CESSA

Compositional Evolution of Secure Services using Aspects

ANR project no. 09-SEGI-002-01

Certification Infrastructure and Mechanisms (Intermediate Version)

Abstract.

In this document we describe our plan for applying and certifying policies expressed in the CSPL language, which is introduced in Deliverable D2.2. We point out several open issues, such as implementation of reference monitors, mechanisms to put in relation abstract and concrete policies, and optimization issues. For some of these issues, ideas can be borrowed from approaches such as the XACML language and Law-Governed Interaction (LGI); other issues require new original research. We discuss our plans to tackle these problems.

Deliverable No.	I2.3
Task No.	2
Type	Deliverable
Dissemination	Public
Status	Intermediate Version
Version	1.0
Date	29 Sep. 2011
Authors	Matteo Dell'Amico, Yves Roudier (EURECOM).

Contents

Chapter 1

Introduction

CSPL [2] is a language designed within the CESSA project framework, with the goal of enabling the specification of security policies in complex distributed systems in a concise, readable and extensible way. According with the aspect-oriented paradigm, the goal of CSPL is to provide a way to express a security policy for a complex distributed system in a centralized fashion, making it easier to maintain and verify the implementation of security in such situations.

A key characteristic of CSPL is that it is designed in order to handle policies spanning through several entities, which can in turn be further split into sub-entities. These entities and sub-entities represent several different actors or groups thereof, which can be entitled to different levels of trust, and they form a hierarchy whose nodes are called *policy domains*. The hierarchical structure is essential to express policies that cover both very large sets of locations and others that apply to much smaller domains. CSPL also includes the possibility of labeling data allowing for information-centric rules and grouping policy domains according to characteristics that may be orthogonal to the hierarchy of policy domains (e.g., rules that apply to all mobile devices within a large organization).

A CSPL policy can be regarded as a document agreed upon by all entities participating in it, but the policy enforcement should not require mutual trust between all the entities in the system. This is realized with a model that requires a reference monitor per policy domain, constraining all communication towards different policy domains through reference monitors, and constraining each reference monitor to exchange messages with neighbors in the hierarchy.

CSPL allows the definition of *abstract* policies, which amount to defining security properties that have to be respected, as well as defining *concrete* rules, which detail the way in which reference monitors should act in order to ensure that the required security properties are implemented.

In this document, we will focus mainly on the strategies that we intend to pursue in order to implement CSPL, to evaluate and certify the relationships between the abstract policies that CSPL should implement and to detect inconsistencies between conflicting rules and to detect mismatches between specification and implementation. This document extends and complements the CESSA deliverable D2.2 [2], which should be used as a reference for the syntax and semantics of the language.

In this document, we will describe the relations between abstract and concrete rules, and our plans to handle and test them (Chapter 2), and ideas for implementation of reference monitors

and considerations with respect to efficiency (Chapter 3).

Chapter 2

Abstract and Concrete Rules

CSPL allows for expressing *abstract* and *concrete* rules. Abstract rules express at a high level of abstraction the security policies that should be enforced; concrete rules instead determine univocally the actions that reference monitors should take. Consider, for example, the following abstract rule requiring that messages tagged as `classified` should only be readable in domains A and B:

```
Root {m:classified -> m is confidential(DomainA, DomainB)}
```

The `confidential` property can be implemented in several ways: for example,

1. a secure channel of communication can be set up between DomainA and DomainB,
2. messages tagged as `classified` are encrypted with appropriate keys when sent between the two domains,
3. as an extreme measure, when no other solution can be taken, `classified` messages can be filtered by reference monitors when they are leaving DomainA or DomainB.

To relate abstract and concrete policies, various solutions are open:

1. Given an abstract specification and a concrete one, verify that the concrete specification matches the properties required in the abstract one
   (a) via inference rules (*e.g.*, using logic programming languages such as Datalog or Prolog)
   (b) via testing or model checking, to ensure that cases violating abstract rules do not appear
2. Given an abstract specification, *generate* a concrete one that satisfies the abstract requirements

(a) A simpler solution is having deterministic rules that expand into a single concrete implementation (*e.g.*, in the above example this translates to concrete rules taking place at the reference monitors of domains A and B that perform encryption to enforce confidentiality of tagged messages). This can be implemented, again, by performing inference as done in Prolog or Datalog.

(b) A more complex task would be solving an optimization problem (related to non-functional requirements: efficiency, simplicity, ease of deployment...) over all possible solutions, trying to find an optimal solution.

3. A third possibility which sits between the extremes of having automated derivation of concrete rules and relying on a human administrator to decide all of them and use automated tools only to verify accordance between abstract properties and concrete implementation is a semi-automated procedure where the administrator uses an automated tool that proposes several solution to comply with the abstract required properties. This could limit the tedious part of requiring the administrator to deal with minute choices, but still give the benefits of human supervision in high-level choices.

As a first approach, our plan is to go with the simpler alternatives (*i.e.*, 1a and 2a), while still considering the more complex possibilities for further work.

2.1 Testing

Testing can be used to certify two goals: first, to certify that the implementation conforms to the concrete specification; second, to certify that it also conforms also to the abstract specification. We will consider the effectiveness of testing in the context of these two cases; moreover, in several cases generating comprehensive tests will be impossible; we plan to consider strategies and heuristics to generate the most significant test cases.

Chapter 3

Implementation Issues

In this Chapter, we outline various issues related to the implementation of reference monitors in CSPL. The closest work to CSPL is Law-Governed Interaction (LGI) [4, 5] which, however, do not take into account the abstract policies described in Chapter 2. In order to exploit what already done for LGI, here we will outline similarities and differences, highlighting what can be reused in existing LGI approaches and what has to be changed in order to implement CSPL.

3.1 Policy Domain Hierarchy

A key property of CSPL is that communication is assumed to travel only along the hierarchy of domains; this property allows creating clear "checkpoints" for the flow of information. We note that, if a policy domain is considered not trusted enough to avoid information leaking through communication channels that are not supervised by reference monitors, the correct approach is to filter information traveling through those domains, in order to avoid that such a leak jeopardizes the security requirements. This kind of control is doable within the framework of CSPL.

A hierarchical approach with strong analogies to CSPL has been proposed by Ao *et al.* [1]. Similarly to the CSPL domain hierarchy, that work proposes a hierarchy of "sets of agents" which adhere to policies ("laws") that become more and more specific as the set of agents to which it applies becomes smaller. More generic laws apply to all the agents in the system (akin to what a Constitution does in a legal framework), while more specific ones apply to smaller fields.

In [1], the communication that each agent perform is filtered by a *controller* whose job is similar in spirit to CSPL's reference monitors. A fundamental difference is however that, in [1], *controllers are trusted*. This entails that there is no point in forwarding messages along different controllers in the whole domain hierarchy: using an inheritance mechanisms, local controllers implement *all* the rules that apply to a piece of communication; conversely, in CSPL, reference monitors only have the task of enforcing rules that concern their domain, deferring rules pertaining to other domains to their respective monitors. This increase in messaging is due to our lack of an assumption of complete trust in all reference monitors.

3.2 Implementation of Reference Monitors

In [2], we have defined reference monitors essentially as application proxies that intercept all traffic directed to services, performing filtering, rerouting and message rewriting tasks that implement the required concrete policies. Similar approaches are taken in XACML [3] and in LGI [5]. XACML defines an architecture that breaks down the implementation of a reference monitor in sub-components called policy enforcement, decision, retrieval, information, and administration points; LGI collapses these into a single component that considers controllers as entities that can represent the current system state as observed locally, and uses Prolog in order to derive the actions that should be taken. We plan to further investigate these approaches in order to decide how much they can be re-used in order to implement CSPL controllers.

In addition, CSPL has the ambition of enforcing policies even within policy-domains that do not have explicit message-passing, such as for example different processes residing on the same machine. Our idea is to consider means for communication between them (e.g., shared memory areas) as another way to do messaging between messages. In these cases, having reference monitors intercept and parse all data exchanged between messages would probably be unsustainable, and we consider using AOP techniques to "weave" in code the same policy enforcement that would be performed by a reference monitor, if it happened to be there intercepting the communication.

We plan to focus our efforts on monitoring web services. We see two possibilities, namely considering SOAP or REST interfaces. We will take this choice aiming for simplicity of implementation and the requirements of the partners.

3.3 Efficiency Issues

CSPL reference monitors can be a bottleneck if the policies to be enforced are too expensive computationally or if they aggregate too much traffic towards a single component. We will consider this kind of efficiency issues, as well as possible solutions such as:

- Considering several possible concrete realizations of abstract policies, and choosing the ones which appear as less computationally expensive;
- Considering, where it makes sense, the possibility of moving towards less strict, but more efficient, ways of enforcing behavior. For example, to detect anomalies, it may be sufficient to sample some of the traffic to analyze rather than working on all of it;
- Considering decentralized implementations of reference monitors: if they have no or small shared state, it may be possible to have several equivalent reference monitors working in parallel; the choice of the concrete security policies to apply will then need to be chosen with this kind of scalability in mind;
- Verifying which states are unreachable, avoiding that reference monitors perform redundant checks that will always be negative;

- Moving from an implementation of reference monitors as application proxies to a style where AOP techniques can be used in order to weave new behavior in the unsupervised services, in order to obtain a result which equivalent to the pair of service and reference monitor, while being more computationally efficient.

- "Shortcutting" communication between reference monitors: when the reference monitor of a particular domain is known not to perform any checks on a subset of messages, it is possible to think about letting them be transfered directly to the first reference monitor that will need to handle data, limiting the number of hops that a message has to traverse and lowering the load on monitors at higher levels of the domain hierarchy. Firewall rules can be implemented in order to make sure that only allowed messages escape the message routing policy which is carried out along the policy domain hierarchy. Performing this kind of optimization is particularly interesting at the "root" of the domain hierarchy, where implementing a reference monitor that should handle all traffic and be trusted by all actors can be very cumbersome. We will consider analyzing which concrete realizations of abstract policies can avoid having to deal with such a "root" reference monitors.

Chapter 4

Conclusion

In this document we have highlighted a set of issues that are relevant to the implementation of the CSPL language. Some of the key issues can be solved by borrowing ideas from related work (LGI and XACML in the forefront), while others can open the road to new research – in particular, the derivation and/or verification step between concrete and abstract rules as defined in Chapter 2, and optimization rules described in Section 3.3.

Bibliography

[1] X. Ao, N. Minsky, and T.D. Nguyen. A hierarchical policy specification language, and enforcement mechanism, for governing digital enterprises. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 38–49. IEEE, 2002.

[2] Matteo Dell'Amico, Sabir Idrees, Yves Roudier, Anderson Santana de Olivera, Gabriel Serme, and Gaetan Harel. Language definition for security specifications. Deliverable D1.1, The CESSA project, July 2011.

[3] T. Moses (ed.). extensible access control markup language (xacml) version 2.0. Technical report, OASIS Standard, 2005.

[4] N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, pages 183–195, 1991.

[5] N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(3):273–305, 2000.