ISSN 1884-0760

GRACE TECHNICAL REPORTS

Security Requirements Analysis and Validation with Misuse Cases and Institutional Modelling

Gideon Dadik BIBU, Nobukazu YOSHIOKA, Julian PADGET

GRACE-TR 2012-02' February 2012

CENTER FOR GLOBAL RESEARCH IN
ADVANCED SOFTWARE SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF INFORMATICS
2-1-2 HITOTSUBASHI, CHIYODA-KU, TOKYO, JAPAN

WWW page: http://grace-center.jp/

Security Requirements Analysis and Validation with Misuse Cases and Institutional Modelling

Gideon BIBU
Dept of Computer Science
University of Bath, UK
G.D.Bibu@bath.ac.uk

Nobukazu YOSHIOKA
GRACE Center
National Institute of Informatics
nobukazu@nii.ac.jp

Julian PADGET
Dept of Computer Science
University of Bath, UK
jap@cs.bath.ac.uk

February, 2012

Abstract

The need for early consideration of security during system design and development cannot be over-emphasized, since this allows security features to be properly integrated into the system rather than added as patches later on. A necessary pre-requisite is the elicitation and analysis of the security requirements prior to system design. Existing methods for the security requirements phase, such as attack trees and misuse case analysis, use manual means for analysis, with which it is difficult to validate and analyse system properties exhaustively. We present a computational solution to this problem using an institutional (also called normative) specification to capture the requirements in the InstAL action language, which in turn is implemented in answer set programming (a kind of logic programming language).

The result of solving the answer set program with respect to a set of events is a set of traces that capture the evolution of the model over time (as defined by the occurrence of events). Verification and validation is achieved by querying the traces for specific system properties. Using a simple scenario, we show how any state of the system can be verified with respect to the events that brought about that state. We also demonstrate how the same traces enable: (i) identification of possible times and causes of security breaches and (ii) establishment of possible consequences of security violations.

1 Introduction

Security has remained an increasingly important issue for organizations deploying information systems. Despite investments in the implementation of security technologies, security still remains a challenge as new kinds of threats keep coming up. This is because security has many faces, and although, security is generally addressed at a technical level, it is ultimately about social, legal and personal concerns of relevant stakeholders, hence need to be addressed using several approaches. Another reason is that security requirements are not usually considered during the system design time but rather added to the system later in the software life-cycle. Consideration of security in the system development life cycle is considered essential to implementing and integrating a comprehensive strategy for managing security risks for all information technology assets in an organization [12]. Therefore, identifying and analyzing security requirements should be an important element of the software engineering process. This would lead to the early identification of security loopholes in the system thereby helping in the provision of appropriate mitigations.

Research in requirements engineering has led to the development of many modeling notations and approaches that help the elicitation and analysis of security requirements. These approaches include Misuse case [22], Misuse patterns [7], the common criteria [26], and the attack tree [20, 15]. The approaches have been used for eliciting security requirements in different domains and have been well accepted in practice. However, the process of elicitation and analysis of security requirements through these approaches are usually either through textual presentations or graphical presentations based on either tree graphs or UML notations. These presentation methods may be convenient and adequate for eliciting security requirements in small systems but would get more complex and tedious with larger and more complex systems. Also, the approaches are driven through manual processes which make it very difficult for the validation and analysis of the elicited requirements. In this work, we proposed a computational approach to the verification and validation of elicited security requirements. The methodology is an institutional framework [5] which provides a mechanism to capture and reason about

"correct" and "incorrect" behaviour within a certain context, which in this case is security. Based on first-order logic, but inspired by deontic logic, the framework monitors the permissions, empowerment and obligations of participants and generates violations when a security breach occurs. Information on the effects of participants' actions are stored in the state of the framework as facts. The "little" facts collected about events/actions triggered by participants over time may eventually lead to "big" facts that reveal vital information about a participant's behaviour with respect to the preservation of the system security. Using misuse case analysis for the initial elicitation of security requirements, our solution provides a means for a rigorous test of the security requirements elicited. The combination of our approach with already established misuse case approach provides a tool that would be more useful in for effectively determining a system's security requirements at design time. Implementation is achieved using an action language InstAL which is based on the semantics of answer set programming (ASP). In section 2 we present a case study through which we illustrate our solution approach. In section 3 we present the our institutional framework briefly but sufficiently to establish the background for understanding our approach. We also give a short introduction to ASP in this section. The action language InstAL is introduced in section 4 and a discussion of how the misuse case is represented in InstAL. We discuss our results in section 5 and presented related works in section 6. Finally conclude in section 7.

2 Case study

The publicly available iTrust Medical Records System documentation [27] provided ample choice of use-cases for the purpose of illustration. After a careful consideration of the use-cases we chose one that we found most suitable for illustrating the institutional framework approach. The criteria for the choice of use-case is that the scenario should consist of more than one actors and a well defined process over which the actors interact in order to achieve a certain goal or subgoal of the system. This is necessary in order for us to be able to express the interactions as events while the actors are taken as agents with the aim of generating the traces of events which could be investigated for some desired security properties. The use case is described in table 1.

The initial misuse case analysis was done by capturing the misuse/threats along with the usecase descriptions. This is as shown in table The misuse case description in table 2 shows an example of some of the security threats that could affect the use-case. Figure 1 gives a UML presentation of the misuse case. For the purpose of

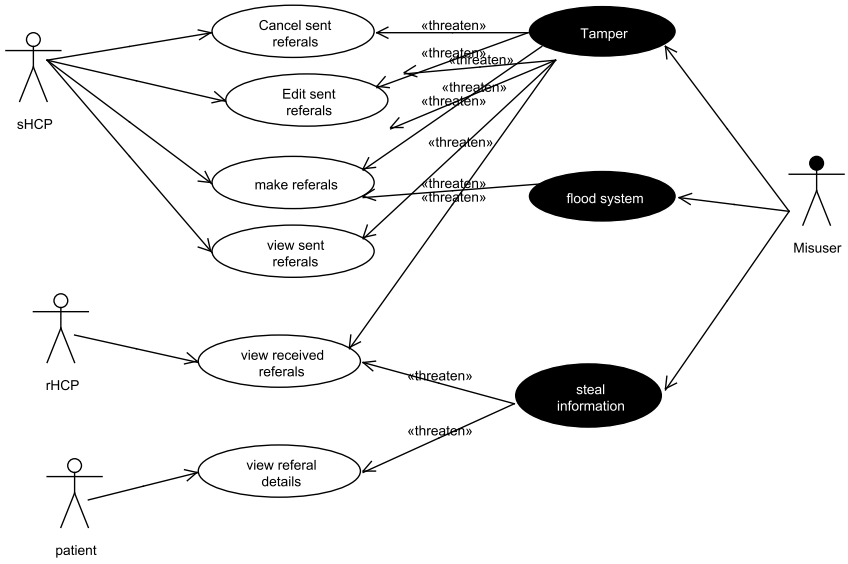

Figure 1: Inititial Misuse Case Diagram

the implementation and analysis which we intend to do through the application of the institutional framework, we reduced the case study to a single process which has a logical beginning and end consisting of a sequence of observable events. The top level use case and misuse case described earlier in tables 1 and 2 consist of activities that could be carried out independently. That is, they do not necessarily end up in a work flow scenario that

UC33 Manage Patient Referrals Use Case

33.1 Preconditions:
A patient and two HCPs are registered users of the iTrust Medical Records system (UC2). The iTrust user has been authenticated in the iTrust Medical Records system.

33.2 Main Flow:
A sending HCP refers the patient to another receiving HCP [S1]. A receiving HCP views a list of received referrals [S2]. A sending HCP views a list of previously sent patient referrals [S3]. A patient views the details of his/her referrals [S4]. A sending HCP edits a previously sent patient referral [S5]. A sending HCP cancels a previously sent patient referral [S6]. All events are logged.

33.3 Sub-flows:

- *[S1]* An HCP chooses to refer a patient to another receiving HCP through the referral feature on a patient's office visit page. The sending HCP must select a receiving HCP by either entering the HCP's MID and confirming the selection, or by searching for the HCP by name. The sending HCP is also presented with a text box to include notes about the referral. The sending HCP then chooses a priority from 1-3 (1 is most important, 3 is least important) for the referral. The HCP may send the referral, cancel the referral [E1], or edit the referral [E2]. Upon sending a referral, the patient, sending HCP, and receiving HCP receive a message summarizing the newly created referral information (sending HCP name & specialty, receiving HCP name & specialty, patient name, referral notes, and referral creation timestamp); additionally, the sending and receiving HCP messages include the referral priority.
- *[S2]* An HCP chooses to view received referrals. The receiving HCP is presented with a list of referrals sorted by priority (from most important to least important). The receiving HCP then selects a referral to view details and is presented with the name and specialty of the sending HCP, the patient's name, the referral notes, the referral priority, the office visit date with a link to the office visit, and the time the referral was created.
- *[S3]* A sending HCP views a list of previously sent patient referrals. The HCP may sort the list of referrals by patient name, receiving HCP name, time generated, and/or priority. The HCP chooses a specific referral from the list to view complete details about the referral: patient name, receiving HCP name and specialty, time generated, priority, office visit date, and notes.
- *[S4]* A patient views a list of his/her referrals. The patient may sort the list of referrals by receiving HCP name, time generated, and/or priority. The patient chooses a specific referral from the list to view complete details about the referral: sending HCP name and specialty, receiving HCP name and specialty, time generated, priority, office visit date, and notes. The patient is also provided with the option to send a message to the receiving HCP to request that an appointment be scheduled.
- *[S5]* A sending HCP edits a previously created patient referral as long as the referral has not been viewed by the receiving HCP. The sending HCP may edit the priority of the referral and/or the referral notes. The sending HCP then chooses to save the edits, cancel the edits, or re-enter the data [E2].
- *[S6]* A sending HCP cancels a previously sent patient referral by visiting the office visit page, viewing the details of a previously sent patient referral [S3], and choosing cancel. The HCP is asked to confirm the decision to cancel the referral. The patient and receiving HCP receive a message indicating that the referral was canceled.

33.4 Alternative Flows:

- *[E1]* The receiving HCP chosen is not the desired HCP. The sending HCP does not confirm the selection and is prompted to try again.
- *[E2]* The patient, receiving HCP, referral notes, and/or referral priority are invalid, and the HCP is prompted to enter this information again.

Table 1: Managing Patients Referrals Use Case [27]

Name:	Refer patient
Summary:	The sending HCP (sHCP) chooses to refer a patient to another receiving HCP (rHCP). He creates the referral details and send to the receiving HCP and the patient
Basic path:	bp1: The sHCP chooses to refer a patient to another rHCP bp2: The sHCP selects the rHCP by entering the HCPs MID and confirming the selection bp3: The system provides the sHCP with a text box to include referral notes bp4: The sHCP include notes about the referral bp5: The sHCP chooses a priority from 1 3 (1 =highest, 3 = lowest) for the referral bp6: The sHCP sends the referral bp7: The HCPs and patient receive summary of referral, with the HCPs messages including the referral priority
Alternative paths:	ap1: In bp2, the sHCP searches for the rHCP by name ap2: In bp6, the sHCP chooses to cancel the referral ap3: In bp6, the sHCP chooses to edit the referral
Exception paths:	ep1: in bp2, the chosen rHCP is not the desired HCP. The sHCP does not confirm the selection and is prompted to try again ep2: In bp1-5, the patient, rHCP, referral notes, and/or referral priority are invalid, the sHCP is prompted to enter this information again
Triggers:	Patient makes office visit
Assumptions:	ass1: all users do not have any malicious intentions
Preconditions:	prc1: The patient and the two HCPs are registered users of the iTrust system prc2: These users have been authenticated in the iTrust system
Postconditions:	psc1: The patient referral is successfully created by the sHCP psc2: The rHCP receives a summary of patient referral details with the referral priority psc3: The patient receives a summary of referral details
Threats:	Trt1: the sHCP is an impostor, possible outcomes are t1-1: A patient is unwillingly referred to a HCP t1-2: A HCP receives a fake referral information t1-3: List of patient referrals exposed t1-4: A patient referral is edited with wrong information such as priority t1-5: Cancellation of a patient referral Trt2: the rHCP is an impostor resulting in these possible outcome t2-1: Patient referral confirmed and revealed t2-2: confirms a HCP is registered on the iTrust system Trt3: the patient is an impostor with the following possible outcome t3-1: patient referral details revealed

Table 2: Misuse case description embedded in the "Refer patient use case

will allow us to model the activities. As a result. we selected the first use-case [S1] from the main flow in table 1. This has to do with the process of making a patient referral and sending to the appropriate recipients.

2.1 The "Making Referrals" Use-case

This use-case consist of two healthcare professionals (designated here as **sHCP** and **rHCP**) and a patient. The process involves the sHCP referring a patient to rHCP. The sHCP initiates the process by login to the system, prepares the referral document, and sends the referral document to rHCP, the patient and him/herself. While the sHCP and rHCP receive similar copies of the referral document, the patient receives a slightly different version.

This use-case was first analyzed for possible security threats using the misuse case approach illustrated

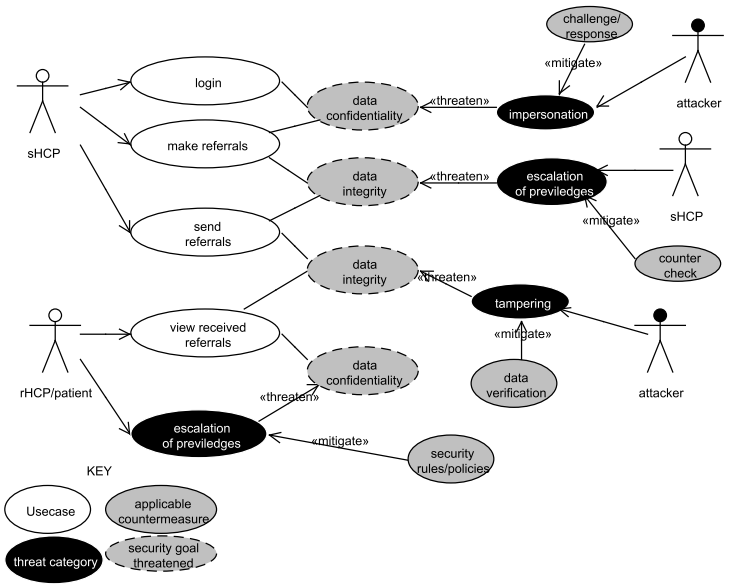

Figure 2: The *Make Referrals* Misuse Case Diagram

in figure 2. The classes of threats identified include *impersonation, escalation of privileges, and information tampering.* We consider these attacks as being the kind of attacks that could be carried out by insider attackers (escalation of privileges) and external attackers (impersonation and tampering). The original misuse-case in [23] was modified to include the security goals under threat at each stage of the process. This is a simplified version of the one proposed in [18]. The reasoning behind this is that security threats are primarily aimed at breaching security goals with respect to confidentiality, integrity, availability, and accountability. Inclusion of these in the use/misuse case diagram would make it easier to see the type of threat/attack an attacker would consider. It also helps in determining the appropriate countermeasure to apply for the attacks or threats. The security goals relevant in each use-case is drawn from the analysis of the most relevant assets associated with the use-case, hence the approach is consistent with [18].

The detail interactions for this use/misuse case description is illustrated in figure 3. This is the mal-activity diagram, based on [7, 17], showing the users, sequence of events, the attackers, attack points, threats, and countermeasures. The mal-activity diagram describes the expected order and sequence of events which include the user initiated events and the attacker/misuser events. It also shows the countermeasures to the identified security threats and the points at which the countermeasures are expected to be applied in order to mitigate appropriately the threats. At this point, the misuse case security requirements elicitation and analysis phase ends. Usually the process may be repeated, depending on whether the countermeasures introduce new threats to the system or not. However, the level of refinement is totally based on the expertise and experience of the security analyst. The misuse case approach also lacks the means of verifying and validating the behavior of the system when the identified countermeasures are applied. We therefore introduce the event-based institutional framework, implemented using the institutional Action Language (Inst*AL*) which is based on *answer set programming (ASP)* [2]. This would provide the needed reasoning mechanism for the purpose of verification and validation of the misuse case analysis.

3 The Institutional Framework

This approach is based of the definitions of institutional framework in [5] where an electronic institution is described as a multi-agent system where the agent's behaviour is governed by a set of published norms, rules or regulations which bring about a set of expected behaviours for agents interacting in a social context. It is assumed here that the norms and their expectations about the behaviour of participating agents are explicit and can be written down in a form which is machine processable.

The model consist of a world model, (potentially several) institutional frameworks, and agents. It is based

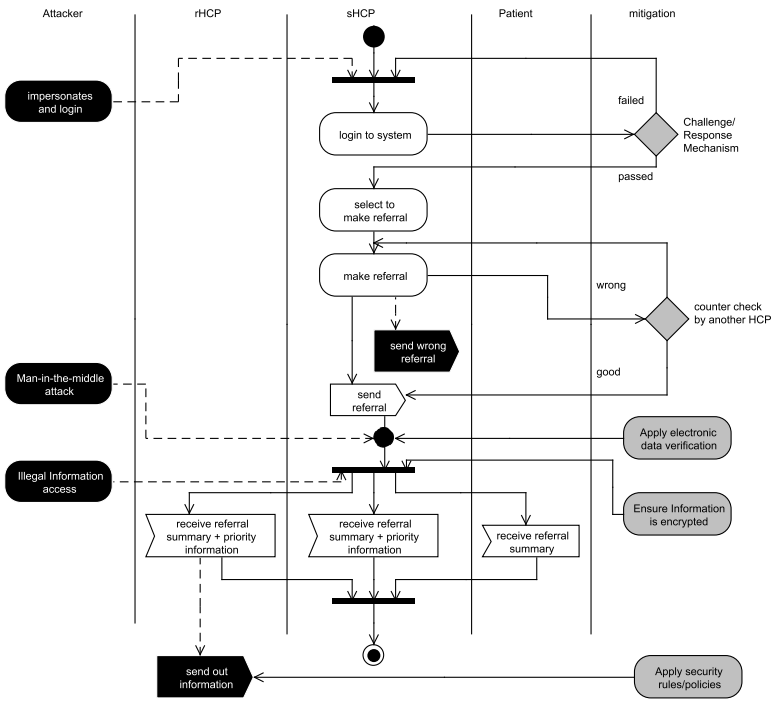

Figure 3: Mis-activity Diagram for *Make Referral* Use-case

on the notion of observable events that capture the notion of physical world events and institutional events that only have meaning within a given social context. Institutional events are not observable, but are created through *Conventional Generation*, whereby an event in one context *Counts As*[10] the occurrence of another event in a second context. Taking the physical world as the first context and by defining conditions in terms of states, institutional events may be created that count as the presence of states or the occurrence of events in the institutional world. Thus, an institution is modeled as a set of states that evolve over time subject to the occurrence of events, where an institutional state is a set of institutional fluents that are considered true at some instant. The formal specifications of the model is well presented in [5], however for the purpose of establishing the context on which this work is based, we give an informal description of the constituents of the framework here.

3.1 Components of the Institutional Framework

The institution consist of;

- **Actors** - These are the entities that interact both within the institution and outside the institution.
- **Actions** - These are the events that actors carry out. They consist of both external (exogenous) events and institutional events.
- **State** - A record of effects of previous actions, obligations and permissions.

Actions and institutional states are modeled as events and fluents respectively. Fluents can also be seen as institutional facts that can change over time as a result of occurrence of events.

Institutional facts consist of

- **Institutional Power:** This is the institutional capability for an event to be brought about meaningfully. This can then cause a change in the institutional state. Without institutional power, the event may not be brought about and hence, has no effect on institutional states.
- **Permission:** This enables events to occur without causing a violation. If an event occurs, and that event is not permitted, then a violation event is generated.
- **Obligation:** This captures the property that a particular event is obliged to occur before a given deadline event (such as a timeout) and is associated with a specified violation. If an obligation fluent holds and the obliged event occurs then the obligation is said to be satisfied. If, on the other hand the correspond-

ing deadline event occurs then the obligation is said to be violated and the specified violation event is generated.

- **Domain facts:** These are fluents specified to store the internal of institutional states of the institution as may be brought about by institutional events.

Events are classified into

- **Exogenous events:** These are observable events, which are events external to the institution and hence could be brought about independently from the institution
- **Institution events:** Events which may be brought about by the institutional semantics. They also include violation events which may arise either from explicit generation, from the occurrence of a non permitted event, or from the failure to fulfill an obligation. In these cases sanctions that may include obligations on violating agents or other agents and/or changes in agents permission to do certain actions, may then simply be expressed as consequences of the occurrence of the associated violation event in the subsequent institutional state.

Finally, a set of institutional rules associate the occurrence of events with some effects in the subsequent state. These can be divided into:

- **Generation rules:** These account for the conventional generation of events. Each generation rule associates the satisfaction of some conditions in the current institutional state and the occurrence of an (observed or institutional) event with a generated institutional event. The generating and generated events are taken by the institution to have occurred simultaneously.
- **Consequence rules:** These associate the satisfaction of some conditions in the current institutional state and the occurrence of an event in the institution or the world to the change in state of one or more fluents in the next institution state.

3.2 Institutional framework and Security

Institutional frameworks provide a means to capture and reason about "correct" and "incorrect" behaviour within a certain context. In this case we are applying it in the context of systems security. Systems security in organizations does not only depend on technical security infrastructures. Security vulnerabilities also arise from behaviours of various actors (humans, systems, processes [1]) as they interact to achieve their goals. Also some processes and system behaviours could lead to security vulnerabilities. For instance failure of a user to logout of the system could make the system vulnerable to various attacks. The institutional framework introduced above, provides a potentially useful approach to addressing these kind of security problems. Events enacted by the actors in the real world will have meanings in the institutional framework based on specified security goals and the expected behaviours for achieving the goals. The participants of our institutional framework are governed by the security goals and rules specified in the institution. The framework monitors the permissions, empowerment and obligations of their participants and generates violations when events occur in an incorrect manner. Institutional information and the effects of participants' actions is stored in the state of the framework. The change of state over time, as a result of these actions, provides participants with information about each others' behaviour. This follows from the notion that "little" facts collected about events/actions over time may eventually lead to "big" facts that reveal vital information about a participant's behaviour and about system integrity.

3.3 Answer Set Programming (ASP)

Answer set programming is a declarative logic programming paradigm that admits reasoning about possible world views in the absence of complete information. ASP allows a programmer to specify "what" needs to be done without specifying "how" it needs to be achieved.

Due to its formal semantics, the answer set semantics, which is based on AnsProlog [2], and combined with efficient solvers, answer set programming provides an excellent basis from which derived models may be queried. As with other logic-programming techniques one of the perceived advantages of using ASP is that its rigorous and formal semantics allow us to reason in a formal and useful way about the semantics of programs. In essence an answer set program can be seen as a formalization of the underlying reasoning problem in its own right, with the advantage of being able to execute this formalization directly through the use of answer set solvers.

ASP is particularly suited to model-based reasoning. An answer set programming problem typically starts with the definition of a domain or class of problem about which we wish to reason. Such definitions (written as answer-set programs) are constructed in such a way that each possible world in the domain corresponds to an answer set of the program. Queries may then be constructed over this domain in order to determine if particular models are valid according to the definition by extending the program to limit the answer sets produced to those which match the models we are interested in. ASP has been successfully applied in a number of domains including multi-agent systems for reasoning about the behaviour of a group of agents, and modeling the reasoning capabilities, knowledge and beliefs of a single agent within a multi-agent system[5].

In using ASP for reasoning about institutions, [6] show that the formal model of an institution can be translated to ASP program such that the solutions of the program, known as answer sets of the program, defined through the stable model semantics [8] correspond to the traces of the institutional framework. This is seamlessly achieved through the action language InstAL.

4 The Action Language InstAL

The language InstAL was defined by Owen Cliffe[5] in order to simplify the process of specifying institutions. Institutions are specified as InstAL programs in a human-readable ASCII text format, which can then be automatically translated into answer set programs that directly represent the semantics of the institutions specified in the original descriptions. In other words, InstAL is an abstraction of ASP that shields the programmer from the intricacies of ASP in specifying institution descriptions.

An InstAL reasoning problem consists of the following:

- One or more InstAL institution descriptions each of which describes a single institution or a multi-institution that should be processed.
- A $domain$ definition that grounds aspects of the descriptions. This provides the domains for types and any static properties referenced in the institution and multi-institution definitions.
- A *trace program* which defines the set traces of exogenous events that should be investigated.
- A *query program* which describes the desired property which should be validated by the InstAL reasoning tool.

The reasoning process over InstAL specification is summarized as follows:

1. The InstAL to ASP translator takes the institution descriptions and domain definition files as input. Using these files, the translator generates a set of answer set programs which describe the semantics of the input institutions.

2. The translated institution programs along with a trace program and query are then grounded by the LParse program (part of the Smodels toolkit).

3. This grounded program description is then given as input to the Smodels answer set solver. This produces zero or more answer sets. Each answer set corresponds to a possible model of the input institution for a given trace described by the trace program that matches the given query.

4. These answer sets may then be visualized and interpreted by the designer.

4.1 Misuse Case Implementation using Inst*AL*

In carrying out the verification and validation of the misuse case description in figure 2, we made use of the mal-activity diagram in figure 3. Since we are doing a static analysis here, we simply assume that attacker events could always happen at anytime. We are therefore concerned with the users' events and the countermeasures. The mal-activity diagram provides the necessary sequence of activities which would be represented as events in the institution specification. The implementation in InstAL consist of the following declarations;

```
institution patientRef;
type HCP; %sHCP rHCP
type Patient; %patient
type System; %sys
type Checker; %cHCP
```

```
7  % Exogenous events
8  exogenous event login(HCP);
9  exogenous event loginChal(System,HCP);
10 exogenous event selectAction(HCP);
11 exogenous event makeRef(HCP);
12 exogenous event checkRef(Checker);
13 exogenous event signRef(System);
14 exogenous event encryptRef(System);
15 exogenous event sendRef(HCP);
16 exogenous event logindl;
17 exogenous event checkRefdl;
18 exogenous event receiveRefwP(HCP);
19 exogenous event enforcePolicy(System);
20
21 % creation event
22 create event create_patientRef;
23
24 % Institutional events
25 inst event ilogin(HCP);
26 inst event iloginChal(System,HCP);
27 inst event iselectAction(HCP);
28 inst event imakeRef(HCP);
29 inst event icheckRef(Checker);
30 inst event isignRef(System);
31 inst event iencryptRef(System);
32 inst event isendRef(HCP);
33 inst event ilogindl;
34 inst event icheckRefdl;
35 inst event ireceiveRefwP(HCP);
36 inst event iapplyPolicy(HCP);
37
38 % violation events
39 violation event loginChalCompromised;
40 violation event refUnchecked;
```

Figure 4: Events declaration for the *Make Referral* use-case.

The InstAL institution specification begins with the declaration of the institution name and types. The types here represent the various actors (agents) that would be interacting in the system. It would be noticed that there is no representation of the attacker or misuser here. This is because the activities of the misuser/attacker is not explicitly modeled in this system.

Next is the declaration of events (figure 4). This consist of four kinds of event declarations;

- *Exogenous events* consist of all observable real world events as described in section 3. These events would consequently generate institutional events and cause changes to the institutional state and may also cause a transition to another real world event. The *creation event* is responsible for the creation of the institution.
- *Institution events* consist of the various events that would be generated in the institution framework as a result of the occurrence of exogenous events. These events may initiate new facts in the institution, thereby resulting in a change in the institutional state.
- *Violation events* declare events that would occur whenever there is a violation in the system, such as failure of obligations.

Following the events declaration is the the declaration of $fluents$ (figure 5). Fluents denote facts that may be present in the system state that can be added or deleted as a result of the occurrence of events in the system. *Noninnertial* fluents e.g. `loginAttacked` in line 53 are used to capture boolean relationships over several fluents.

We now describe the generation and consequence relations of the model in three phases for simplicity and ease of understanding. The three phases are *login, create referral, send referral.*

```
41 % fluents
42 fluent chalSuccess;
43 fluent chalFail;
44 fluent loginCompromised;
45 fluent goodRef;
46 fluent badRef;
47 fluent hasRef(Patient);
48 fluent hasRefwP(HCP);
49 fluent misdCheck;
50 fluent refSigned;
51 fluent refEncrypted;
52 fluent policyApplied(HCP);
53 noninertial fluent loginAttacked;
```

Figure 5: Fluents declaration for the *Make Referral* use-case.

4.1.1 The $login$ Phase

The sequence of events start off with the login event. The $login$ phase specifies the initiation of the login event and the rules that apply in order to counter the expected attack at the login phase. From figure 3, it is assumed that an external attacker can impersonate after possibly acquiring the login information by another means which we are not concerned with here. Since this attack is assumed to happen at anytime, we therefore specify the countermeasure, which in this case is a challenge-response event initiated by the system the moment there is an initial login event. We are not concerned about the detailed implementation of this countermeasure (which could be a series of events), but rather we treat it as a single event which we expect to happen at and within some time interval. The challenge-response event `loginChal(System,HCP)` is specified as an obligation (figure 6, line 60) which must happen before a deadline event happens, else it triggers a violation event `loginChalCompromised`.

The challenge-response event is expected to be triggered just once and by the deadline, the countermeasure is either successful or failure and the institution state is set accordingly. The next exogenous event in the system's sequence of events can only take place when there is no violation at this stage.

4.1.2 *Create Referral* Phase

In this phase, we specify the rules and the conditions for the creation of the referral. Also the security threats to the system which falls within this phase include escalation of privileges (an authorized user misusing his privileges), man-in-the-middle attack, and illegal access to information. Since it is equally assumed that any or all of these attacks can always happen and at anytime, the countermeasures are also specified in this phase.

The escalation of privileges threat is expected to be mitigated by initiating a check on the created patient referral for correctness. This is specified as a `checkRef(Checker)` event which must happen within a certain time span, else a violation event `refUnchecked` is triggered (figure 7, line 99). The completion of this countermeasure event will set the institution state at this point to either `goodRef` indicating the referral was good and hence satisfying the condition for the next event to happen, or `badRef`, indicating that the referral was bad, in which case the referral would have to be corrected (figure 7, line 107).

The man-in-the-middle attack is mitigated by applying some form of digital signature on the prepared and proofed referral such that any form of tampering could be detected. This is specified in the model as an event `signRef(System)` which will be permitted to happen only if the referral has been checked and it is good (figure 7, lines 11-118). It is also expected that the signature would be applied only once, hence the power and permission for this event is terminated once it has happened.

Encryption is taken as the countermeasure for illegal access of information. This is specified in the model as an event `encryptRef(System)` which would happen after the signature event has happened.

4.1.3 The *Send Referral* Phase

This is the final phase of the process we are considering. In this phase, we still see escalation of privileges as a possible threat to the confidentiality of the patient's vital information which could be contained in the

```
% trigger challenge-response upon initial login
login(HCP) generates ilogin(HCP);
logindl generates ilogindl;
loginChal(System,HCP) generates iloginChal(System,HCP);
ilogin(HCP) initiates perm(loginChal(System,HCP)),
  perm(iloginChal(System,HCP)), pow(iloginChal(System,HCP)),
%%this should happen before some deadline by which the next
    event would be triggered
  obl(loginChal(System,HCP),logindl,loginChalCompromised);

%%challenge response triggered once
iloginChal(System,HCP) terminates perm(loginChal(System,HCP)),
  perm(iloginChal(System,HCP)), pow(iloginChal(System,HCP));

%% challenge-response sets a state of success or failure
iloginChal(System,HCP) initiates chalSuccess;
iloginChal(System,HCP) initiates chalFail;
iselectAction(HCP) terminates chalFail;

%% institutional state is set to indicate an attack at login
    when challenge fails
always loginAttacked when chalFail;

%% the institutional state is set to indicate a compromise when
    obligation fails
loginChalCompromised initiates loginCompromised;

%%challenge response is triggered again whenever the obligation
    fails
loginChalCompromised initiates perm(loginChal(System,HCP)),
  perm(iloginChal(System,HCP)), pow(iloginChal(System,HCP)) if
     loginCompromised;
```

Figure 6: Generation and Consequence relations for the $login$ phase of *Make Referral* model.

referral information. The receiving health care professional (rHCP) to whom the patient has been referred could send out the information to a third party. There is therefore the need to institute appropriate rules or policies that would prevent the rHCP from carrying out such activities. This is specified in the model as an event whose permission would be initiated when the rHCP receives the referral information. The permission remains through the life time of the institution.

Finally, we declare the initial state of the institution. This represent the events that are permitted to happen from the start of the institution. These are expressed in InstAL as follows;

```
initially
perm(login(sHCP)), pow(ilogin(sHCP)), perm(ilogin(sHCP)),
perm(logindl),pow(ilogindl),perm(ilogindl),
perm(checkRefdl),pow(icheckRefdl),perm(icheckRefdl),
perm(enforcePolicy(sys)),perm(iapplyPolicy(rHCP)),
pow(iapplyPolicy(rHCP));
```

4.1.4 Domain Specification

So far, we have given the complete specification of our *Make Referral* scenario. However, there is need to specify the domain information which would be used for grounding some aspects of the institution when translating the InstAL institution to $AnsProlog$. This is specified, providing arguments for each of the declared types in the institutional specification as follows;

```
HCP: sHCP rHCP
Patient: pat01
System: sys
Checker: cHCP
```

```
selectAction(HCP) generates iselectAction(HCP);
ilogindl initiates
    perm(selectAction(HCP)),pow(iselectAction(HCP)),
  perm(iselectAction(HCP)) if chalSuccess, not loginCompromised;

%%HCP can then create the patient referral information when the
    action is selected
makeRef(HCP) generates imakeRef(HCP);
iselectAction(HCP) initiates perm(makeRef(HCP)),
  pow(imakeRef(HCP)), perm(imakeRef(HCP));

%% action selected once
iselectAction(HCP) terminates perm(selectAction(HCP)),
  pow(iselectAction(HCP)),perm(iselectAction(HCP));

%%when referral is created the event to proof the referral is
    trigered
checkRef(Checker) generates icheckRef(Checker);
checkRefdl generates icheckRefdl;
imakeRef(HCP) initiates perm(checkRef(Checker)),
  perm(icheckRef(Checker)), pow(icheckRef(Checker)),
%% and expected to occur before some deadline
  obl(checkRef(Checker),checkRefdl,refUnchecked);

%%terminate the ability to create the referral
imakeRef(HCP) terminates perm(makeRef(HCP)),pow(imakeRef(HCP)),
  perm(imakeRef(HCP));% if patientRefCreated(HCP);

%%proof can either set the state to good or bad
icheckRef(Checker) initiates goodRef;
icheckRef(Checker) initiates badRef;
isignRef(System) terminates badRef;

%%obligation failure sets institutional fluent
refUnchecked initiates misdCheck;

%% sign the referral once if checked and good
signRef(System) generates isignRef(System);
icheckRefdl initiates perm(signRef(System)),
    pow(isignRef(System)),
  perm(isignRef(System)) if goodRef, not misdCheck;
isignRef(System) terminates perm(signRef(System)),
  pow(isignRef(System)), perm(isignRef(System));

%% remake the referral if bad
icheckRefdl initiates perm(makeRef(HCP)),
  pow(imakeRef(HCP)), perm(imakeRef(HCP)) if badRef;

%%encrypt referral once after signing
encryptRef(System) generates iencryptRef(System);
isignRef(System) initiates refSigned, perm(encryptRef(System)),
  pow(iencryptRef(System)), perm(iencryptRef(System));
iencryptRef(System) terminates perm(encryptRef(System)),
  pow(iencryptRef(System)), perm(iencryptRef(System));
```

Figure 7: Generation and Consequence relations for the *create referral* phase of *Make Referral* model.

5 Results

The result (answer set, in the language of ASP) provides a rich database of information in form of ordered traces of events. The use of queries, as experience from databases would suggest, provides a means of examining the traces for any system behaviors that might be of interest. The properties being investigated could be expressed as facts and/or rules. The results can then be interpreted and decisions or actions taken appropriately.

```
130 %%send referral once after encryption
131 sendRef(HCP) generates isendRef(HCP);
132 receiveRefwP(HCP) generates ireceiveRefwP(HCP);
133 enforcePolicy(System) generates iapplyPolicy(HCP) if
        hasRefwP(HCP);
134 iencryptRef(System) initiates refEncrypted, perm(sendRef(HCP)),
135   pow(isendRef(HCP)), perm(isendRef(HCP));
136 isendRef(HCP) terminates perm(sendRef(HCP)),
137   pow(isendRef(HCP)), perm(isendRef(HCP));
138
139 isendRef(HCP) initiates
        perm(receiveRefwP(HCP)),pow(ireceiveRefwP(HCP)),
140   perm(ireceiveRefwP(HCP)), hasRef(Patient),hasRefwP(HCP);
141
142 %%set states to indicate the presence of referrals
143 isendRef(HCP) initiates hasRef(Patient),hasRefwP(HCP),
144   perm(iapplyPolicy(HCP)), pow(iapplyPolicy(HCP));
145 %%security policies should be applied by the HCP upon receiving
        referrals
146 iapplyPolicy(HCP) initiates policyApplied(HCP) if
        hasRefwP(rHCP);
```

Figure 8: Generation and Consequence relations for the *send referral* phase of *Make Referral* model.

```
observed(create_patientRef,i00).
observed(login(sHCP),i01).
observed(loginChal(sys,sHCP),i02).
observed(logindl,i03).
observed(selectAction(sHCP),i04).
observed(makeRef(sHCP),i05).
observed(checkRef(cHCP),i06).
observed(checkRefdl,i07).
observed(signRef(sys),i08).
observed(encryptRef(sys),i09).
observed(sendRef(sHCP),i10).
observed(receiveRefwP(rHCP),i11).

#hide.
#show occurred(E,I).
```

Figure 9: Verifying the expected sequence of observed events for the *Make Referral* model.

Query formulation and trace construction are intimately tied up. However, before traces can be generated, the program must be grounded, which means being explicit about the meaning of variables and time instants, defining precisely how many there are, which in turn determines the length of the trace. For time instants $t_i : 0 \leq i \leq n$, we define the following three rules: `instant(`t_i`)`, `next(`t_i`,` t_{i+1}`)` and `final(`t_n`)`, denoting each ground instant of time, relative order and final state, respectively. The grounding information is provided in a domain file which is passed to the ASP translator along with the instAL description file. We do not provide details here due to space.

The general trace program generates the answer sets containing all possible combinations of n exogenous events, but by the addition of constraints, the answer sets can be limited to those containing desired traces. This is expressed in the form of a query specification. For example, figure 9 shows how the answer set can be constrained to the consequences of specific observed events[1].

The result, as listed in figure 10 verifies the correctness of the model, in that first of all, events occurred in the expected sequence, subject to the constraint `occurred(`E`,` I`)`, which is read as event E occurred at instant I.

[1]Real world events are tagged `observed` in traces, while institutional ones are tagged `occurred`.

```
Answer: 1
occurred(create_patientRef,i00) occurred(login(sHCP),i01)
occurred(loginChal(sys,sHCP),i02) occurred(logindl,i03)
occurred(selectAction(sHCP),i04) occurred(makeRef(sHCP),i05)
occurred(checkRef(cHCP),i06) occurred(checkRefdl,i07)
occurred(signRef(sys),i08) occurred(encryptRef(sys),i09)
occurred(sendRef(sHCP),i10) occurred(receiveRefwP(rHCP),i11)

occurred(ilogin(sHCP),i01)
occurred(iloginChal(sys,sHCP),i02) occurred(ilogindl,i03)
occurred(iselectAction(sHCP),i04)
occurred(imakeRef(sHCP),i05) occurred(icheckRef(cHCP),i06)
occurred(icheckRefdl,i07) occurred(isignRef(sys),i08)
occurred(iencryptRef(sys),i09) occurred(isendRef(sHCP),i10)
occurred(ireceiveRefwP(rHCP),i11)
SATISFIABLE
```

Figure 10: Output of the model verification query for the *Make Referral* model.

This is a way of testing the correctness of the model. With this, the model can be used to investigate any behaviour of interest which in this case could be verification of occurrence of violation events, effects of such violations, and perhaps the security state of the system after or before the occurrence of certain events.

In this scenario, the criteria for a successful and secure process would be that the countermeasure events were successfully observed at the proper time instants. We can therefore examine the states of the institution at the final instant to see if those conditions or countermeasures actually holds. Such a query is written as a rule:

```
success:- holdsat(chalSuccess,F),
  holdsat(goodRef,F),
  holdsat(refSigned,F),
  holdsat(refEncrypted,F),
  holdsat(policyApplied(rHCP),F),

  not holdsat(misdCheck,F),
  not holdsat(chalSkipped,F),
  not holdsat(loginAttacked,F),
  not holdsat(policyViolated(rHCP),F),

  final(F).
```

These states should be reachable and remain true up to the final instant

States that are not expected to be true at the finale instant

defines the final instant of institutional time

It is possible to see traces of events that happened before a particular state of the system. For instance, the following query shows traces before the state `loginCompromised`.

```
happened(E,I0) :-
  holdsat(loginCompromised, i03),
  occurred(E,I0), before(I0,i03),
  instant(I0).
```

This ensures that events at instant **I0** occur before event at instant **i03**

This query provides the following result:

The security designer can also know the consequences of a violation event occurring. Figures 11 and 12 show the query specification and the resulting traces.

5.1 Generalization of case study

So far, we have used our case study to illustrate how validation and verification of security requirements can be effectively carried out at design time. We have only used a few examples since we cannot do exhaustive investigation of all properties here. However, our approach would work efficiently for analyzing various kinds of security requirements. Although a lot of work has been done on using misuse case approach to eliciting security requirements ([22, 21, 3, 18, 7]), none of these has provided a computational means of verifying the security requirements elicited. This problem also applies to other approaches for security requirements

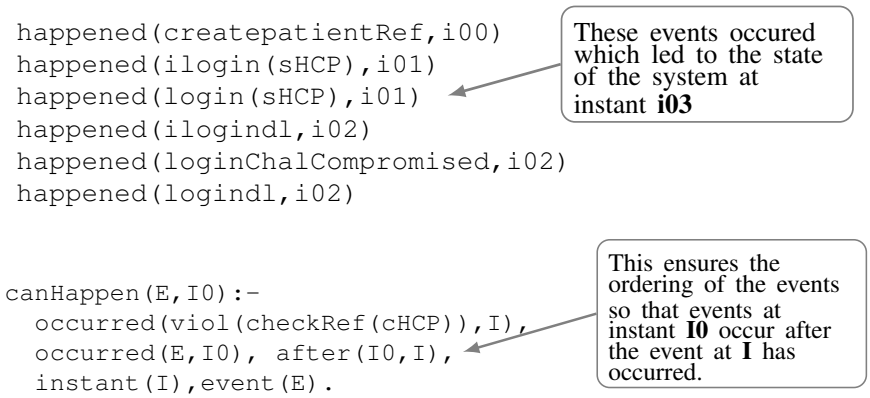

Figure 11: Consequences of violation - Query.

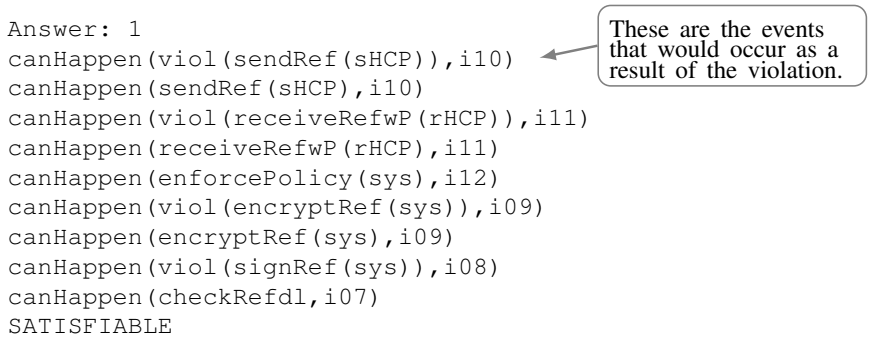

Figure 12: Consequences of violation - Result.

elicitation. Since our idea captures the system specification with security requirements, it enables both system and security designers to capture the system-to-be in terms of actors or roles, events, and rules through the use of permissions, powers, and obligations.

6 Related work

The literature abounds with methods for the elicitation of security requirements. We try to put forward a representative selection to contrast with what we have presented here. SQUARE [16] was aimed at integrating security requirements engineering into software development process. It seeks to build security concepts into the early stages of the development cycle through a model consisting of nine steps of activities. However, the method does not provide an explicit way for verification of the security requirements. The goal-oriented approach, KAOS [24] is concerned with the goals to be achieved by the system-to-be, and focuses less on the elicitation of security requirements. Although linear real-time temporal logic is used to specify goals, properties, and conditions, there is no clear methodology for verification. Secure i^* [14] is based on the i^* modeling framework [28] for modeling and analyzing security trade-offs. It aims to analyze the problem of modeling security at an organizational level. This is very close to our concern since attention is placed on the security of relevant social interactions. It is founded on the notion of permission, delegation, and trust with its formalization in first-order predicate logic. Our approach differs from this in the sense that our framework is inspired by deontic logic in which we monitor permissions, empowerments, and obligations.

Model checking has been applied in some cases. For instance Ma et al. [13] and Kikuchi et al. [11] used model checking to validate information system security policies. The system behaviours were modelled as Kripke structures, while the system properties were described in linear temporal logic (LTL) formula. Security policy verification was achieved by applying the model checker SPIN. Another approach, based on decision tables, an incremental policy validation method, is presented in [9]. Both these approaches require a high level of background in logic to be able to interact with the specifications. Also, Wahsheh et al. [25] show how

Prolog can be used to verify system correctness with respect to policies for high assurance computing systems. In contrast, our approach is based on answer set programming (ASP) which has a number of advantages over traditional logic programming languages for implementing event based systems. ASP offers a purely declarative language, offering ways to model specifications without allowing the programmer to control the search; ASP is as expressive as many other non-monotonic logics, yet it provides a simpler syntax and well-developed and efficient implementations; ASP is more expressive than propositional and first-order logic, allowing us to elegantly encode causality and transitive closure [19]. ASP is intuitive, requires less background in logic, and its semantics is robust to changes in the order of literals in rules and rules in programs. Also in comparison to Prolog where solutions are computed by query answering which amounts to proof search, ASP solutions are encoded in answer sets; that is, in models, hence model-finding, rather than proof-finding [4]. This way, it offers us the opportunity to make verifications in context, giving us answer models consisting of a sequence of traces which can be more easily interpreted in the problem context.

7 Conclusion

This has been a journey from the initial design of a misuse case analysis to its verification and validation through the use of an institutional framework. The need for early consideration and inclusion of security features in the design and development of systems cannot be overemphasized. Several approaches have been investigated to ensure the ease of security requirements elicitation and analysis. However, the fact still remains that it is difficult for humans to reason about all the possibilities of the occurrence of security threats. Also, it is not possible for humans to reason about the possible behavior of the system when the security countermeasures are implemented at design time. These therefore call for the need to have a reasoning mechanism by which security analysts and system designers can thoroughly analyze the possibilities of security threats. This tool and approach has proven to be helpful in this regard due to its ability to provide a kind of a database of all traces of events captured in the model from which the analyst can verify any system property of interest.

8 Acknowledgment

We appreciate Dr. Marina De Vos (University of Bath) and Dr. Tina Balke (University of Surrey) for their support in sorting out InstAL issues. Finally, I also thank the Petroleum Technology Development Fund (PTDF) Nigeria who is sponsoring my PhD research.

References

[1] Amanda Andress. *Surviving security: how to integrate people, process, and technology*. AUERBACH PUBLICATIONS - A CRC Press Company, 2nd edition edition, 2004.

[2] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[3] Fabricio A. Braz, Eduardo B. Fernandez, and Michael VanHilst. Eliciting security requirements through misuse activities. *Database and Expert Systems Applications, International Workshop on*, 0:328–333, 2008.

[4] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54:92–103, December 2011.

[5] Owen Cliffe. *Specifying and Analysng Institutions in Multi-Agent Systems using Aswer Set Programming*. PhD thesis, University of Bath, UK, 2007.

[6] Owen Cliffe, Marina De Vos, and Julian Padget. Answer set programming for representing and reasoning about virtual institutions. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *Computational Logic in Multi-Agent Systems*, volume 4371 of *Lecture Notes in Computer Science*, pages 60–79. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-69619-3_4.

[7] Eduardo Fernandez, Michael VanHilst, Maria Larrondo Petrie, and Shihong Huang. Defining security requirements through misuse actions. In Sergio Ochoa and Gruia-Catalin Roman, editors, *Advanced Software Engineering: Expanding the Frontiers of Software Technology*, volume 219 of *IFIP International Federation for Information Processing*, pages 123–137. Springer Boston, 2006. 10.1007/978-0-387-34831-5_10.

[8] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.

[9] A. Graham, T. Radhakrishnan, and C. Grossner. Incremental validation of policy-based systems. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 240 – 249, june 2004.

[10] Andrew J.I. Jones and Marek Sergot. A Formal Characterisation of Institutionalised Power. *ACM Computing Surveys*, 28(4es):121, 1996. Read 28/11/2004.

[11] Shinji Kikuchi, Satoshi Tsuchiya, Motomitsu Adachi, and Tsuneo Katsuyama. Policy verification and validation framework based on model checking approach. In *ICAC*, page 1. IEEE Computer Society, 2007.

[12] Richard Kissel, Kevin Stine, Matthew Scholl, Hart Rossman, Jim Fahlsing, and Jessica Gulick. Security considerations in the system development life cycle. Technical Report 800-64 Revision 2, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, October 2008. http://csrc.nist.gov/publications/nistpubs/800-64-Rev2/SP800-64-Revision2.pdf Accessed 7 February 2012.

[13] Jianli Ma, Dongfang Zhang, Guoai Xu, and Yixian Yang. Model checking based security policy verification and validation. In *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, pages 1 –4, may 2010.

[14] Fabio Massacci, John Mylopoulos, and Nicola Zannone. *Handbook of Ontologies for Business Interaction*, chapter An Ontology for Secure Socio-Technical Systems, pages 188–207. 2008.

[15] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In Dong Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer Berlin / Heidelberg, 2006. 10.1007/11734727_17.

[16] Nancy R. Mead and Ted Stehney. Security quality requirements engineering (square) methodology. *SIGSOFT Softw. Eng. Notes*, 30:1–7, May 2005.

[17] T. Okubo, H. Kaiya, and N. Yoshioka. Effective security impact analysis with patterns for software enhancement. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 527 –534, aug. 2011.

[18] T. Okubo, K. Taguchi, and N. Yoshioka. Misuse cases + assets + security goals. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 3, pages 424 –429, aug. 2009.

[19] Enrico Pontelli. Answer set programming in 2010: A personal perspective. In Manuel Carro and Ricardo Pea, editors, *Practical Aspects of Declarative Languages*, volume 5937 of *Lecture Notes in Computer Science*, pages 1–3. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-11503-5_1.

[20] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley Publishing, Inc., 2000.

[21] Guttorm Sindre. Mal-activity diagrams for capturing attacks on business processes. In Pete Sawyer, Barbara Paech, and Patrick Heymans, editors, *Requirements Engineering: Foundation for Software Quality*, volume 4542 of *Lecture Notes in Computer Science*, pages 355–366. Springer Berlin / Heidelberg, 2007.

[22] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements by misuse cases. In *TOOLS (37)*, pages 120–131. IEEE Computer Society, 2000.

[23] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.

[24] Axel van LAMSWEERDE. Engineering requirements for system reliability and security. In Manfred Broy, Johannes Grnbauer, and Charles Antony Richard Hoare, editors, *Software system reliability and security*, volume 9 of *NATO Security through Science Series*, pages 196–238. IOS Press, 2007.

[25] Luay A. Wahsheh, Daniel Conte de Leon, and Jim Alves-Foss. Formal verification and visualization of security policies. *JCP*, 3(6):22–31, 2008.

[26] M.S. Ware, J.B. Bowles, and C.M. Eastman. Using the common criteria to elicit security requirements with use cases. In *SoutheastCon, 2006. Proceedings of the IEEE*, pages 273 –278, 31 2005-april 2 2006.

[27] Laurie Williams, Tao Xie, Andy Meneely, Lauren Hayward, and Jason King. itrust medical care requirements specification, October 2011. `http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=requirements`.

[28] Eric Siu-Kwong Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, Toronto, Ont., Canada, Canada, 1996. UMI Order No. GAXNN-02887 (Canadian dissertation).