as a natural, formal means of characterizing user interactions whereby potential misuses map to norm violations. *Our research goal is to help analysts identify misuse cases by formal reasoning about norm enactments.* We propose NANE, a formal framework for identifying such misuse cases using a semiautomated process. We demonstrate how NANE enables monitoring of potential misuses on a healthcare scenario.**

***Index Terms*—Security requirements, sociotechnical systems**

I. INTRODUCTION

Data breaches pose a major threat to stakeholders of modern software systems. Healthcare IT systems are no exception, and millions of patients have suffered from compromised health records in the recent years [22], [26]. Although some breaches arise due to vulnerabilities in the software, an increasing number of them are caused by misuse. For example, a failure to erase patient data contained on photocopiers' hard drives led to the disclosure of 350,000 patient files, and resulted in a fine of $1.2 million from the US Department of Health and Human Services (HHS). Example 1 describes a typical misuse.

Example 1. *A physician, Alice, and a nurse, Don, are reviewing the electronic health records (EHR) of a patient together on Alice's computer. Each of them is authorized to review EHRs of patients they are treating, and prohibited from reviewing the EHRs of other patients. Alice receives a call and leaves the room without ending her EHR session. Don knows that Alice is treating one of his neighbors. He accesses his neighbor's EHR using Alice's session.*

The leading current approach for implementing security requirements is role-based access control (RBAC) [23], which provides an effective way of protecting access to sensitive resources (assets). In Example 1, in regular (nonemergency) practice, the hospital software does not allow any staff member to access the EHR of a patient he or she is not treating. However, Don circumvents these technical controls by using Alice's computer. That is, a breach occurs here because Alice fails to end her EHR session, thereby inadvertently giving (otherwise prohibited) access to Don.

This example illustrates the key limitation of RBAC: it is impossible to prevent misuse because it occurs outside of the technical realm. The only way to tackle misuse is through social mechanisms, specifically, by making users accountable for correct use. For example, Alice should be accountable for ending her EHR session when her computer is unattended.

Logging is the established computational means to support accountability of users. We define an *event* as a discrete occurrence such as a user action (e.g., a physician accessing a patient's EHR) or an environment condition (e.g., a medical emergency) becoming true or false. Logging seeks to record each relevant event and its time of occurrence. However, current logging approaches suffer from two shortcomings. First, they do not guarantee that all events necessary to assess misuse are logged, thereby inadequately supporting accountability. Second, they do not guarantee that only events necessary to assess misuse are logged, thereby creating avoidable vulnerabilities through the log data itself [1]. These shortcomings arise because of a lack of a suitable formal model based on which to log events.

Accordingly, we propose NANE (from the Turkish slang for shenanigans), a framework for identifying misuse cases. NANE is based on a sociotechnical conception [29] that brings together technical considerations such as how users access assets via RBAC and social considerations such as the norms that characterize users' expectations of each other. We define an *enactment* as a possible history of the system, including all events along that history. We adopt a formal model for norms (including conditional commitments, authorizations, and prohibitions, as explained below) that precisely describes (i) the enactments in which each norm may be satisfied or violated, and (ii) who is accountable for the norm. For example, Alice accessing her patient's EHR complies with the authorization and prohibition of Example 1 for which she is accountable.

Our research goal is to help analysts identify misuse cases by formal reasoning about norm enactments.

Usage: A typical NANE episode proceeds as follows. First, a requirements analyst specifies norms based upon stakeholder requirements. Second, NANE applies the formal semantics of norms to generate all possible enactments of each stated norm, differentiating between compliant and violating enactments. For example, Don accessing his neighbor's EHR is a violating enactment of the norm that prohibits Don from accessing EHRs of patients he is not treating. Third, an analyst modifies the stated norms based on a review of these enactments. Fourth, Nane produces formal representations of violating enactments to enable the monitoring of potential misuses.

Contributions: We address two main questions.

RQ$_1$ *Identification:* How can we systematically enumerate the potential misuse cases of a software system?

RQ$_2$ *Monitoring:* How can we formally represent violations of norms so as to enable monitoring of potential misuses?

We give a formal representation of norms and misuse cases in the Event Calculus [20], a first-order logic with primitives for events and temporal reasoning. We provide a semiautomated process for identifying misuse cases. A formal representation of norms enables us to automatically determine which of the possible enactments violate norms (potential misuse). We demonstrate how NANE enables monitoring of potential misuses based on the proposed representation. These contributions differentiate our work from the literature on misuse, as discussed throughout the paper.

Structure: Section II reviews the relevant background. Section III describes temporal norms and their enactments. Section IV describes the details of the NANE framework and how misuse cases are generated. Section V demonstrates how NANE is used for monitoring of misuses. Section VI describes the limitations of our work. Section VII reviews the relevant literature. Section VIII presents future directions.

II. BACKGROUND

We review the necessary background for developing our formalization on misuse cases, and demonstrating how this formalization enables monitoring of potential misuse.

A. Temporal Reasoning

We use first-order logic to represent and reason about misuse cases. Event Calculus (EC) [20] is an extension of first-order logic to interpret and reason about events in time. Table I summarizes the domain-independent axioms of EC.

TABLE I
DOMAIN-INDEPENDENT AXIOMS OF THE EVENT CALCULUS.

Predicate	Description
happens(E, T)	Event E happens at Time T
initially(F)	Fluent F is true at Time 0
holds_at(F, T)	Fluent F is true at Time T
broken(F, Ts, Te)	Fluent F is made false between times Ts and Te
initiates_at(E, F, T)	Event E initiates fluent F at Time T
terminates_at(E, F, T)	Event E terminates fluent F at Time T

The happens predicate records events and the time points at which they occur. The initially predicate specifies fluents that hold at the beginning of time. A fluent is a predicate whose value can be changed in time due to the occurrence of events. The holds_at predicate queries the happened events to check whether a fluent holds at a given time. The broken predicate checks whether a fluent is made false (i.e., terminated) during a time period (Ts and Te are both exclusive). The initiates predicate states that an event makes a fluent true at a given time. The terminates predicate states that an event makes a fluent false at a given time. Briefly, a normative temporal theory in EC consists of the following components:

- the domain-independent axioms of EC (Table I);
- the domain-independent normative theory that describes the norm lifecycle (Section III-A);
- a domain model that describes which events initiate or terminate which fluents in the domain; and
- an enactment given in the form of a list of happens assertions (Listing 1) that describes what has occurred.

Note that the normative theory is relevant only to norm-based EC formalizations like ours. The domain model contains a list of initiates and terminates rules, e.g., the event of a patient leaving a physician's office terminates the fluent regarding patient's visit. An enactment in our formalization corresponds to a *narrative* in the EC literature. Listing 1 demonstrates a sample enactment in EC.

Listing 1
SAMPLE ENACTMENT IN EC.

```
1 happens(give_consent(drBob, john), 1).
2 happens(access_EHR(drBob, john), 4).
3 happens(visit(drBob, john), 5).
4 happens(emergency(hospitalNC), 12).
5 happens(access_EHR(drBob, kate), 13).
6 happens(access_EHR(drAlice, adam), 13).
```

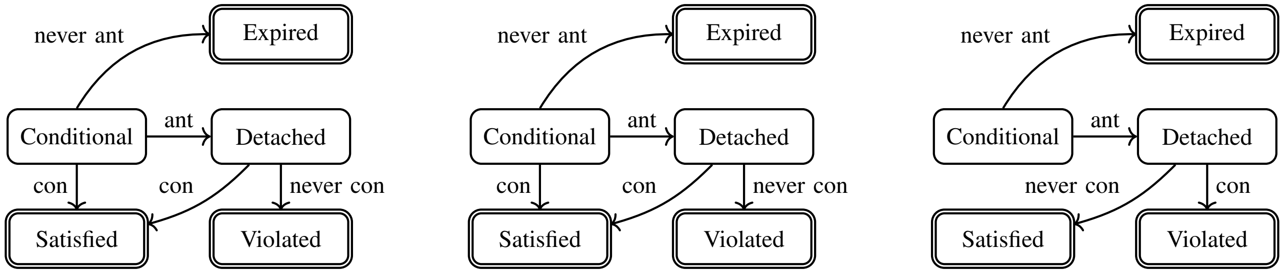
Each event is represented with a happens predicate describing the event and when the event happened. For example, patient John gives physician Bob consent to view his EHR at Time 1 (Line 1), or there is an emergency at hospitalNC at Time 12 (Line 4). EC supports concurrent events in discrete time. That is, multiple events can happen at the same time point. For example, physicians Bob and Alice access patients' EHR during the emergency (Lines 5–6). Such an enactment can be extracted using the logs of EHR software (see the open source OpenEMR project [2] for an example).

B. Norms

We capture the interactions among users via social norms. Definition 1 describes a *norm*. We adopt Singh's [29] model of social norms, and extend it with temporal constraints. Below, N is a placeholder for C, A, P, signifying a commitment, authorization, and prohibition, respectively.

Definition 1. *A norm* N(SUBJECT, OBJECT, *antecedent, consequent) is a directed relationship between its subject and object about its consequent to be satisfied when its antecedent holds.*

Table II describes the formal syntax of norms. For simplicity, we describe norm syntax and semantics via exam-



(a) Commitment (accountable: subject)

(b) Authorization (accountable: object)

(c) Prohibition (accountable: subject)

Fig. 1. Lifecycle of norms. Double rectangles represent terminal states (i.e., the norm's lifecycle ends in those states). To highlight the differences, we use the same layout of states in each diagram.

TABLE II
SYNTAX OF NORMS.

Norm	→	Commitment | Authorization | Prohibition
Commitment	→	C(ROLE, ROLE, Expr, Expr)
Authorization	→	A(ROLE, ROLE, Expr, Expr)
Prohibition	→	P(ROLE, ROLE, Expr, Expr)
Expr	→	true | ϕ | never Expr | ¬Expr | Expr ∧ Expr

ples. Below, PHYSICIAN, HOSPITAL, and PATIENT represent roles to be instantiated with actual agent names at run time. Authorization A(PHYSICIAN, HOSPITAL, treat(PHYSICIAN, PATIENT), access_EHR(PHYSICIAN, PATIENT)) means that physicians are authorized by a hospital to access their patients' EHR. Commitment C(PHYSICIAN, HOSPITAL, true, operate(PATIENT)) means that a physician is (unconditionally) committed to the hospital to operating upon patients. Prohibition P(PHYSICIAN, HOSPITAL, ¬consent(PHYSICIAN, PATIENT), access_EHR(PHYSICIAN, PATIENT)) means that a physician is prohibited from accessing a patient's EHR without consent in regular practice (nonemergency) mode.

C. Requirements

We consider three types of requirements [7]: *Must*, *Must Not*, and *May*. A *Must* requirement indicates a commitment for its user. Consider the following requirement from HIPAA [32]: a physician must obtain consent before accessing the patient's EHR. A *Must Not* requirement indicates a prohibition for its subject (e.g., regarding security and privacy). Consider the following: a physician must not disclose a patient's protected health information (PHI), which means that physicians are prohibited from performing any action that discloses the patient's PHI. A *May* requirement indicates an authorization. Consider the following: a physician may access a patient's EHR without consent in emergencies. We assume that the requirements are explicit, and do not tackle requirements elicitation.

III. TEMPORAL NORM ENACTMENTS

We describe how to formalize norms and their enactments.

A. Temporal Norms

We incorporate deadlines in all norm types, extending previous work on commitments [9], [16]. Listing 2 provides the rules defining a norm's lifecycle (as Figure 1 shows).

Listing 2
NORM LIFECYCLE IN THE EVENT CALCULUS.

```
1  %%% states %%%
2  conditional(N, T):-
3   holds_at(status(N, conditional), T).

5  expired(N, T):-
6   holds_at(status(N, expired), T).

8  detached(N, T):-
9   holds_at(status(N, detached), T).

11 satisfied(N, T):-
12  holds_at(status(N, satisfied), T).

14 violated(N, T):-
15  holds_at(status(N, violated), T).

17 %%% transitions %%%
18 terminates(E, status(N, conditional), T):-
19  expire(E, N, T).

21 initiates(E, status(N, expired), T):-
22  expire(E, N, T).

24 terminates(E, status(N, conditional), T):-
25  detach(E, N, T).

27 initiates(E, status(N, detached), T):-
28  detach(E, N, T).

30 terminates(E, status(N, detached), T):-
31  discharge(E, N, T).

33 initiates(E, status(N, satisfied), T):-
34  discharge(E, N, T).

36 terminates(E, status(N, detached), T):-
37  violate(E, N, T).

39 initiates(E, status(N, violated), T):-
40  violate(E, N, T).
```

Listing 2 considers four norm states: conditional, detached, satisfied, and violated. Each state is described via a status predicate (Lines 2–15). Transitions among states are described via initiates and terminates predicates. An expire event terminates the conditional state (Lines 18–19), and initiates the expired state (Lines 21–22). A detach event terminates the conditional state (Lines 24–25), and initiates the detached state (Lines 27–28). A discharge event terminates the detached state (Lines 30–31), and initiates the satisfied state (Lines 33–34). A violate event terminates the detached state (Lines 36–37), and initiates the violated state (Lines 39–40). Note that the lifecycle rules of Listing 2 apply to all norm types. The lifecycle operations *expire*, *detach*, *discharge*, and *violate* are specified for each norm type according to the lifecycle of that norm type (Figure 1).

Listing 3
PROHIBITION LIFECYCLE IN THE EVENT CALCULUS.

```
1  expire(E,p(S,O,Ant,[Ts,Te],Con,Tc),T):-
2   conditional(p(S,O,Ant,[Ts,Te],Con,Tc),T),
3   T > Te.

5  detach(E,p(S,O,Ant,[Ts,Te],Con,Tc),T):-
6   conditional(p(S,O,Ant,[Ts,Te],Con,Tc),T),
7   initiates(E, Ant, T),
8   T >= Ts, T =< Te.

10 discharge(E,p(S,O,Ant,Ta,Con,[Ts,Te]),T):-
11  detached(p(S,O,Ant,Ta,Con,[Ts,Te]),T),
12  T > Te.

14 violate(E,p(S,O,Ant,Ta,Con,[Ts,Te]),T):-
15  detached(p(S,O,Ant,Ta,Con,[Ts,Te]),T),
16  initiates(E, Con, T),
17  T >= Ts, T =< Te.
```

Listing 3 describes the lifecycle operations that are specific to a prohibition. A prohibition is expired (Line 1) if no event that satisfies its antecedent happens within the antecedent deadline, i.e., Te passes (Line 3) when the prohibition is conditional (Line 2). This corresponds to the "never ant" transition shown in Figure 1(c), i.e., the antecedent is never initiated before the deadline. An event detaches a prohibition (Line 5) if the prohibition is conditional (Line 6), the event initiates the antecedent of the prohibition (Line 7), and the event happens within the antecedent deadline [Ts, Te] (Line 8). A prohibition is satisfied (Line 10) if no event that satisfies its consequent happens within the consequent deadline, i.e., Te passes (Line 12) when the prohibition is detached (Line 11). This corresponds to the "never con" transition shown in Figure 1(c), i.e., the consequent is never initiated before the deadline. An event violates a prohibition (Line 14) when the prohibition is detached (Line 15), the event initiates the consequent of the prohibition (Line 16), and the event happens within the consequent deadline [Ts, Te] (Line 17). Lifecycle operations for commitments and authorizations are analogous.

With respect to a prohibition, a commitment reverses the transitions from detached to satisfied and violated. An authorization is violated when the antecedent holds but the consequent fails to hold. A subject who is authorized for performing a consequent is not committed to doing so. Importantly, whereas the subject of a commitment or prohibition is accountable to its object, it is the object of an authorization that is accountable to its subject. This understanding of authorizations as privileges of the authorized party agrees with established approaches [29], [33].

Definition 2 describes the EC proof procedure. Queries can be given in the form of (i) ground predicates such as "holds_at(consent(drBob, john), 3)." for which the solution is either true or false, or (ii) nonground predicates such as "holds_at(consent(drBob, Patient), 5)." for which the solution consists of all patients who have given Bob consent.

Definition 2. *Given EC axioms Ax, a normative theory Th, a domain model D, an enactment $\mathcal{N}$, and a query Q, an EC reasoner returns a solution set $\mathcal{S}$, denoted $\mathcal{S} \Leftarrow$ EC(Ax, Th, D, $\mathcal{N}$, Q), based on the Prolog proof procedure [20].*

B. Enactments

Definition 3 formally defines an enactment.

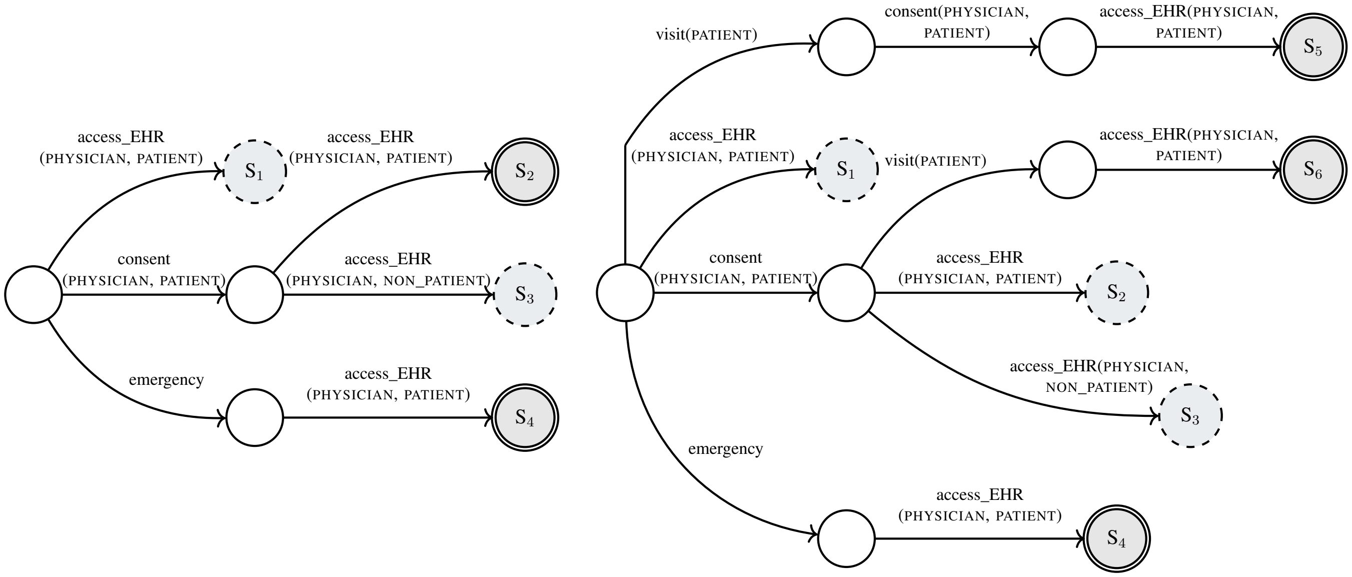
Definition 3. *An enactment $\mathcal{N}$ is a finite set of $\langle$event, time$\rangle$ pairs $\{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\}$.*

Recall that an EC formalization supports concurrent events. Definition 4 describes a compliant enactment with regards to an enactment (Definition 3) and the EC reasoner (Definition 2). An enactment $\mathcal{N}$ of a norm n is compliant if the events contained in $\mathcal{N}$ discharge n, i.e., n is satisfied at t_{m+1}, which is the next point in time after $\mathcal{N}$ ends.

Definition 4. *Let $\mathcal{N} = \{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\}$ be an enactment. Let $t_{m+1} = \max(\{t_1, \ldots, t_m\}) + 1$ be the time point immediately after $\mathcal{N}$. Then $\mathcal{N}$ is a compliant enactment of a norm n if and only if EC(Ax, Th, D, $\mathcal{N}$, "satisfied(n, t_{m+1}).")*

Consider P(PHYSICIAN, HOSPITAL, ¬consent(PHYSICIAN, PATIENT) ∧ ¬emergency, access_EHR(PHYSICIAN, PATIENT)). Figure 2(a) shows the enactments of the above prohibition. If the physician views a patient's EHR without consent, the norm is violated (S_1). If the physician first obtains consent, then views the EHR, this is a compliant enactment of the norm (S_2). If the physician views a patient's EHR without consent, the norm is violated (S_3). If there is an emergency and the physician views a patient's EHR, this is a compliant enactment (S_4).

We extend such enactments with refinements of a norm. Consider the following usual practice of viewing EHR: "EHR is usually accessed during a nonemergency patient visit." We implement this practice as a refinement of the original norm. A refinement of a norm is the generalization or specialization of its antecedent or consequent. The above prohibition is refined into P(PHYSICIAN, HOSPITAL, (¬consent(PHYSICIAN, PATIENT) ∨ ¬visit(PATIENT)) ∧ ¬emergency, access_EHR(PHYSICIAN, PATIENT)). Figure 2(b) shows the enactments of the refined norm. Two new compliant enactments are created where a patient visit precedes the physician viewing EHR (S_5 and S_6). Both enactments are compliant enactments according to the refinement of the norm. Moreover, S_4 remains compliant since the refinement does not cover emergency situations. However, S_2 now represents a potential misuse since EHR is accessed without a patient visit.


access_EHR (PHYSICIAN, PATIENT)
S$_1$
access_EHR (PHYSICIAN, PATIENT)
S$_2$
consent (PHYSICIAN, PATIENT)
access_EHR (PHYSICIAN, NON_PATIENT)
S$_3$
emergency
access_EHR (PHYSICIAN, PATIENT)
S$_4$
visit(PATIENT)
consent(PHYSICIAN, PATIENT)
access_EHR(PHYSICIAN, PATIENT)
S$_5$
access_EHR (PHYSICIAN, PATIENT)
S$_1$
visit(PATIENT)
access_EHR(PHYSICIAN, PATIENT)
S$_6$
consent (PHYSICIAN, PATIENT)
access_EHR (PHYSICIAN, PATIENT)
S$_2$
access_EHR(PHYSICIAN, NON_PATIENT)
S$_3$
emergency
access_EHR (PHYSICIAN, PATIENT)
S$_4$

(a) Enactments of the original prohibition.

(b) Enactments of the refined prohibition.

Fig. 2. Norm enactments. The left model shows enactments for the original prohibition P(PHYSICIAN, HOSPITAL, ¬consent(PHYSICIAN, PATIENT) ∧ ¬emergency, access_EHR(PHYSICIAN, PATIENT)). The right model shows enactments for the refined prohibition P(PHYSICIAN, HOSPITAL, (¬consent(PHYSICIAN, PATIENT) ∨ ¬visit(PATIENT)) ∧ ¬emergency, access_EHR(PHYSICIAN, PATIENT)). Circles represent alternative points in time that correspond to various norm states. Edges represent events. The norm is satisfied in a double circle, and violated in a dashed circle.

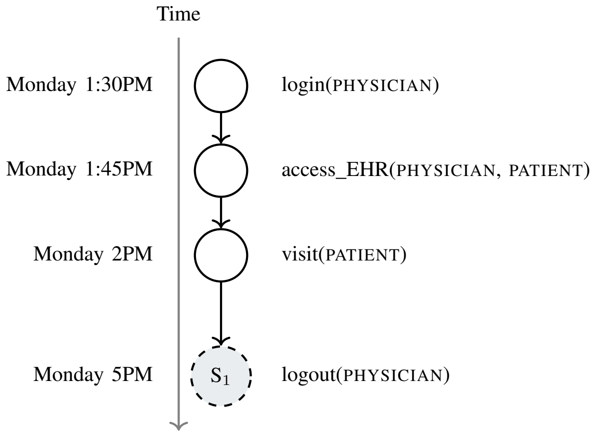


Fig. 3. Temporal enactments for norm C(PHYSICIAN, HOSPITAL, access_EHR(PHYSICIAN, PATIENT), logout(PHYSICIAN, one_hour)). A dashed circle represents a nonconformant event (potential misuse).

Having a formal refinement process can improve capturing misuse cases. However, the enactments in Figure 2 rely on ordering of events $\langle e_1, \ldots, e_n \rangle$, not absolute deadlines. Because such a representation might miss important misuse cases, we introduce explicit temporal enactments to capture conformant events (formally described in Section IV). Figure 3 demonstrates an enactment regarding active EHR sessions. Consider C(PHYSICIAN, HOSPITAL, access_EHR(PHYSICIAN, PATIENT), logout(PHYSICIAN, one_hour)), which means that a physician must logout from the active EHR session within an hour of accessing a patient's EHR. On Monday, the physician logs in to the computer and views the patient's EHR before the scheduled visit with the patient. This is usual practice for physicians to prepare for patient visits, and is compliant with respect to the prohibition of Figure 2(a). However, the physician does not log out of the computer after viewing the patient's EHR. This is a violation of the commitment, and the late log out (at 5PM in S$_1$) is a nonconformant event.

IV. The Nane Framework

We adopt the conception of a sociotechnical system (STS). An STS is a social organization [29], wherein autonomous agents representing stakeholders interact with each other through and about technical components. Figure 4 summarizes our conception of the Nane framework based on STS elements: software components and social norms. Requirements analysts specify norms based on requirements. Some norms map to software implementations (e.g., RBAC policies). A software controller realizes these policies, and produces logs based on user actions and environment conditions. Other norms regulate the interactions of users. For each norm, Nane generates enactments that involve only the events relevant to the norm's antecedent and consequent. Enactments that violate a norm are potential misuses. Nane presents each norm along with any identified potential misuses to the analyst. The analyst may refine a norm by weakening or strengthening its antecedent or consequent.

Nane takes as input: (i) a domain model that consists of misuse cases and RBAC mechanisms represented as temporal rules, (ii) an enactment extracted from logs, (iii) the normative theory that computes the progression of norms due to the enactment, and (iv) a monitoring query. Nane then computes whether the given enactment contains a potential misuse. Section V presents details of how Nane enables monitoring of

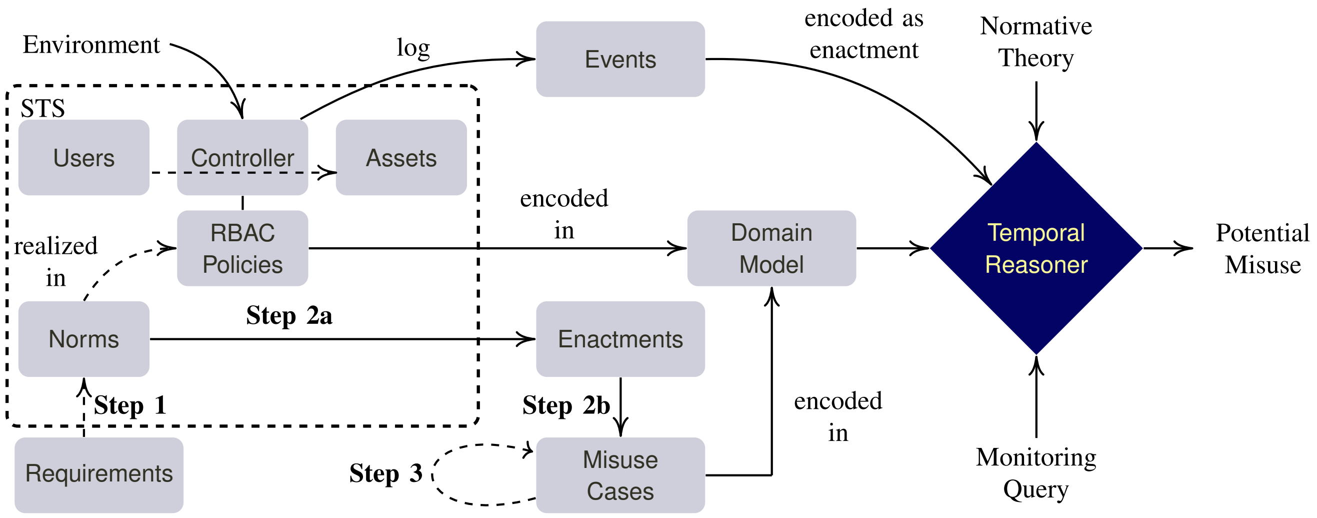


Fig. 4. Conception of the NANE framework. Solid arrows represent automated tasks. Dashed arrows represent tasks that require the involvement of the analyst.

potential misuses. Next, we describe each phase of generating misuse cases.

Step 1: Specification of norms from requirements

Norms formally capture requirements, as shown in Section II-C. Let us review some requirements from HIPAA [31]:

> "In most cases, parents are the personal representatives for their minor children. Therefore, in most cases, parents can exercise individual rights, such as access to the medical record, on behalf of their minor children."

We represent this requirement via the authorization A(PARENT, HOSPITAL, representative(PARENT, MINOR), access_EHR(PARENT, MINOR)).

> "A covered entity must disclose protected health information to HHS when it is undertaking a compliance investigation."

We represent this requirement via the commitment C(COVERED_ENTITY, HHS, investigation, disclose_PHI).

> "A covered entity may not disclose protected health information, except the individual who is the subject of the information authorizes in writing."

We represent this requirement via the prohibition P(COVERED_ENTITY, HOSPITAL, consent(PATIENT), disclose_PHI(PATIENT)).

Step 2a: Generation of norm enactments

Algorithm 1 generates all possible enactments of a norm that involve the predicates occurring in the norm's antecedent and consequent. First, the algorithm initializes the set of enactments E (Line 1), and extracts propositions from the antecedent and consequent of the norm (Line 2). Then, the algorithm populates E with permutations of the set of extracted propositions P (Lines 3–4), and removes impossible enactments according to the domain model (Line 5). For example, a patient visit cannot happen before the patient is admitted to the hospital. Finally, the algorithm returns E (Line 6). Consider prohibition P(PHYSICIAN, HOSPITAL, ¬consent, access_EHR). There are two propositions (consent and access_EHR), and thus five enactments (permutations(2,0) + permutations(2,1) + permutations(2,2)): {⟨ ⟩, ⟨consent⟩, ⟨access_EHR⟩, ⟨consent, access_EHR⟩, ⟨access_EHR, consent⟩}. There are no impossible enactments in this set of generated enactments. Note that we do not deal with enactments where an event can be repeated an infinite number of times. Moreover, our generation process explores one norm at a time, which reduces its complexity.

```
Algorithm 1: E ⇐ enactments(n)
  Input: n: norm
  Output: E: norm enactments
1 $E \leftarrow \emptyset$;
2 $P \leftarrow$ propositions(n);
3 foreach $i = 0 \ldots size(P)$ do
4     $E \leftarrow E \cup$ permutations(P, i);
5 $E \leftarrow$ prune(E);
6 return E;
```

Step 2b: Identification of misuse cases

Algorithm 2 identifies which enactments generated by Algorithm 1 are misuse cases. First, the set of misuse cases M is initialized (Line 1), and enactments are gathered (Line 2). Then, we compute for each enactment whether it is a compliant enactment of the norm (Definition 4). If the enactment is not compliant (Line 4), then it is added to M (Line 5). Formal representation of misuse cases in EC are given in Section V. Finally, the algorithm returns M (Line 6). Algorithm 2 is run for each norm to identify potential misuse cases.

Step 3: Refinement of misuse cases

An analyst goes through the stated norms, and specifies necessary refinements. Let us revisit P(PHYSICIAN, HOSPITAL, ¬consent(PHYSICIAN, PATIENT) ∧ ¬emergency,

```
Algorithm 2: M ⇐ misuse(n)
  Input: n: norm
  Output: M: potential misuses
  M ← ∅;
  E ← enactments(n);
  foreach e ∈ E do
      if !compliant(e, n) then
          M ← M ∪ {e};
  return M;
```

access_EHR(PHYSICIAN, PATIENT)). One refinement of this prohibition is P(PHYSICIAN, HOSPITAL, (¬consent(PHYSICIAN, PATIENT) ∨ ¬visit(PATIENT)) ∧ ¬emergency, access_EHR(PHYSICIAN, PATIENT)). The consequent of the norm remains the same, whereas its antecedent is more specific with the inclusion of a patient visit. This refinement generates additional misuse cases.

A refinement of the above norm does not capture how a physician's access to EHR is related to the time a patient visit happened. Therefore, we need to specify additional temporal rules to describe time-related properties of events. These *conformance* rules check whether (i) an event happened at an expected point in time (an *absolute time* temporal rule), e.g., a conformant patient visit should happen during the day, and (ii) an event happened at an expected point in time with respect to another conformant event (a *relative time* temporal rule), e.g., a conformant EHR access should happen within two hours of a patient visit.

First, we describe a conformant patient visit using an absolute time temporal rule. Consider the following expectation: "A visit happens during normal office hours." This is represented with the rule in Listing 4. A patient visit is conformant if it happens at Time T (Line 1), and T is between the normal office hours (Lines 2–3). Office hours are given as 9AM to 5PM (Line 5).

Listing 4
CONFORMANCE WITH RESPECT TO AN ABSOLUTE TIME INTERVAL.

```
1  conformant(visit(Physician, Patient), T):-
2   office_hours(Ts, Te),
3   T >= Ts, T =< Te.

5  office_hours(9, 17).
```

Next, we revisit the requirement about accessing a patient's EHR: "EHR is usually accessed during a patient visit in nonemergencies". We represent this requirement with P(PHYSICIAN, HOSPITAL, true, access_EHR(PHYSICIAN, PATIENT, two_hours_before_visit) ∨ access_EHR(PHYSICIAN, PATIENT, two_hours_after_visit)). That is, the physician is not supposed to access a patient's EHR more than two hours before or after the patient's visit. We can represent this with a relative time temporal rule as shown in Listing 5. An access to a patient's EHR is conformant with respect to a patient visit if it happens at Time T (Lines 1–2), there is a conformant visit from the patient at Time Tv (Lines 3–4), and T is close enough to Tv (Lines 5–6). For example, two hours before or after the visit is conformant (Line 8).

Listing 5
CONFORMANCE RELATIVE TO A REFERENCE EVENT.

```
1  conformant(
2   access_EHR(Physician, Patient),T):-
3   happens(visit(Physician, Patient), Tv),
4   conformant(visit(Physician, Patient), Tv),
5   close_to_visit(Tm),
6   T > Tv-Tm, T < Tv + Tm.

8  close_to_visit(2).
```

V. MONITORING MISUSE WITH NANE

We now describe how NANE enables monitoring of potential misuses via temporal reasoning. A misuse corresponds to either (i) a norm violation, which can be automatically identified from an enactment, or (ii) a nonconformant event, as described by other domain rules. Listing 6 demonstrates each rule. An event is considered a misuse if it violates a norm (Lines 1–4), or is not a conformant event (Lines 6–9). The \+ symbol denotes negation as failure [12].

Listing 6
REPRESENTING MISUSE IN EC.

```
1  misuse(evt(Event, Te), T):-
2   happens(Event, Te),
3   violate(Event, Norm, Te),
4   Te =< T.

6  misuse(evt(Event, Te), T):-
7   happens(Event, Te),
8   \+ conformant(Event, Te),
9   Te =< T.
```

Definition 5 formally describes a monitoring task in EC. The enactment includes a window of events that happened until the time monitoring takes place. Window-based approaches [6] are known to improve efficiency of run time tasks such as monitoring. The purpose of monitoring is to find potential misuses given such an enactment.

Definition 5. *A monitoring task for a domain D is represented as an EC reasoning task $\mathcal{S} \Leftarrow EC(Ax, Th, D, \mathcal{N}, Q)$ at each time point T_m, where*

- $\mathcal{N} = \{\langle e_i, t_i\rangle \mid T_m - w \leqslant t_i \leqslant T_m\}$*, where w is the window size;*
- Q = *"findall(Misuse, misuse(Misuse, T_m), Misuses).";*
- $\mathcal{S} = \{evt(e_1,\ t_1),\ \ldots,\ evt(e_n,\ t_n)\}$.

Now, let us revisit Example 1 and formalize its domain model in EC. Listings 5–8 constitute the domain model. We have the following prohibition from earlier: P(PHYSICIAN, HOSPITAL, ¬consent(PHYSICIAN, PATIENT), access_EHR(PHYSICIAN, PATIENT)), which means that physicians are prohibited from accessing patients' EHR without consent. Moreover, we have the following commitment: C(PHYSICIAN, HOSPITAL, access_EHR(PHYSICIAN, Patient), logout(Physician)), which means that physicians are committed to logging out from their EHR sessions.

Listing 7 describes a temporal rule for the above commitment. A logout by a physician is conformant if it happens at Time T (Line 1), there is an access to EHR by the physician at Time Te (Line 2), and the physician is inactive only for a short period of time in between Te and T (Lines 3–5). For simplicity, we adopt one hour as the time of inactivity (Line 7).

Listing 7
Conformant logout event.

```
1  conformant(logout(Physician), T):-
2   happens(access_EHR(Physician,Patient),Te),
3   inactive(Physician, [Te, T]),
4   active_session(Ts),
5   T-Te < Ts.

7  active_session(1).
```

In addition to the temporal rules that verify whether an event is conformant, we model domain facts and access control rules via initiates and terminates predicates as shown in Listing 8. The fluent logged_in is initiated when a user logs in (Line 1), and terminated when the user logs out (Line 2). A physician can access a patient's EHR only after logging in (Lines 4–5). Some events are marked conformant using domain facts (Line 7), and hence they are not misuses.

Listing 8
Domain rules and facts.

```
1  initiates(login(User), logged_in(X), T).
2  terminates(logout(User), logged_in(X), T).

4  initiates(access_EHR(X, Y), ehr(X, Y), T):-
5   holds_at(logged_in(X), T).

7  conformant(login(X), T).
```

Next, we demonstrate how we can perform monitoring by combining various pieces of the domain model. Listing 9 shows an enactment for a period of 72 hours (the window size). The time points represent hours from the start of the work week. For example, 1–24 represents Monday.

Listing 9
Sample enactment for the monitoring scenario.

```
1   % Monday
2   happens(login(drBob), 8).
3   happens(access_EHR(drBob, john), 9).
4   happens(logout(drBob), 10).
5   happens(give_consent(drBob, john), 16).
6   % Tuesday
7   happens(login(drBob), 32).
8   happens(access_EHR(drBob, john), 33).
9   happens(visit(drBob, john), 34).
10  happens(logout(drBob), 35).
11  % Wednesday
12  happens(login(drBob), 56).
13  happens(access_EHR(drBob, kate), 60).
14  happens(logout(drBob), 64).
```

We use the following query to find potential misuses related to this enactment: "findall(Misuse, misuse(Misuse, 72), Misuses)." That is, the EC reasoner finds all predicates Misuse that represent a misuse (Listing 6) that happened before Time 72 (end of Wednesday), and puts them in the list Misuses. The solution is the following: [evt(access_EHR(drBob, john), 9), evt(access_EHR(drBob,kate), 60), evt(logout(drBob), 64)]. Let us review each misuse:

- Physician Bob's access to John's EHR at Time 9 (Line 3) is a violation of the prohibition because there is no consent from John.
- Physician Bob's access to Kate's EHR at Time 60 (Line 13) is a violation of the prohibition because there is no consent from Kate.
- Physician Bob does not log out (Line 14) until after four hours of accessing Kate's EHR at Time 60. Thus, logout is a nonconformant event.

Note that physician Bob's access to John's EHR at Time 33 is a conformant event (Listing 5) because there is a conformant visit from John one hour later. Thus, this access is not listed as a misuse in the solution set.

VI. Limitations and Threats to Validity

Nane faces the following important limitations.

Modeling: Since Nane captures all and only the stated norms, a missing norm might lead to a misuse being unnoticed. Therefore, the correct identification of misuse cases depends upon how well the requirements analyst captures the norms as well as the domain model.

Tooling: We did not evaluate how well Nane scales with increasing number of norms. Although Nane's enactment generation process is intended as a design time tool, inefficiency might be a concern.

Representation of conflicts: Not all norm violations are misuses. Depending on the context, a norm violation might be necessary. For example, the consent requirement for accessing a patient's EHR might be waived to save a patient's life. Nane does not deal with such conflicting norms [3], and may include such norm violations as false positives in its solution set.

In light of our research goal, we can see that a threat to validity of Nane achieving that goal is of its applicability. Specifically, real-life misuse cases may be more subtle than we can characterize and explore via Nane. An important future direction, therefore, is to conduct case studies involving real-life misuses and ideally in more than one domain.

VII. Related Work

The sociotechnical aspects of requirements engineering have been gaining prominence [10], [34]. Nane demonstrates a computational approach that goes beyond the conceptual work in previous approaches. Our recent approach, Revani [15], provides a temporal logic approach for developing specifications of sociotechnical systems. Nane goes beyond Revani in capturing misuse cases and determining what must be logged to detect such misuses.

Brost and Hoffmann [8] discuss misuse in eHealth systems. We plan to investigate real misuse incidents in healthcare to evaluate NANE's coverage of identifying those via norm violations. Matulevičius et al. [24] investigate misuse cases for security modeling using the Information System Risk Management (ISSRM) model. They propose a conceptual model and template for representing misuse cases. However, these works lack a rich temporal representation like ours or support for monitoring of misuses. Moreover, we incorporate autonomy and social interactions among users via norms.

Karpati et al. [17] perform an experiment on the usefulness of misuse case maps (MUCMs). A MUCM extends a traditional set of use cases with vulnerabilities and exploit paths to identify security threats. Karpati et al.'s results indicate that a MUCM promotes understanding of scenarios (e.g., a bank hacking case) better than system architecture diagrams. However, MUCMs do not perform significantly better than system architecture diagrams in identifying vulnerabilities. Whereas MUCMs are helpful for discovering technical vulnerabilities, NANE extends MUCM by providing coverage of misuse on the social level by investigating norm violations.

Jureta et al. [14] present a classification of requirements for adaptive systems based on the expectations and needs of stakeholders. They propose a formal method for monitoring requirements as well as weakening (relaxation) some requirements when they cannot be satisfied all the time. Jureta et al. discuss the details of their conceptual framework. However, they do not provide its formalization in logic. In contrast, NANE supports a nonmonotonic logic theory via EC, and follows a normative approach for incorporating the social aspects of software systems. Georg et al. [13] use Activity Theory (AT) to identify the relations between the elements of a sociotechnical system, and combine it with the User Requirements Notation (URN) goal modeling tool to elicit requirements. They do not consider norms for modeling requirements. Moreover, we do not elicit requirements, but rather use the stated requirements to identify misuse cases. However, goal modeling techniques might help represent the goals of an attacker as misuse cases.

Kafalı and Yolum [16] propose an approach for monitoring an agent's interactions to determine whether the agent is progressing as expected. In particular, they verify whether the agent's expectations are satisfiable by its current state. Chesani et al. [9] propose a monitoring approach for commitments via the Reactive Event Calculus ($\mathcal{REC}$). We support a more general model of norms, and provide generation of expectations from norm enactments. Artikis et al. [6] propose a novel dialect of EC, called RTEC, for efficient run time recognition of events. Their approach is based on windowing techniques, and provide a scalable implementation. RTEC can be used as the EC reasoner for our monitoring task to enable experiments on large streams of events.

Chopra and Singh [11] propose Custard, a language to compute the states of norm instances based on happened events. SQL queries can be generated based on their specification (tuple relational calculus), and used to query the progression of norms in a sociotechnical system. Chopra and Singh model an additional norm, *power*, to describe institutional authorities for granting or revoking other norms. Their treatment of events is similar to ours. Unlike their formalization, we do not give the details of how norm states are computed from events, but rely on the EC proof procedure.

Having a formal representation and enumeration of misuse cases helps establish correct forensic logging, i.e., determining which user actions need to be logged [4], [21]. King et al. [19] define a forensicability metric, and develop heuristics for identifying logging requirements. They perform experiments on iTrust, an open source electronic health records system. Peisert et al. [27] propose a forensic logging model based on the goals of an attacker. The attacker (according to their model) performs a series of actions to achieve a set of intermediate goals, which eventually leads to the ultimate goal. Their framework works backwards from the ultimate goal with a *requires* and *provides* (i.e., preconditions and postconditions) capability model for goals. We, on the other hand, focus on the temporal aspects of events without regard to goals. Moreover, we take a sociotechnical view, where the ultimate goal of the attacker corresponds to a norm violation. Some approaches discuss the integrity of logs [35]—how a dishonest user can modify the logs either to save a malicious user or to frame an honest user. However, log integrity is beyond our scope.

Arasteh et al. [5] investigate how various attack patterns can be detected via logs. Their model builds a lifecycle of attacks as intrusion–compromise–misuse–withdrawal. Khodabandelou et al. [18] propose Map Miner Method, which uses Hidden Markov Models to extract users' intentions from activity logs. Sindre and Opdahl [28] propose five steps for the integration of misuse cases into the security requirements elicitation process: identify critical assets, define security goals, identify threats (misuse), analyze risks, and define security requirements. Our approach focuses on the misuse identification step and provides a rich temporal model to capture such cases.

Amir-Mohammadian et al. [4] propose an audit model based on information algebra (e.g., treating traces of programs as information), implement logging specifications via program rewriting, and prove the correctness of their model. They provide a case study on the OpenMRS medical records system. Their audit mechanism for medical emergencies should provide the level of accountability equal to the strict access control rules in regular practice. For example, a physician is allowed to access any patient's records, but all access is logged. Marinovic et al. [23] propose a similar break-glass access control language, Rumpole, to incorporate exceptions (e.g., waive some of the access control rules). Rumpole reasons based on knowledge of relevant facts to determine whether there is an exception situation. Our model introduces the social aspects that lack in these approaches via norms, which enables us to systematically identify potential misuses that arise from users' interactions.

VIII. Conclusions

We proposed Nane, a temporal reasoning framework to systematically generate norm enactments and identify misuse cases. We demonstrated how Nane can be used for proactive monitoring of potential misuses via logs. We identify the following directions for future work.

Classification: A systematic classification of breaches regarding real healthcare incidents would help identify how user-originated cases (misuse) differ from software-originated cases. Maintaining such a repository of healthcare-related incidents would enable us to perform an empirical evaluation of Nane's misuse identification process.

Scalability: Nane explores one norm at a time when generating norm enactments so as to ensure that the generation process is tractable for practical problems (e.g., norms with fewer predicates in the antecedent and the consequent). We will improve the generation process in future work by considering additional pruning techniques to reduce the number of generated enactments. Implementations of EC, such as $\mathcal{REC}$ [9], suffer from performance limitation in run time monitoring. However, recent implementations of EC such as RTEC [6] support efficient event recognition for both retrospective analysis (e.g., diagnosis) and monitoring. We will perform experiments with RTEC to see whether it is helpful for our window-based misuse monitoring process.

Reasoning: Nane can be extended with probabilistic reasoning [25] to calculate the likeliness of identifying a misuse given a portion of the logs. Using a domain ontology [30] would greatly simplify the requirements analyst's effort in refining norms. In addition, the process of norm extraction from requirements can be automated via adopting a natural language processing approach.

Acknowledgments

Thanks to the anonymous reviewers and Pradeep Murukannaiah for their helpful comments, and members of the Realsearch Group at NCSU for useful discussions. This research is supported by the US Department of Defense under the Science of Security Lablet grant.

References

[1] https://cwe.mitre.org/data/definitions/779.html.

[2] http://www.open-emr.org/wiki/index.php/3.1_Auditing_in_OpenEMR.

[3] N. Ajmeri, J. Jiang, R. Y. Chirkova, J. Doyle, and M. P. Singh. Coco: Runtime reasoning about conflicting commitments. *Proc. 25th International Joint Conference on Artificial Intelligence*, July 2016.

[4] S. Amir-Mohammadian, S. Chong, and C. Skalka. Correct audit logging: Theory and practice. *Proc. 5th International Conference on Principles of Security and Trust*, pages 139–162. Springer, 2016.

[5] A. R. Arasteh, M. Debbabi, A. Sakha, and M. Saleh. Analyzing multiple logs for forensic evidence. *Digital Investigation*, 4:82–91, Sept. 2007.

[6] A. Artikis, M. J. Sergot, and G. Paliouras. An event calculus for event recognition. *IEEE Trans. Know. & Data Eng.*, 27(4):895–908, 2015.

[7] S. Bradner. Key words for use in RFCs to indicate requirement levels. Harvard University, 1997. https://www.ietf.org/rfc/rfc2119.txt.

[8] G. S. Brost and M. Hoffmann. Identifying security requirements and privacy concerns in digital health applications. In A. S. Fricker, C. Thümmler, and A. Gavras, editors, *Requirements Engineering for Digital Health*, pages 133–154. Springer, 2015.

[9] F. Chesani, P. Mello, M. Montali, and P. Torroni. Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multi-Agent Systems*, 27(1):85–130, July 2013.

[10] A. K. Chopra, F. Dalpiaz, F. B. Aydemir, P. Giorgini, J. Mylopoulos, and M. P. Singh. Protos: Foundations for engineering innovative sociotechnical systems. *Proc. 18th IEEE International Requirements Engineering Conference (RE)*, pages 53–62, Aug. 2014.

[11] A. K. Chopra and M. P. Singh. Custard: Computing norm states over information stores. *Proc. 15th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1096–1105, 2016.

[12] K. L. Clark. Negation as failure. In M. L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 311–325. Morgan Kaufmann, 1987.

[13] G. Georg, G. Mussbacher, D. Amyot, D. Petriu, L. Troup, S. Lozano-Fuentes, and R. France. Synergy between activity theory and goal/ scenario modeling for requirements elicitation, analysis, and evolution. *Information and Software Technology*, 59(C):109–135, Mar. 2015.

[14] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos. The requirements problem for adaptive systems. *ACM Trans. Management Information Systems*, 5(3):17:1–17:33, Sept. 2014.

[15] Ö. Kafalı, N. Ajmeri, and M. P. Singh. Revani: Revision and verification of normative specifications for privacy. *IEEE Intell. Syst.*, To appear.

[16] Ö. Kafalı and P. Yolum. PISAGOR: A proactive software agent for monitoring interactions. *Knowl. & Info. Syst.*, 47(1):215–239, Apr. 2016.

[17] P. Karpati, A. L. Opdahl, and G. Sindre. Investigating security threats in architectural context. *J. Systems and Software*, 104:90–111, 2015.

[18] G. Khodabandelou, C. Hug, and C. Salinesi. Mining users' intents from logs. *Intl. J. Info. Syst. Model. & Design*, 6(2):43–71, Apr. 2015.

[19] J. King, R. Pandita, and L. Williams. Enabling forensics by proposing heuristics to identify mandatory log events. *Proc. Symposium and Bootcamp on the Science of Security (HotSoS)*, pages 6:1–6:11, 2015.

[20] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[21] L. Layman, S. D. Diffo, and N. Zazworka. Human factors in webserver log file analysis. *Proc. Symposium and Bootcamp on the Science of Security (HotSoS)*, pages 9:1–9:11, 2014.

[22] D. Liginlal. HIPAA and human error. In A. Gkoulalas-Divanis and G. Loukides, editors, *Medical Data Privacy Handbook*, chap. 25. Springer, 2015.

[23] S. Marinovic, N. Dulay, and M. Sloman. Rumpole: An introspective break-glass access control language. *ACM Trans. Info. Syst. Sec.*, 17(1), 2014.

[24] R. Matulevičius, N. Mayer, and P. Heymans. Alignment of misuse cases with security risk management. *Proc. Third International Conference on Availability, Reliability and Security (ARES)*, pages 1397–1404, 2008.

[25] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Proactive self-adaptation under uncertainty. *Proc. 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 1–12, 2015.

[26] S. Murphy. Is cybersecurity possible in healthcare? *National Cybersecurity Institute Journal*, 1(3):49–63, 2015.

[27] S. Peisert, M. Bishop, and K. Marzullo. Computer forensics in Forensis. *ACM Operating Systems Review (OSR)*, 42(3):112–122, Apr. 2008.

[28] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, Jan. 2005.

[29] M. P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Trans. Intell. Syst. Tech. (TIST)*, 5(1):21:1–21:23, Dec. 2013.

[30] A. Souag, C. Salinesi, R. Mazo, and I. Comyn-Wattiau. A security ontology for security requirements elicitation. In F. Piessens, J. Caballero, and N. Bielova, editors, *Engineering Secure Software and Systems*, LNCS 8978, pages 157–177. Springer, 2015.

[31] United States Department of Health & Human Services (HHS). Summary of the HIPAA privacy rule, 2003. http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/.

[32] United States Department of Health & Human Services (HHS). Bulletin: HIPAA privacy in emergency situations, 2014. http://www.hhs.gov/ocr/privacy/hipaa/understanding/special/emergency/.

[33] G. H. Von Wright. Deontic logic. *Ratio Juris*, 12(1):26–38, 1999.

[34] E. Yu, P. Giorgini, N. Maiden, and J. Mylopoulos, editors. *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[35] S. Zawoad, A. K. Dutta, and R. Hasan. SecLaaS: Secure logging-as-a-service for cloud forensics. *Proc. 8th ACM SIGSAC Symp. Info., Comput. Comm. Security*, pages 219–230, 2013.