Modelling Dynamic Access Control Policies for Web-Based Collaborative Systems*

Hasan Qunoo and Mark Ryan

School of Computer Science,
University of Birmingham, UK
{H.Qunoo,M.D.Ryan}@cs.bham.ac.uk

Abstract. We present a modelling language, called *X-Policy*, for web-based collaborative systems with dynamic access control policies. The access to resources in these systems depends on the state of the system and its configuration. The *X-Policy* language models systems as a set of actions. These actions can model system operations which are executed by users. The *X-Policy* language allows us to specify execution permissions on each action using complex access conditions which can depend on data values, other permissions, and agent roles. We demonstrate that *X-Policy* is expressive enough to model collaborative conference management systems. We model the EasyChair conference management system and we reason about three security attacks on EasyChair.

1 Introduction

Large conference management systems like iChair, WSAR, HotCRP and EasyChair are widely used to manage academic conferences. However, the size and the complexity of these systems make it hard to analyse their policies and their security properties. The policies of those systems are designed to preserve the system security and serve their desired purpose. Systems, however, can still fail some basic security properties. Users can compromise the system policy and its security properties by interactions of rules, co-operations between agents and multi-step actions. In EasyChair, the system is designed to collect a number (usually between 3 and 4) of reviewer's opinions of a submitted paper. These opinions determine whether a paper should be accepted or rejected. For the system rules to be fair, no single reviewer should be able to determine the outcome of a paper reviewing process by writing all three reviews of that paper. However, the intention of these rules can be breached by interaction of rules to allow a single user to write all the three reviews of a paper. For example, a single sub-reviewer manages to write all three paper reviews while all the agents still comply with the system rules.

In this paper, we present a simple yet expressive modelling language, called *X-Policy*, to model large web-based collaborative management systems. Our language enables us to:

* The long version[1] of this paper can be found at the authors' web pages.

S. Foresti and S. Jajodia (Eds.): Data and Applications Security XXIV, LNCS 6166, pp. 295–302, 2010.

- model dynamic systems as a set of atomic multi-assignment write actions.
- specify read actions that gives us the ability to specify who can read what.
- specify action executing policy as preconditions which the user has to satisfy to execute the action.

We also use ***EC*** as a case study for our language. We model ***EC*** in *X-Policy* and we reason about three security attacks on EasyChair using our model. The long version of this paper is available at [1].

1.1 Paper Structure

We detail in Section 2 the related work. In Section 3, we present the *X-Policy* language and the process of expressing the ***EC*** model using *X-Policy* language and formalism. We introduce a selection of ***EC*** actions with their execution permissions statements which are used in the security attacks on ***EC*** and EasyChair in Section 3.2. The conclusion and future work are in Section 4.

2 Related Work

Recently, there has been a plethora of languages and logics to express access control policies. These logics and languages try to solve various issues arising from decentralisation[2,3,4,5,6,7]. DeTreville was the first to propose a Datalog based security language called Binder[4]. Since then Datalog has become the foundation of recent logic-based access control policies like the RT family [7] and SecPAL[2]. Researchers are mainly attracted to Datalog[8] as they can start from a tractable and expressive language with the advantage of deducing trust relations effectively based on well developed logic programming concepts and deductive databases. Unfortunately, Datalog is stateless. Inherently, the ability of datalog-based languages to express dynamic access control policies is restricted. Cassandra[9], a Datalog-based language, has a separate mechanism to maintain the authorisation state by inserting and retracting "hasActivated" facts according to the policy rules.

Gurevich et. al. introduced Distributed Knowledge Authorisation Language DKAL[5] and DKAL2[6] that extend SecPAL's expressiveness. However, Cassandra, SecPAL, DKAL, DKAL2 and other authorisation languages lack the ability to express the dynamic aspect of access control where policies depend on and update the system state like those we have in ***EC***. They, also, cannot express the effect of actions as part of the language and it has to be hard-coded in an ad-hoc way. More recently, DyNPAL[3] aims to specify dynamic policies with the ability to specify the effect of executing these actions. DyNPAL allows conditional bulk insertion and retraction of authorisation facts with transactional execution semantics (either all or none are committed). However, DyNPAL's declarative nature and minimalistic approach make it hard to follow the control flow of the actions. Also the lack of parameter typing does not allow us to establish the relation between the agent who can execute an action and the

action itself. They tend to focus on answering the question "under what conditions can an action be executed?" rather than "under what conditions can an agent execute an action?". This is indeed necessary to enable us to define agent coalitions and establish which agent is executing an action. It allows us to detect attacks where we are interested in who can execute a set of actions rather than whether a set of actions can be executed regardless of the actors involved. RW framework[10], a precursor of *X-Policy*, can analyse the consequences of multi-agent multi-step actions by performing temporal reasoning. RW is both model checking based frameworks. However, RW, unlike *X-Policy*, cannot express the actions with multiple assignments needed to preserve the integrity constraints of the modelled system.

To the best of our knowledge, this is the first paper to model and analyse dynamic access control policy for a large web-based collaborative system with atomic actions like EasyChair.

3 Modelling EasyChair Conference Management System

We specify system operations as *X-Policy* programs[1] which can be either *write programs* that change the state of the system or *read programs* that give the user/agent the knowledge about the state of the system. Programs in *X-Policy* can not read and change the state of the system at the same time. Although this is formally a restriction, most actions in collaborative web-based systems are indeed either a read or write and rarely both. This is true for EasyChair in particular. We believe that this is a sensible heuristic for modelling web-based systems. Users are only enabled to perform only one operation per time.

A *read program* allows the user to know the value of a ground proposition by returning the value of that proposition to the user who executed the program. A *write program* allows the user to change the value of a set of ground propositions using assignment statements in the form $p(\overrightarrow{y}) := \top$; or $p(\overrightarrow{y}) := \bot$; where $p(\overrightarrow{y})$ is a ground proposition. We allow a proposition to occur at most once at the left of ":=". The assignment statements within the same program can be written in any order. Such an assumption result in making the programs effect independent from the state of the system and each program has the same effect at all the time. A program permission statement exec(g,u) defines the conditions for an agent u to execute a program g. These conditions are defined as propositional logic formulae using the ground propositions and logical connectors. The full syntax and semantics of *X-Policy* are details in the long version[1].

3.1 *EC* Model in *X-Policy* Formalism

In this section, we discuss the model ***EC***. We build ***EC***, a model of our understanding of EasyChair and restrict ***EC*** to a single conference system. We express the ***EC*** model in *X-Policy* formalism[1]. To model ***EC***, we define a number of predicates P. For a,b of type Agent, p of type Paper, P includes:

Chair-review-en()	Review menu is enabled for Chair to manage the reviews of papers.
PCM-access-reviews-en()	PC members can access (view) other papers reviews.
PCM-review-editing-en()	PC members can add/modify reviews.
PCM-review-menu-en()	Review menu is enabled for PC members.
Review-assig-enabled()	Review assignments enabled.
Sub-anonymous()	Submissions are anonymous. The name of authors are obscured.
View-sub-title-permitted()	PC members can view the submissions title of all the papers.
Auth(p,a)	a is an author of p.
Chair(a)	a is the chair of the PC.
Conf-of-interest(p,a)	a has a conflict of interest with the p.
Decided-subrev(p,a,b)	b has accepted or rejected the subreviewing request for p issued by a.
PCM(a)	a is a PC member.
Requested-subrev(p,a,b)	a has requested b to be his subreviewer for p.
Reviewer(p,a)	p is assigned to PC member a for reviewing.
Submitted-review(p,a,b)	b's review of p has been submitted by a.
Subreviewer(p,a,b)	b has accepted the subreviewing request for p issued by a.

We now define the set of actions Actions and their execution permissions using the formula exec(act, u) for each action act $\in$ Actions. The execution permission statements define whether or not u of type Agent is allowed to execute such an action and in what state. In the following, we list a sub-set of ***EC*** actions and their permission statements which are used in our properties analysis in *X-Policy*:

1. When the review menu is enabled and the submitted paper is not deleted: **(a)** A PC chair can read all the paper reviews. **(b)** A PC member can read a review for a paper p if she is a reviewer of that paper and has submitted her review. **(c)** A PC member can read a review for a paper to which she is not assigned, when PC members are permitted to access the titles and reviews of submitted papers. She also must have no conflict of interest with that paper.

   Action ShowRev(p,a,b):- { return Submitted-review(p,a,b); }

$$\text{exec}(\mathsf{ShowRev}(\mathsf{p},\mathsf{a},\mathsf{b}),\mathsf{u}) \rightleftharpoons \left(\begin{array}{c} \begin{pmatrix} \mathsf{Chair}(\mathsf{u}) \wedge \mathsf{Chair\text{-}review\text{-}en}() \\ \wedge \exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Auth}(\mathsf{p},\mathsf{d}) \end{pmatrix} \\ \vee \left(\begin{array}{c} \mathsf{PCM}(\mathsf{u}) \wedge \mathsf{Reviewer}(\mathsf{p},\mathsf{u}) \\ \wedge \mathsf{PCM\text{-}review\text{-}menu\text{-}en}() \\ \wedge \exists\ \mathsf{c} : \mathsf{Agent}\ .\ \mathsf{Submitted\text{-}review}(\mathsf{p},\mathsf{u},\mathsf{c}) \\ \wedge \exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Auth}(\mathsf{p},\mathsf{d}) \end{array}\right) \\ \vee \left(\begin{array}{c} \mathsf{PCM}(\mathsf{u}) \wedge \neg\mathsf{Reviewer}(\mathsf{p},\mathsf{u}) \\ \wedge \mathsf{PCM\text{-}review\text{-}menu\text{-}en}() \\ \wedge \mathsf{View\text{-}sub\text{-}title\text{-}permitted}() \\ \wedge \mathsf{PCM\text{-}access\text{-}reviews\text{-}en}() \\ \wedge \neg\mathsf{Conf\text{-}of\text{-}interest}(\mathsf{p},\mathsf{u}) \\ \wedge \exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Auth}(\mathsf{p},\mathsf{d}) \end{array}\right) \end{array}\right)$$

2. When the review menu is enabled and the submitted paper is not deleted: **(a)** A PC chair can submit a review for any paper as himself. **(b)** A PC chair can submit a review for a paper as another PC member using "log in as another pc member" if the PC member is allowed to submit a review for that paper. **(c)** A PC member can review a paper if she is assigned to review that paper. **(d)** A PC member can review a paper to which she is not assigned when PC members are permitted to access the titles and reviews of submitted papers. She also must have no conflict of interest with that paper.

Action AddRev(p,a,b):-{ Submitted-review(p,a,b):=⊤;}

$$\text{exec}(\text{AddRev}(p,a,b),u)) \rightleftharpoons \left(\begin{array}{c} \left(\begin{array}{c}\text{Chair}(u) \wedge \text{Chair-review-en}()\\ \wedge a = u \wedge \exists d : \text{Agent} \;.\; \text{Auth}(p,d)\end{array}\right) \\ \vee \left(\begin{array}{c}(a = u \vee \text{Chair}(u)) \wedge \exists c : \text{Agent} \;.\; \text{Auth}(p,c) \\ \wedge \left(\begin{array}{c}\left(\begin{array}{c}\text{PCM}(a) \wedge \text{Reviewer}(p,a)\\ \wedge \text{PCM-review-menu-en}()\\ \wedge \text{PCM-review-editing-en}()\end{array}\right) \\ \vee \left(\begin{array}{c}\text{PCM}(a) \wedge \neg\text{Reviewer}(p,a)\\ \wedge \text{PCM-review-menu-en}()\\ \wedge \text{View-sub-title-permitted}()\\ \wedge \text{PCM-access-reviews-en}()\\ \wedge \neg\text{Conf-of-interest}(p,a)\end{array}\right)\end{array}\right)\end{array}\right)\end{array}\right)$$

3. When the review menu is enabled and the submitted paper is not deleted: **(a)** A PC chair can request another agent to subreview any paper. **(b)** A PC member can invite another agent to subreview a paper: (1) if she is the reviewer of the paper or (2) if the system is configured to give PC members access to the paper submission titles and reviews. The invited agent can decide whether to accept or reject the reviewing request as long as the paper has not been withdrawn. A PC member cannot cancel the subreviewing request but can accept or reject the request on behalf of the invited agent. Once the decision is made, only the PC member can change the decision.

Action RequestRev(p,a,b):- { Requested-subrev(p,a,b):= ⊤;}
Action AcceptRevRequest(p,a,b):-
{ Decided-subrev(p,a,b):=⊤; Subreviewer(p,a,b):=⊤;}
Action RejectRevRequest(p,a,b):-
{ Decided-subrev(p,a,b):=⊤; Subreviewer(p,a,b):=⊥;}

$$\text{exec}(\text{RequestRev}(p,a,b),u) \rightleftharpoons \left(\begin{array}{c} \left(\begin{array}{c}\text{Chair-review-en}() \wedge \text{Chair}(u)\\ \wedge \exists c : \text{Agent} \;.\; \text{Auth}(p,c)\end{array}\right) \\ \vee \left(\begin{array}{c}\text{PCM}(u) \wedge \text{Reviewer}(p,u)\\ \wedge \text{PCM-review-menu-en}()\\ \wedge \exists c : \text{Agent} \;.\; \text{Auth}(p,c)\end{array}\right) \\ \vee \left(\begin{array}{c}\text{PCM}(u) \wedge \neg\text{Reviewer}(p,u)\\ \wedge \text{PCM-review-menu-en}()\\ \wedge \text{View-sub-title-permitted}()\\ \wedge \text{PCM-access-reviews-en}()\\ \wedge \neg\text{Conf-of-interest}(p,u)\\ \wedge \exists c : \text{Agent} \;.\; \text{Auth}(p,c)\end{array}\right)\end{array}\right)$$

$$\text{exec}(\text{AcceptRevRequest}(p,a,b),u) \rightleftharpoons \left(\begin{array}{c}\text{Requested-subrev}(p,a,b)\\ \wedge \exists c : \text{Agent} \;.\; \text{Auth}(p,c)\\ \wedge \left(\neg\text{Decided-subrev}(p,a,u) \vee u = a\right)\end{array}\right)$$

$$\text{exec}(\text{RejectRevRequest}(p,a,b),u) \rightleftharpoons \left(\begin{array}{c}\text{Requested-subrev}(p,a,b)\\ \wedge \exists c : \text{Agent} \;.\; \text{Auth}(p,c)\\ \wedge \left(\neg\text{Decided-subrev}(p,a,u) \vee u = a\right)\end{array}\right)$$

4. Given that paper assignments are enabled, a PC chair can assign/de-assign a submitted paper to a PC member or a PC chair for reviewing, when she has no conflict of interest with that paper.

Action AddReviewerAssignment(p,a):-{ Reviewer(p,a) := $\top$; }

$$\text{exec}(\text{AddReviewerAssignment}(\mathsf{p},\mathsf{a}),\mathsf{u}) \rightleftharpoons \begin{pmatrix} \text{Chair}(\mathsf{u}) \wedge (\text{PCM}(\mathsf{a}) \vee \text{Chair}(\mathsf{a})) \\ \wedge \text{Review-assig-enabled}() \\ \wedge \neg \text{Conf-of-interest}(\mathsf{p},\mathsf{a}) \\ \wedge \exists \mathsf{c} : \text{Agent} \ . \ \text{Auth}(\mathsf{p},\mathsf{c}) \end{pmatrix}$$

3.2 Case Study: Analysis of *EC* Security Properties

In this Section, we will present three security properties in ***EC***. We have discovered these issues while using EasyChair. In each case, we show an attack strategy to achieve an undesirable state. Each strategy is an execution sequence of read and write actions. A strategy can be executed by more than one agent where agents collaborate to reach the goal. These attacks can be derived using ***EC*** and have succeeded on EasyChair as of 1st of Spetember 2009. In the following, we report the results of each attack and make some suggestions on how to fix the system. For our ***EC*** model, we create the following configuration:

1. The system has five agents: Alice, Bob, Eve, Carol and Marvin. The system has two submitted papers: p1 and p2.
2. Alice is the Chair of PC. Bob and Carol are PC members. Paper p1 is submitted by the author Marvin while p2 is submitted by the author Eve.

The detailed configuration and the attacks derivation of the model ***EC*** can be found at [1].

Property 1: A single subreviewer should not be able to determine the outcome of a paper reviewing process by writing two reviews of the same paper. We show that we can derive an attack against ***EC*** involving 4 agents: Alice, Bob, Carol, and Eve. We explain the attack scenario as a sequence of actions executed by these agents as follows:

1. Alice acts as chair. She executes the actions: AddReviewerAssignment(p1,Bob) to assign Bob to review the paper p1. She also executes AddReviewerAssignment(p1,Carol) to assign Carol to review the paper p1.
2. Bob and Carol both assign Eve as their sub-reviewer for paper p1 by executing the actions RequestRev(p1,Bob,Eve) and RequestRev(p1,Carol,Eve) respectively.
3. Eve accepts the two paper subreviewing requests and sends Bob and Carol two similar reviews using AcceptRevRequest(p1,Carol,Eve) and AcceptRevRequest(p1,Bob,Eve).
4. Bob and Carol receive Eve's reviews and submit them to the system using AddRev(p1,Bob,Eve) and AddRev(p1,Carol,Eve).

EasyChair fails this property and allows Eve to submit two reviews for the same paper. One possible fix for this attack is as follows. Every time an agent a invites another agent b to subreview a paper, EasyChair should check whether

agent b has been invited by another agent to subreview the same paper. We conjoin the condition $\neg\exists$ d : Agent . Requested-subrev(p, d, b) to the permission statement exec(RequestRev(p,a,b),u). This will prevent Carol from executing RequestRev(p1,Carol,Eve) as Requested-subrev(p1,Bob,Eve) is in the previous state.

Property 2: A paper author should not review her own paper. As before, we explain the attack scenario as a sequence of actions executed by the agents Alice, Bob and Eve:

1. Alice acts as Chair and assigns Bob, who is a PC member, to review the paper p2 submitted by Eve by executing the action AddReviewerAssignment(p2,Bob).
2. Bob executes the action RequestRev(p2,Bob,Eve) to assign Eve as his subreviewer as she is a good researcher in the field.
3. Eve accepts the request using AcceptRevRequest(p2,Bob,Eve).
4. Bob submits the review using AddRev(p2,Bob,Eve).

In this case, EasyChair fails the property and allows Eve to review her own paper. Note that the names of the authors and other reviewers are not known to the PC members. One possible fix for this attack is that every time an agent a invites another agent b to subreview a paper, EasyChair should check whether agent b is actually an author of that paper. We add the condition $\neg$Auth(p, a) to the permission statement exec(RequestRev(p,a,b),u). In this case Bob cannot execute RequestRev(p2,Bob,Eve).

Property 3: Users should be accountable for their actions. This property is violated in several ways, all of which involve the use of "log in as another pc member". For example, the system should not allow the chair to submit a review for a paper as another PC member without making it clear that it is actually the chair who has submitted the review and not the PC member. The following attack scenario involves Alice and Bob:

1. Alice is the chair. She executes AddReviewerAssignment(p1,Bob) to assign Bob to review the paper p1.
2. Bob submits his review using AddRev(p1,Bob,Bob).
3. Alice reads Bob's review of paper p1 by executing ShowRev(p1,Bob,Bob).
4. Alice submits a review for the paper p1 as if she is Carol who is a very famous and sought after academic by executing AddRev(p1,Carol,Carol).

EasyChair fails this property and allows the chair to read another reviewer's review for a paper and then submits a review for that paper as another PC member without being detected by the other PC members or the other chairs. This attack is possible because the system does not register the name of the user who updated the review. It will appear to others as if Carol has submitted the review herself. One possible fix for this attack is for AddRev() to have an additional parameter. Alice would then need to execute the action AddRev(p,a,b,c) where agent a is the chair acting on behalf of b who is the PCmember submitting the review written by agent c. The predicate Submitted-review() also has to be changed accordingly.

4 Conclusion and Future Work

We present a modelling language, *X-Policy*, to model the dynamic execution permissions of large web-based collaborative systems. We demonstrate the applicability of *X-Policy* to real-life web-based collaborative systems like EasyChair. Using *X-Policy*, we reason about the security properties of three security properties for EasyChair and described the possible attacks on these properties as well as ways the system could be changed to prevent these attacks. We have informed the developer of EasyChair of our findings. The full ***EC*** model is available at [11]. It contains 49 actions and permission statements. This is relatively concise given the size and complexity of EasyChair. The way the system functionality is split into actions is decided by our understanding of how the system is actually designed. Due to space restrictions, we detail the syntax and semantics of *X-Policy* and the traces for the discussed attacks in the long version of this paper [1]. In future work, we would like to model and analyse more systems, develop, and implement an algorithm to automate the analysis of these systems using model checking techniques.

References

1. Qunoo, H., Ryan, M.: Modelling dynamic access control policies for web-based collaborative systems - long version. Technical report, School of Computer Science, University of Birmingham, Available at the authors' webpage (April 2010)
2. Becker, M., Fournet, C., Gordon, A.: Design and semantics of a decentralized authorization language. In: 20th IEEE Computer Security Foundations Symposium, CSF 2007, pp. 3–15 (2007)
3. Becker, M.Y.: Specification and analysis of dynamic authorisation policies. IEEE Computer Security Foundations Symposium, 203–217 (2009)
4. DeTreville, J.: Binder, a logic-based security language. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy (2002)
5. Gurevich, Y., Neeman, I.: DKAL: Distributed-knowledge authorization language. In: CSF 2008: Proceedings of the 2008, 21st IEEE Computer Security Foundations Symposium, Washington, DC, USA, pp. 149–162. IEEE Computer Society, Los Alamitos (2008)
6. Gurevich, Y., Neeman, I.: DKAL 2: A simplified and improved authorization language. Technical report, Microsoft Research - Cambridge (2009)
7. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust management framework. In: Proc. IEEE Symposium on Security and Privacy, Oakland (May 2002)
8. McDermott, D., Doyle, J.: Nonmonotonic logic 1. Artificial Intelligence 13, 41–72 (1980)
9. Becker, M.Y., Sewell, P.: Cassandra: distributed access control policies with tunable expressiveness. In: 5th IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY (2004)
10. Zhang, N., Ryan, M., Guelev, D.P.: Synthesising verified access control systems in XACML. In: 2004 ACM Workshop on Formal Methods in Security Engineering, Washington DC, USA, October 2004, pp. 56–65. ACM Press, New York (2004)
11. Qunoo, H., Ryan, M.: EC model in X-policy (December 2009), `http://www.cs.bham.ac.uk/~hxq/X-policy/`