Validating access control policies with Alloy

Waël Hassan[2], Luigi Logrippo[1,2], Mahdi Mankai[1]

[1]*Département d'informatique et ingénierie, Université du Québec en Outaouais*

[2] *School of Information Technology and Engineering, University of Ottawa*

ABSTRACT. We present two projects that are being developed in our research group. The first is on inconsistency detection in XACML policies. The second one develops a novel access control paradigm, process-based access control, and has inconsistency detection as part of its goal. The model analyzer Alloy is being used in both projects.

1. Background and motivation

The old way of controlling computing systems was by hard-coding all control in monolithic programs. Then the concept of application program came up pre-packaged with a number of features. Everybody now knows how to activate or deactivate features by checkbox selection. In telephony, the Intelligent Network architecture allows users to choose and possibly build their telephony features out of prefabricated components. However when the IN concept was introduced some sort of critical threshold was crossed, because the phenomenon of *feature interaction* was discovered. We had a highly distributed system that was constructed from independent components, each capable of complex independent behavior. The features composed in this way can produce behavior that can contradict the *intentions* of coexisting features, or basic system requirements.

In many cases, more flexibility in customization is desired than what can be obtained by checkbox selection, however the full power of a programming language is not required. We are then in the area of *policy languages*. Firewalls and routers can be programmed by using policy languages. Internet telephony can be programmed by languages such as CPL, the Call Processing Language[1]. The policy languages that apply in these areas are specialized languages, that specify what the user expects in various cases. The power of these languages is usually rather constrained, both in order to simplify programming and in order to prevent certain behaviors. However *interactions* are still possible, in fact they acquire a whole new dimension [3][4].

Many types of interactions can be explained in terms of inconsistencies in sets of rules. Typical examples are the following: one rule leads to blocking a call, another that is simultaneously enabled leads to forwarding it; one rule allows a certain user to access a certain document, another rule forbids such access; one rule in the firewall lets in a certain packet, another rule blocks it. Therefore, many feature interactions can be detected by looking for inconsistencies in sets of rules.

In some systems, many policy interactions are taken care of by priority mechanisms that are built in the policy execution system. For example, in telephony features, the first rule that matches a condition may be applied and other following rules that match the same condition are then ignored. However, this does not solve the problem for two reasons. The first reason is the fact that the coexistence of two conflicting rules in a system could be an error and therefore should be brought to the attention of the user. The second reason is the fact that if the inconsistencies are due to combination of rules that reside in different locations, then it may be very difficult to ensure that only one of these rules is applied, and even if this is possible, it may be contrary to the intention of one of the parties involved.

In this paper, we propose an approach to policy interaction detection that is based on the use of a logical model checker. The specific tool we propose is Alloy [5].

2. Access control policies

Access control policies are policies that determine who, and under what conditions, can access certain systems resources, especially data resources. Firewalls are network level mechanisms for enforcing access control, and some of the concepts discussed below apply to the analysis of firewalls. However we are mainly interested in higher level languages, where access control rules can be specified in terms that are directly related to the roles and purposes of users. RBAC (role-based access control) [7][9] is a well-known access control paradigm, and the paradigms we will discuss below intend to be more general than RBAC, in the sense that the latter can be expressed in them.

The practical problem here is that access control policies are often complex, difficult to understand, and developed incrementally. They will be administered by lawyers, clerks, and other people who won't be necessarily computer-literate, so they must be made manageable for these people. Hence, methods and tools are needed to identify and report semantic errors.

We are pursuing two research areas, where the second one is more ambitious and longer-term

1) Analysis of access control policies expressed in the OASIS standard XACML language
2) Study of new access control methods, namely 'process-based' ones.

2.1 Inconsistencies of policies expressed in the XACML language

XACML, the eXtensible Access Control Markup Language, intends to be a language for the expression of a wide range of access control policies. It is defined as a set of XML schemas .[12]

A simple example of rules expressible in XACML is the following (where X is the URL of a XML file):

- Rule1: A person may read any medical record in the X namespace for which he or she is the designated patient
- Rule2: A person may read any medical record in the X namespace for which he or she is the designated parent or guardian, and for which the patient is under 16 years of age
- Rule3: An Administrator shall not be permitted to read or write medical elements of a patient record in the X namespace.

It is easy to see that when the subject is both administrator and patient, both permit and deny responses are allowed by the rules for X. It should be noted that XACML has methods to take care of such conflicts. These have the self-explaining names of "deny-overrides", "permit-overrides", "first applicable", and "only-one applicable". However, as mentioned, conflicts should be brought to the attention of the user because the user may be unaware of them, leading to unintended access decisions.

Although a very simple example had to be selected for this brief paper, clearly the use of automatic methods will enable the discovery of much more involved inconsistencies, as are possible in sets of policies containing hundreds of rules.

We have defined the semantics of XACML in terms of the Alloy notation, and shown the feasibility of the approach. We are completing the work by implementing an automatic translator from XACML to the Alloy notation. As a by-product of this research, we have also developed a translator from XACML notation into a more user-friendly natural language notation that can be used in a user interface for generating XACML scripts.

2.2 Process-based access control and inconsistency detection

This second project goes well beyond validation and intends to first develop a new access control paradigm. The latter is based on models of enterprise processes.

Using high level policies, enterprise control systems will soon be self managing [2][6]. High level policies will be decoupled from enterprise structural elements, such as roles, and applied to processes. While policies deliver the purpose required by a designer, the governance model, thus created, is prone to ambiguity, complexity, and interactions. In this project we will show how to help enforce policies, reduce complexity, and detect interactions in enterprise privacy management systems.

In the past, several methods have been used to capture enterprise requirements. ebXML, XACML [12], and EPAL [8] are good examples of policy languages that implement enterprise policies. RBAC helped reduce complexity by decoupling users from permissions and grouping them into roles [7][9]. RBAC has succeeded in representing hierarchical organizations that are divided by function (example: finance, accounting, manufacturing); however, it is less successful at capturing organizations that are based on service, product, or activity. (Ex: Loan Application), which may include several people from different departments. All these techniques are restricted to functional enterprises and not process (Activity) based organizations. None of these techniques or languages address the issue of policy interactions.

We propose the use of process-based access-control methods in the specification of privacy systems. A process has a target usually, and is broken into discrete steps. Each of the steps can be assigned to a role with an access right. We show that a process-based model, including roles, actions, and permissions, can correctly represent policy intent and hence enable enforceability of privacy policies. We show how such a model can be described in UML and then we show how policy interactions in the model can be detected by using an analyzer such as Alloy.

Our philosophy: By grouping business activities in processes, we can specify how roles and resources relate in a business context. After compiling the set of processes, we attach one or more policies to each process. A policy specifies access rights of roles participating in processes. After analysis, a semi-formal UML model is created. The model is translated into Alloy and interactions and inconsistencies are detected.

In order to address requirements of the evolving enterprise, one that is focused on business activities rather than structure, the project is broken down into three stages: Our first goal is to find a meta-model that captures enterprise specifications. In stage two, we create a model following the methodology, this should reduce complexity. In stage three, we translate the model into a formal language to verify it, and to identify conflicts that can then be removed.

This technique has already been demonstrated on some small examples, and we are now exploring its applicability in a more general setting. Issues to be addressed are delegation and separation of concerns. Delegation occurs when a user assigns his role to another person, separation of concerns occurs when a user is denied access to two resources at the same time.

The concept of separation of concerns is important in the banking industry. One of its applications is that a principal is not allowed to combine access to specific functionalities. Should these capabilities be combined, a violation of a policy may occur. For example, an employee cannot have access to customer address *and* credit card information. Such a rule is general and applies to the Credit Card processing process. However, one of the tasks of accepting a new card includes the mailing of the credit card to the consumer. Such a process violates the conditions set by the process. In this scenario, a local policy violates a more general policy. The Verify Address action, which is a part of the Mail Card process, violates the condition set forth by the Credit Card process.

The concept of *process ontology* has an important role in our approach. It defines the organization as a tree of processes. Ontologies, normally described as the structure of the domain, define concepts and relationships. Each process is a group of transactions; however granular, transactions (actions) follow a certain workflow and have a specific purpose. Process ontology is an alternative to traditional Role Based structure. In our enterprise mapping to processes, roles participate in processes and policies govern processes.

The ontology provides us the structure that defines the organization. It offers us the ability to localize policies by presenting scope. Since a policy is attached to a process, its scope is defined by the process and its sub processes. A second advantage of an ontology is that it can decouple organizational structure from workflow structure. The gap between both is important because when we allow Alex to open file F as a part of his organizational role structure, Alex will have access to file F at all times, and can use it regardless of her job function. However, if she was assigned file F as a part of process Loan Application, then the permission is only available during the sequence of operations leading to a Loan Application, therefore flow has been followed and context is achieved.

3. The Alloy system and its use for the detection of inconsistencies

As mentioned, the logic engine being used in our projects is MIT's Alloy[5]. Alloy is a 'model' or 'constraint' analyser. It allows to express systems as sets of logical constraints in a logical language based on standard first-order logic. As well, when creating a model the size the system must be specified (for example, how many users, how many subjects). Alloy then compiles a large Boolean matrix for the constraints, and it can be asked to check if a model exists, or if there are counterexamples.

The advantage of Alloy with respect to a theorem prover is that the latter usually requires human interaction and direction in order to specify theorems to be proved and to complete the proofs. Alloy instead is fully automatic and in all the examples we have tried has always terminated with a result. It is true that the problem of satisfaction of Boolean tables is NP-complete. However, on the one hand, Alloy uses many optimizations and heuristics; on the other hand, the complexity of NP-complete problems depends on the number of variables, and in this area there are usually few.

One of the main questions in policy systems is policy compliance. This is important in the privacy domain, because an organization needs to prove that its operations are compliant with a set of policies. A prerequisite for compliance to a set of policies, is having a system that is void of interactions. Alloy can be used to construct the model, and verify its logical consistency. In order to prove compliance to the set of policies, they need to be bundled in a module, the module is asserted, and Alloy will produce a result specifying whether the module is compliant to the system described.

The first step in using Alloy is to define the data containers. This is equivalent to defining classes in an object oriented model. The second step is to create relationships. Relationships can be used to specify membership. For example one can specify that each set of employees has exactly one manager, and therefore no one can be left without a manager. By using relationships it is possible to represent the ontology of an organization and access rights. For our separation of concerns question, for example, the following statement:

```
(no employee->processA &  employee->ProcessB)
```

suffices to stipulate that no one employee who has access to Processes A can have B concurrently.

The third step is to group the privacy statement policies that need be verified in a module. The user can then issue the statement

```
(assert Module_My_Policies, manager 3, employee 4, and printer 6).
```

Such a statement will populate a world with the exact number of entities and a random set of relationships. Alloy can come back with the answer that the module satisfies the system or it may come back with a counter-example describing a case in which these policies fail. It is important to note however that Alloy may not be able to return a counter-example, because it is constrained to a specific number of instances of each entity.

The examples discussed in this paper, and of course a number of others, were successfully run under Alloy.

4. Conclusion

Access-control governance of enterprise using policies requires a methodology for specification and verification. Our process control model can capture privacy requirements and deliver user intent by setting context (the ontology). Alloy can verify system integrity and can detect interactions in a constrained space. Future work requires the creation of a policy language set that can be translated into Alloy, in addition to formal refinement methodology using our enterprise meta-model.

We are also planning to assess other tools beside Alloy.

References

[1] J. Lennox, X. Wu , H. Schulzrinne, Call Processing Language (CPL): A language for User Control of Internet Telephony Services, IETF RFC3880, October 2004.

[2] J. Kephart, D. Chess, The vision of Autonomous Computing, Computer Journal, IEEE computer society, 41-50, January, 2003.

[3] S. Reiff-Marganiec and K. Turner. A Policy Architecture for Enhancing and Controlling Features, Proc. Feature Interactions in Telecommunication Networks VII, 239-246, IOS Press, Amsterdam, June 2003.

[4] P.Dini, A.Clemm, T.Gray, F.J. Lin, L. Logrippo, S. Reiff-Marganiec. Policy-enabled Mechanisms for Feature Interactions: Reality, Expectations, Challenges. Computer Networks, 45 (5), 2004, 585 - 603.

[5] D.Jackson. Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM) 11, 2 (2002) 256 - 290

[6] V. Thurner, A formally founded description technique for business processes, Technical Report, Technical University of Munich, Germany, 1997.

[7] D. Ferraiolo, D. Kuhn, R. Chandramouli, Role-Based Access Control, Artech House Publishers, April 2003.

[8] M. Schunter Ed., Enterprise Privacy Authorization Language (EPAL 1.1), http://www.zurich.ibm.com/security/enterprise-privacy/epal, Accessed Jan. 2004.

[9] Q. He, Privacy Enforcement with an Extended Role-Based Access Control Model, Report North Carolina State University, April 2003.

[10] G. Karjoth and M. Schunter, A Privacy Policy Model for Enterprises ,5th IEEE Computer Security Foundations Workshop, pp.271-281, IEEE, 2002.

[11] N. Zhang, M. Ryan, D. P. Guelev, Synthesising Verified Access Control Systems in XACML, FMSE’04, October 29, 2004, Washington, DC, USA.

[12] XACML specification, OASIS, http://www.oasis-open.org/specs/index.php#xacmlv1.0, Accessed 2004.