Security Policy Verification for Internal Information Security Threats using Answer Set Programming

Gideon Bibu*[1] and Julian Padget[2]
[1]Department of Computer Science University of Jos, Nigeria
[2]Department of Computer Science University of Bath, UK
Contact: dadikg@unijos.edu.ng

ABSTRACT

The adoption of technologies that allow for electronic processes and storage of information has also resulted in the growing concern for the security of the information system. The security challenges come in different ways including technologies, processes, and humans through their interactions with one another. While technical solutions abound for technology-related security problems, the security challenges that arise from processes and humans interactions are usually addressed by the application of appropriate organisational security policies. Hence the design and specification of such policies is crucial to the achievement of the desired security level in the organisation. Therefore, it is very crucial to verify the policies against the system actors' interactions when designing them. In this paper, we present a computational reasoning mechanism that helps with the design-time static verification of security policies. Using a simple example that allows us to illustrate the principles of the approach, we show how processes and interactions are represented in Inst$\mathcal{AL}$, an action language based on answer set programming. The reasoning mechanism of Inst$\mathcal{AL}$ is based on an institutional (normative) framework in which actions lead to changes in state over a sequence of time instances resulting in a collection of event traces. We demonstrate how the traces can then be used to verify desired properties through examples.

KEYWORDS: Information security, Policy, ASP, Institution, verification

Introduction

The security of information and systems is a matter of major concern to most organisations. This is as a result of the adoption of technologies that allow for electronic processes and storage of information. As more and more highly sensitive content is processed and managed electronically, the need for more secure systems and efficient and compliant processes becomes increasingly critical. Security threats to information and systems are generally categorised into external and internal threats, underlining the fact that attacks could be launched from both inside and outside the organisation. External threats are usually addressed by creating a "perimeter" around the organisations' assets which provide defences against the perceived external attacks. This does not address the insider threat problem which is more subtle than the external threat problem. The internal threat manifests itself through many ways, including users' behaviours that violate security policies. These behaviours could either be malicious or non-malicious, however, whatever the intention, these behaviours always have negative impact on the organisation. A malicious insider is potentially more dangerous than an outside attacker. This is because an insider has legitimate and privileged access to information resources, practical knowledge of the organisation and its processes, and knowledge of the location of valuable and critical assets. An important step in mitigating the risks posed by insider attacks is to carefully construct an enterprise wide security policy that addresses usage and security issues (Mike and Kemp, 2005) in addition to the implementation of necessary security threat mitigation mechanisms.

In this paper, we use the term Policy to refer to *security policy*. Policy has been used in many ways to address security issues consisting of confidentiality, integrity, and availability. Policies can be sets of rules that define choices in behaviour of actors in terms of the conditions under which predefined operations or actions can be invoked. Security policies provide the first step in preventing insider abuse in organisations where expected security behaviours are usually presented in the form of some high level security policies. However, the problem with policies is that compliance cannot be guaranteed and hence the likelihood for the security threats to persist, despite the existence of policies that should have ensured proper behaviour by users. With this in mind, it is important that security policy designers are able to verify that the policy is consistent with the operations of the organisation so that everyday organizational processes do not stand in the way of compliance. The process of verification would help in the refining of the policy before it is deployed. This paper presents a mechanism for expressing and verifying security policies. The methodology is based on an institutional framework (Cliffe *et al.*, 2007b) 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 policies are not adhered to. The framework is implemented in an action language Inst$\mathcal{AL}$ (Cliffe *et al.*, 2007b) which is based on the answer set semantics of answer set programming (Baral, 2003; Gelfond and Lifschitz, 1988). In section 2, we reviewed the literature for related work. Section 3 describes a motivating scenario, examination paper security, which we use to demonstrate the solution approach. In section 4, we describe the institutional framework briefly but sufficiently for understanding the approach. Policy representation in Inst$\mathcal{AL}$ is presented in section 5 using the scenario described in section 3. Results are presented in section 6 along with the discussion of the analysis of the results and we finally conclude in section 7.

Considered as a fundamental protection strategy in information systems, security policy has received much attention from researchers who have approached its various problems in different ways in different domains. Bandara *et al.*, (2003) presented a translation of the policy and system behaviour specifications to a formal notation based on standard event calculus and used reasoning techniques to identify conflicts. While their approach produced useful results, it is not easy translating specifications into standard event calculus logic. Model checking approach has also been applied to some of the problems. For instance Ma *et al.*, (2010) and Kikuchi *et al.*, (2007) used model checking to validate information system security policies. The system behaviours were modelled as Kripke structure 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 was presented in Graham *et al*., (2004). These approaches require a high level of background in logic to be able to interact with the specifications. Also, Wahsheh *et al*., (2008) show how Prolog was used to verify system correctness with respect to policies for high assurance computing systems. However, our solution 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 (Pontelli, 2010). 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 (Brewka *et al*., 2011). This way, it offers us the opportunity to make verifications in context, giving us answer models consisting of a sequence of traces which we can easily interpret. Although ASP has been used in authorization specifications (Wang and Zhang, 2007; Lee, Wang, and Zhang, 2015), these works are focused on access control policies without considering the "social" dimension of the security vulnerabilities.

In order to illustrate our approach to the problem of security policy verification, we present here an example scenario. The motivation for this example is that it is a part of a process which has clearly defined participant roles and security policies associated with those roles. It is simple enough for the purpose of illustration and also complete in the sense that it consists of the system goal, actors, events, and the process. The process is also one that can easily be understood and related to.

The model represents a segment of the process of preparing examination papers. The model consists of the roles: head of department (hod), examination officer (eo), course lecturer (lect), and the resources: exams server (svr) and a question paper (paper01). The *lect* uploads the exams file to the secure server dedicated for exam preparation. The *svr* sends acknowledgement to the *hod.* The *hod* acknowledges and triggers *eo* to download and print the paper. The *eo* who is in charge of the next and final stage of the preparation, downloads the paper from the server, prints the required quantity for each paper and packages them securely ready to be administered. From an insider perspective, the *eo* could be responsible for misuse, where misuse is any action or behaviour that may lead to the leakage of the exam paper, thereby compromising the confidentiality and integrity of the exams. These behaviours would be those that violate the following security rules (policies):

pol 1. The *hod* is not permitted to download paper from the server

pol 2. The *eo* can only access the exam files and not store them on any external storage device

pol 3. *eo* should delete his/her copy of the paper once it is printed

pol 4. A printed paper is considered secure when its instance is only available in the server at the end of its lifecycle which is the printed state.

This example therefore clearly spells out the actors involved, in terms of roles and the expected behaviour that would preserve the integrity and confidentiality of the examination paper. We shall be using this example in the subsequent sections of this paper.

Materials and Methods

Our approach is based on the definitions of institutional framework in (Cliffe *et al.*, 2007b) where an electronic institution is described as a multi-agent system and 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. Institutional frameworks therefore provide a mechanism to capture and reason about "correct" and "incorrect" behaviour within a defined social context, which in this case is security. This therefore provides a way to explore security vulnerabilities arising from behaviours of various actors (humans, systems, processes) as they interact to achieve their goals. In applying this framework to the problem of security, the system of interacting actors which consists of humans and systems and their associated interactions (processes), and the security policies which define the expected behaviour of the actors fits into the description for the framework.

An institutional framework is formally defined as a 5-tuple $\mathcal{I} := \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ where $\mathcal{E}$ is a set of events, $\mathcal{F}$ a set of fluents, $\mathcal{C}$ a set of causal rules, $\mathcal{G}$ a set of generation rules and an initial state Δ.

The agents of change in the (security) model are the actors, and it is the actions they take that constitute the events, which are then interpreted by the institutional rules and bring about a change in the institutional state ($\mathcal{S}$). Actions and institutional states are modeled as *events* ($\mathcal{E}$) and a set of *fluents* (i.e. $\mathcal{S} = \{\mathcal{F}\}$), where the state is a record of the effects of previous actions, and extant obligations, powers and permissions. Expressing this abstractly, let $s_i \in \mathcal{S}$ denote a state of the system (model) and $e_i \in \mathcal{E}$ denote an event that affects the system (model), then an actor interacting with the model generates a *trace*:

$$s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \ldots s_F \in \tau$$

that characterises a single trace system interaction. However, for verification we need all possible traces. Therefore, given an initial system state s_0 and a state transformer function $\tau : \mathcal{S} \times \mathcal{E} \to 2^{\mathcal{S}}$, we can compute next states exhaustively. The transformer function comes in two parts: the generation relation, denoted $\mathcal{G}$, and the consequence relation $\mathcal{C}$. $\mathcal{G}$ is responsible for recognising relevant real world events and turning them into institutional events, and also for ensuring that all the institutional events that ensue from an institutional event are also generated. $\mathcal{C}$ is responsible for the addition and deletion of fluents in the institutional state, arising from all the events identified by $\mathcal{G}$. Hence, τ is the application of $\mathcal{C}$ to the events arising from the transitive closure of $\mathcal{G}$. We illustrate all of these elements, expressed in the Inst*AL* specification language, using the case study, in section 3.1.

An answer set programming (ASP) 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 being investigated.In using ASP for reasoning about institutions, Cliffe *et al.*,(2007b) shows 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 (Gelfond and Lifschitz,1988)correspond to the traces of the institutional framework. This is seamlessly achieved through the action language Inst$\mathcal{AL}$ by mapping the action language to an answer set program (Cliffe *et al*., 2007a).

Policy representation in Inst

In our example, three classes of policies can be identified:

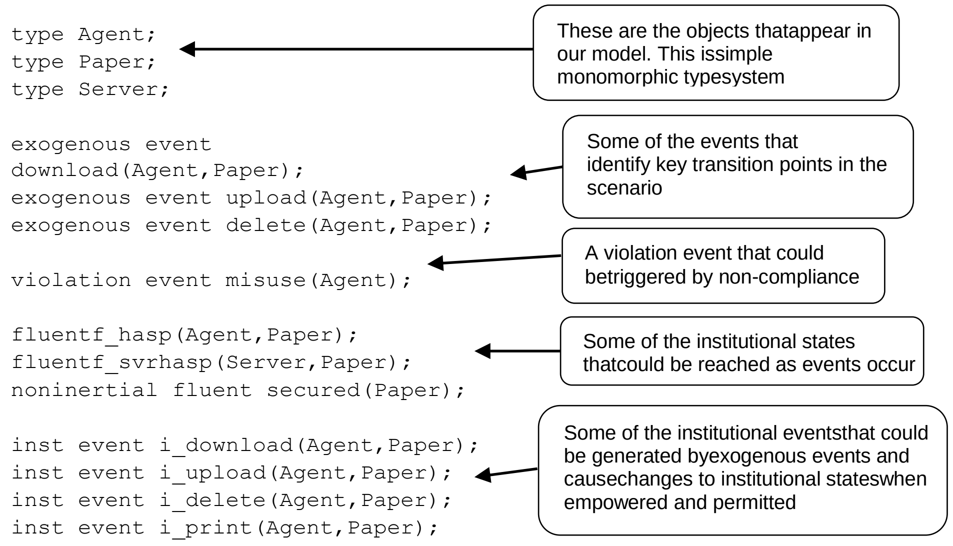
i. policies that prohibit certain actions (pol1, pol2),
ii. policies that require events to occur in a certain order (pol3), and
iii. policy that state the final expectation of the system (pol4).

Using annotated fragments of the code, we describe how the specification of these policies are achieved in Inst$\mathcal{AL}$.

We start by presenting the declarations of some of the key features of the specification. These include the **type**, **events**, and **fluent** declarations (figure 1). The declaration of events consists of;

- *Exogenous events*which express all observable real world events in the model. These events may generate institutional events and cause changes to the institutional state.
- *Institutional 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 the failure to satisfy an obligation.

These would enhance understanding as we describe the specifications of the policies.

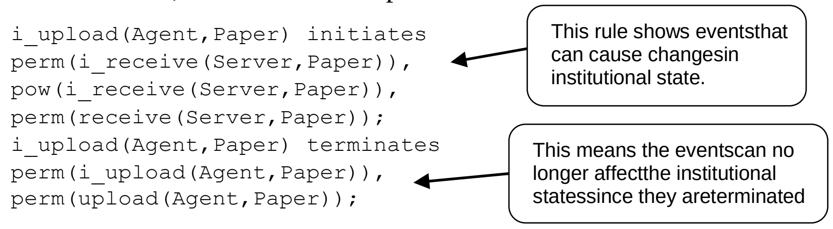


```
type Agent;
type Paper;
type Server;

exogenous event
download(Agent,Paper);
exogenous event upload(Agent,Paper);
exogenous event delete(Agent,Paper);

violation event misuse(Agent);

fluentf_hasp(Agent,Paper);
fluentf_svrhasp(Server,Paper);
noninertial fluent secured(Paper);

inst event i_download(Agent,Paper);
inst event i_upload(Agent,Paper);
inst event i_delete(Agent,Paper);
inst event i_print(Agent,Paper);
```

These are the objects thatappear in our model. This issimple monomorphic typesystem
Some of the events that identify key transition points in the scenario
A violation event that could betriggered by non-compliance
Some of the institutional states thatcould be reached as events occur
Some of the institutional eventsthat could be generated byexogenous events and causechanges to institutional stateswhen empowered and permitted

Figure 1: Types, predicates and events

With respect to the institutional framework, events or actions can occur and hence affect the institutional state only when they are permitted. Therefore, using the causal rules, we are able to express this as



```
i_upload(Agent,Paper) initiates
perm(i_receive(Server,Paper)),
pow(i_receive(Server,Paper)),
perm(receive(Server,Paper));
i_upload(Agent,Paper) terminates
perm(i_upload(Agent,Paper)),
perm(upload(Agent,Paper));
```

This rule shows eventsthat can cause changesin institutional state.
This means the eventscan no longer affectthe institutional statessince they areterminated

It is also important to note that events that are not empowered cannot occur even if permitted. This applies basically to institution events since exogenous events are empowered by default. By empowerement we mean the capability of an event to be brought about (generated) in the institution. Prohibitions are not explicitly represented, they are rather implicitly represented by the absence of permission for that event to occur. Therefore a prohibited event is simply not permitted and hence would trigger a violation if it is performed. In our example, policies that prohibit certain actions such as such as pol1 is represented by not permitting the hod to perform a download operation. It is therefore expected, in verification that a download operation by the hod should trigger a violation event. We express this as

i_download(hod,paper01) generates misuse(hod);

Policies that require events to occur in a certain order are captured in Inst$\mathcal{AL}$ using obligations. Obligations are treated as fluents and expressed as obl(event,deadlineevent,violationevent).

This means that *event* is expected to occur before *deadline event* occurs else it triggers a *violation event*. In our example, pol3is therefore expressed in terms of obligations as follows;

```
initially
obl(delete(eo,paper01),
end of process(paper01),
misuse(eo));
```

if the event `end of process` happens before `delete`, violation event `misuse` will be triggered

It is convenient to be able to declare these as initial states of the institution. This enables it to persist in the institution until it is terminated.

Pol6 is achieved by evaluating the states of the institution and expressing this as:
always secured(paper01) when not has(eo,paper01),
not has(hod,paper01), fsvrhas(svr,paper01), fprinted(paper01).
This is interpreted as: paper01 *is secure when it is not the case that neither* eo *nor* hod *has* paper01 *but* svr *has* paper01 *and* paper01 *is printed*. If any of the stated conditions fails, the security of the paper would be questionable. The lifecycle of the paper here is considered to be the interval between when the paper was uploaded by the lect and when the paper has been printed by the eo. Therefore the point of evaluating this condition is at the end of the lifecycle which is specified in the institution as an exogenous event deadline which generates the institutional event end of process (paper01).

Results and Discussion

Having set out the institution framework and the domain specification, we can use the model to examine the traces for both expected and unexpected behaviours. The Inst*AL* reasoning tool can generate all the possible traces for the encoded institution framework. Therefore in the analysis of the traces, we focus on the ones that are of interest to us. In this case we focused on those that bother on policy compliance.

The analysis is performed through a query file where we can restrict the traces to the desired events pattern using the fact observed(Event, Time). Time here is not in the sense of real time but in the sense of order in which events occur relative to each other. The institutional timing starts with the creation event which creates the institution at time instance i00. Timing of subsequent events follows from here. To investigate policy compliance, the conditions for non-compliance are presented in form of rules as follows:

Where each of the policies states the kind of events or event sequences that would be considered a violation of the policy. According to policies pol1-pol2, it is considered non-compliance whenever any of the events occur at any time instant *I*. Policy pol3 is specific about the order in which the events should occur. Event at time instant *T1* is expected to occur before the event at time *T2*. Policy pol4 considers it non-compliant if it is the case that any of the agents A (i.e. eo and hod) has the paper at the final institutional time instant F. Taken that, the expected sequence of events is:

```
noncomp(eo,I) :- occurred(download(hod,P),I),
         instant(I).                   (pol1)
noncomp(hod,I) :- occurred(download(eo,P),I),
       instant(I).                     (pol2)
noncomp(A) :- occurred(download(eo,P),T1),
       occurred(delete(hod,P),T2),
       T1<T2, instant(T1), agent(A),
       instant(T2).                    (pol3)
noncomp(A,F) :- holdsat(f_hasp(A),F),
       agent(A),final(F).              (pol4)

observed(createsecurexams,i00).
observed(upload(lect,paper01),i01).
observed(receive(svr,paper01),i02).
observed(download(eo,paper01),i03).
observed(print(eo,paper01),i04).
observed(delete(eo,paper01),i05).
observed(deadline,i06).

#hide.
#show noncomp(A,I).
#show noncomp(hod,I).
#show noncomp(A).
#show noncomp(A,F).
#show occurred(viol(A),I).
```

These rules define non compliance, where `instant(I)` indicate a sense of time

None of the agents, `eo` and `hod` should be in possession of the paperat the end of theinstitution

Events are expected to happen in this order

hides all answer sets

Outputs only theseresults

Such a query specification will not return any result as expected. This implies that the policies were complied with by the actors. However, testing for noncompliance will mean altering the observed events. For example

.

.

observed(download(eo,paper01),i02).

.

.

Would mean eo downloading paper before hod deletes it, contrary to pol3. The result of this is

```
Answer: 1
occurred(viol(deadline),i07)
occurred(viol(print(eo,paper01)),i05)
occurred(viol(download(eo,paper01)),i03)
noncomp(hod)
noncomp(eo)
SATISFIABLE
```

These lines show theevents that areviolated as a result of non-compliance of

The presence of theseelements in the answerset tells us that the actions of `hod` and `eo` are non-compliant

which clearly shows violation events with their corresponding time instances and also the non-compliant agents responsible for the violations.

Another example we can test for is how the system is affected when a required event does not happen at all. For instance, following from pol3, we can see what happens if eo fails to delete the paper after printing. This means removing observed (delete(eo,paper01),i05) from the observed events. This results in the following;

```
Answer: 1
occurred(viol(deadline),i06))
noncomp(eo,i05)
insecure (paper01,i06)
SATISFIABLE
```

This shows that the event deadline occurring at time instant i06 is a violation due to the non-compliance of the agent eo at time instant i05 making the paper insecure at the last time instant i06.

Also, let us assume that in violation of pol2, the eo uploads the paper (we take upload here to mean storing the file on a device or even on email). We add the fact observed(upload(eo,paper01),i04) to the list of observed events. This results in

```
Answer: 1
occurred(viol(deadline),i06))
noncomp(eo,i05)
insecure(paper01,i06)
SATISFIABLE
```

The presence of this item indicates the paper is insecure at the time instant `i06`

which also means that a violation occurred at time instant i04 triggered by the agent eo and the paper is insecure at the instant i06. Further properties of the policy based system can be tested in a similar way to verify the policy before implementation.

Conclusion

Organisations deploying computing and information systems would usually formulate security policies that would ensure that the processes undertaken by actors (employees and systems) are such that preserve the security of the organisation's resources. While computer system and networks security policy research has received much attention, organisational security policies have received less. This work therefore offers a contribution in this direction.

Another contribution is the provision of an intuitive methodology for the representation of events (organisational processes) which allows us to verify security policies guarding the secure enactment of processes. Our approach is based on an institutional framework in which enactment of events and the consequent changes in states lead to sequence of traces which provide a "database" that can be queried for desired properties. We used an action language Inst*AL* to code the system specifications and describe the query properties in answer set programming. We showed the results of our some verification using a simple example. The proposed solution will therefore be a useful tool for organisational security policy designers.

In the future, we hope to extend this work by considering multiple institutions. By this, we aim to analyse security policies across organisational boundaries since organisational security policies differ from one organisation to another and where they interact, there is also potential for security loopholes.

References

Bandara, A. K., Lupu, E. C., and Russo, A. (2003). Using event calculus to formalise policy specification and analysis. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on* (pp. 26-39). IEEE.

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press.

Brewka, G., Eiter, T., and Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92-103.

Cliffe, O., De Vos, M., and Padget, J. (2007a). Answer set programming for representing and reasoning about virtual institutions. In *Computational Logic in Multi-Agent Systems* (pp. 60-79). Springer Berlin Heidelberg.

Cliffe, O., De Vos, M., and Padget, J. (2007b). Specifying and reasoning about multiple institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II* (pp. 67-85). Springer Berlin Heidelberg.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Thiele, S. (2008). Engineering an incremental ASP solver. In *Logic Programming* (pp. 190-205). Springer Berlin Heidelberg.

Gelfond, M., and Lifschitz, V. (1988). The stable model semantics for logic programming. In *ICLP/SLP* (Vol. 88, pp. 1070-1080).

Gelfond, M., and Lifschitz, V. (1998). Action languages. *Electronic Transactions on AI*, *3*(16). 193–210

Graham, A., Radhakrishnan, T., and Grossner, C. (2004). Incremental validation of policy-based systems. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on* (pp. 240-249). IEEE.

Kemp, M. (2005). Barbarians inside the gates: addressing internal security threats. *Network Security*, *2005*(6), 11-13.

Kikuchi, S., Tsuchiya, S., Adachi, M., and Katsuyama, T. (2007). Policy verification and validation framework based on model checking approach. In*Autonomic Computing, 2007. ICAC'07. Fourth International Conference on* (pp. 1-1). IEEE.

Lee, J., Wang, Y., and Zhang, Y. (2015). Automated reasoning about XACML 3.0 delegation using answer set programming. In *CEUR-WS*.

Ma, J., Zhang, D., Xu, G., and Yang, Y. (2010). Model checking based security policy verification and validation. In *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on* (pp. 1-4). IEEE.

Pontelli, E. (2010). Answer set programming in 2010: A personal perspective. In Carro, M. and Peña, R. eds, *Practical Aspects of Declarative Languages* (pp. 1-3). Springer Berlin Heidelberg.

Wahsheh, L. A., Leon, D. C. D., and Alves-Foss, J. (2008). Formal verification and visualization of security policies. *Journal of Computers*, *3*(6), 22-31.

Wang, S., and Zhang, Y. (2007). Handling distributed authorization with delegation through answer set programming. *International Journal of Information Security*, *6*(1), 27-46.