

Policy evaluation contracts

Tom Goovaerts Bart De Win Wouter Joosen

Report CW 543, May 2009



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Policy evaluation contracts

Tom Goovaerts Bart De Win Wouter Joosen

Report CW 543, May 2009

Department of Computer Science, K.U.Leuven

Abstract

The management of expressive authorization policies in open and dynamic systems is a significant challenge. In this report, policy evaluation contracts are defined to manage the composition of authorization policy components. Policy evaluation contracts specify security policy-related requirements and capabilities in terms of a generic policy domain model. Furthermore, an algorithm is described for generating a dependency graph between authorization policy components, based on their contracts. Using these proposed concepts, management components of authorization infrastructures can maintain a correct configuration of the infrastructure when changes occur in policies or in the environment.

Policy Evaluation Contracts

Tom Goovaerts Bart De Win Wouter Joosen

IBBT-Distrinet, Katholieke Universiteit Leuven, 3001 Leuven, Belgium

{tomg,bartd,wouter}@cs.kuleuven.be

Abstract

The management of expressive authorization policies in open and dynamic systems is a significant challenge. In this report, policy evaluation contracts are defined to manage the composition of authorization policy components. Policy evaluation contracts specify security policy-related requirements and capabilities in terms of a generic policy domain model. Furthermore, an algorithm is described for generating a dependency graph between authorization policy components, based on their contracts. Using these proposed concepts, management components of authorization infrastructures can maintain a correct configuration of the infrastructure when changes occur in policies or in the environment.

1 Introduction

In this report, we aim to give assurance about the correctness of the configuration of authorization architectures that enforce attribute-based policies. Our approach leverages on the well-known software engineering technique of software contracts [1] to explicitly specify the requirements and capabilities of collaborating authorization policy components.

It is common practice in modern authorization architectures to decompose the policy enforcement process (or parts thereof) into collaborating and replaceable authorization policy components. The first decomposition is the one between policy enforcement points (PEPs) and policy decision points (PDPs) (enforcement functions and decision functions in [3]). PEPs intercept access requests in the applications and make decision requests to the PDP. In response, the PDP evaluates the policy, makes an access decision and returns it to the PEP which enforces it. Additionally, PDPs can

be composed with policy information points (PIPs) (attribute functions in [2]), that obtain additional attributes from the applications or from other sources.

A policy evaluation contract, or simply a contract, is associated with one particular authorization policy component (PEP, PDP or PIP) and lists which kinds of requests it can process or requires to be processed. Contracts of PDPs can depend on the contents of the policy. Based on the contracts that are associated with authorization policy components, a dependency graph can be constructed that contains all valid configurations of the set of components.

Section 2 defines policy evaluation contracts. Section 3 defines operations that are performed on contracts and in Section 4, the translation of a set of contracts into a dependency graph is discussed.

2 Definitions

Universe of contract items (set U) The universe is the set of all possible contract items. The universe is the policy domain model that holds concepts that can be used in the policies. Most importantly, it holds the types of actions and attributes that can be protected and obtained.

Attributes and actions (sets U_{att} and $U_{azn} \subset U$) The subsets of subject and resource attributes and actions.

Type (set T) The set of possible contract types.
 $T = \{att, azn\}$

Type Domain (function d) The function that maps each type into a set of allowed values, called the domain for the type.
 $d : T \rightarrow \mathcal{P}(U) : \{(att, U_{att}), (azn, U_{azn})\}$

(Atomic) Contract (set C) A contract specifies a number of required and provided dependencies (a contract cannot not specify a dependency upon itself). The set of contracts C is defined as follows:
 $C \subseteq \{(t, R, P) | (t, R, P) \in T \times \mathcal{P}(U) \times \mathcal{P}(U), R, P \subseteq d(t), R \cap P = \emptyset\}$
 We denote a contract (t, R, P) as $c_t(R, P)$.
 For instance, $c_{azn}(\{document.read\}, \emptyset)$

Type/Required/Provided functions (R/P) These functions map a contract onto its type, required set and provided set components:

$$\begin{aligned}
type : C \times T &: c_t(X, Y) \mapsto type(c_t(X, Y)) = t \\
R : C \times \mathcal{P}(U) &: c_t(X, Y) \mapsto c_t(X, Y).R = X \\
P : C \times \mathcal{P}(U) &: c_t(X, Y) \mapsto c_t(X, Y).P = Y
\end{aligned}$$

Attribute/Authorization contracts (C_{azn} and C_{att}) These sets partition C into the sets of contracts with the same type and are defined as follows:

$$\begin{aligned}
C_{azn} &= \{x \in C \mid type(x) = azn\} \\
C_{att} &= \{x \in C \mid type(x) = att\}
\end{aligned}$$

Composite Contract (set C_c) A composite contract is a tuple of two atomic contracts; one authorization contract and one attribute contract. The set of composite contracts C_c is defined as follows:

$$\begin{aligned}
C_c &\subseteq C_{azn} \times C_{att} \\
\text{Elements in } C_c &\text{ are denoted by } \{x, y\}_c.
\end{aligned}$$

Required/Provided functions for composite contracts (R_c/P_c) These functions map a composite contract onto its required set and provided sets:

$$\begin{aligned}
R_c : C \times \mathcal{P}(U) &: \{c_1, c_2\}_c \mapsto c_1.R \cup c_2.R \\
P_c : C \times \mathcal{P}(U) &: \{c_1, c_2\}_c \mapsto c_1.P \cup c_2.P
\end{aligned}$$

Instead of R_c and P_c , we simply use R and P in the rest of this report. It is clear from the context which function is intended.

PEPs have an atomic authorization contract with an empty provided part, because they only require the processing of authorization requests and because they provide nothing. PDPs have a composite contract that consists of (1) an authorization contract with an empty required part and (2) an attribute contract with an empty provided part. PIPs have an atomic attribute contract with an empty required part because they only provide attributes.

3 Contract Matching and Satisfaction

Atomic Contract Matching (\oplus_a) is defined as a function that maps two atomic contracts of the same type into a new contact in which the required and provided parts are cross-matched with each other:

$$\oplus_a : C \times C \rightarrow C : (c_t(R_1, P_1), c_t(R_2, P_2)) \mapsto c_t(R_3, P_3), \text{ with } R_3 =$$

$R_1 \cup R_2 \setminus ((R_1 \cap P_2) \cup (R_2 \cap P_1))$ and with $P_3 = P_1 \cup P_2$

Composite Contract Matching (\oplus_c) $\oplus_c : C_c \times C_c \rightarrow C_c : (\{c_1, c_2\}_c, \{c_3, c_4\}_c) \mapsto \{c_1 \oplus_a c_3, c_2 \oplus_a c_4\}_c$

Contract Matching (\oplus) $\oplus : (C \cup C_c) \times (C \cup C_c) \rightarrow (C \cup C_c) : (c_1, c_2) \mapsto c_1 \oplus c_2$, which is defined as follows:

- If $c_1, c_2 \in C$ then
 - If $type(c_1) = type(c_2)$ then $c_1 \oplus c_2 = c_1 \oplus_a c_2$.
 - Else, if $type(c_1) = azn$ then $c_1 \oplus c_2 = \{c_1, c_2\}_c$.
 - Else, if $type(c_1) = att$ then $c_1 \oplus c_2 = \{c_2, c_1\}_c$.
- If $c_1, c_2 \in C_c$ then $c_1 \oplus c_2 = c_1 \oplus_c c_2$.
- If $c_1 \in C$ and $c_2 \in C_c$ then
 - If $type(c_1) = azn$, $c_1 \oplus c_2 = \{c_1, c_{att}(\emptyset, \emptyset)\}_c \oplus_c c_2$
 - If $type(c_1) = att$, $c_1 \oplus c_2 = \{c_{azn}(\emptyset, \emptyset), c_1\}_c \oplus_c c_2$
- If $c_1 \in C_c$ and $c_2 \in C$ then $c_1 \oplus c_2 = c_2 \oplus c_1$.

Thus, if the contracts are both atomic or composite, the atomic or composite matching is taken, and else the result is a composite contract in which the compatible contacts are matched atomically.

Contract satisfaction (function sat^*) A contract is satisfied if it does not have any required items:

$$sat^* : C \rightarrow \{0, 1\} : c \mapsto 1 \text{ if } c.R = \emptyset, \text{ and } c \mapsto 0 \text{ if } c.R \neq \emptyset$$

Contract set satisfaction (function sat) A set of contracts is satisfied if the matching of all its elements is satisfied:

$$sat : \mathcal{P}(C \cup C_c) \rightarrow \{0, 1\} : c_1, c_2, \dots, c_n \mapsto sat^*(c_1 \oplus c_2 \oplus \dots \oplus c_n)$$

4 Dependency Graphs

Via their required and provided sets, contracts express dependencies. Given a certain set of contracts, we can build a directed dependency graph of those contracts, in which the nodes represent the contracts and the edges represent the elements via which the contracts depend on each other.

4.1 Definition

The dependency graph $DG(S)$ for a set of composite contracts $S \subseteq C_c$ is defined as a tuple (V, E) in which

$$V = S$$

$E = \{(u, v) \in V \times V \mid \exists x \in U : x \in u.R \text{ and } x \in v.P\}$ sometimes we use the notation $(u, v)_x$ to denote the precise dependency of an edge.

Thus, there is an edge between two contracts if there is an element in the required part of the first contract that occurs in the provided part of the second contract. The universe of dependency graphs is the set DG .

4.2 Construction Algorithm

The algorithm for constructing a dependency graph is straightforward:

Given: a set of composite contracts S

- For each contract $c_i = \{R_i, P_i\}_c$
 1. Add c_i to V
 2. For each required item $r_i \in R_i$
 - For each other contract $c_j = \{R_j, P_j\}_c$ in $S \setminus \{c_i\}$
 - * If $r_i \in P_j$, add (c_i, c_j) to E

If N is the amount of contracts in S , and each contract mentions M items, the algorithm has a complexity of $O(MN^2)$.

5 Conclusion

This report defines policy evaluation contracts. Contracts can be matched with each other and a set of contracts can be satisfied when there are no required items. Based on a set of contracts, a dependency graph can be constructed that holds the valid configurations of the authorization infrastructure.

References

- [1] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.

- [2] K. Beznosov. Object security attributes: Enabling application-specific access control in middleware. *Proceedings of the 4th International Symposium on Distributed Objects & Applications (DOA)*, 2002.
- [3] ISO. ISO/IEC 10181-3:1996 information technology - open systems interconnection - security frameworks for open systems: Access control, 1989.