Difference in Security Policies for Dynamic Systems

Martin Kähmer
Institute of Computer Science and Social Studies, Dept. of Telematics
University of Freiburg, Germany
kaehmer@iig.uni-freiburg.de

ABSTRACT

Advances in three aspects of computing, namely reachability, pervasiveness, and autonomy, lead to highly dynamic systems, whose components increasingly manage themselves. However, their security and trustworthiness must always be evaluated in the user's context and in tight association with his personal goals and current preferences. Therefore, this paper defines an operator for policies to determine the difference in policies. This policy operator enables the user to check for additional permissions that would come along with the acceptance of a new or modified policy and can assist him in estimating the risk he puts his security or privacy in before adopting it.

1. DYNAMIC SYSTEMS

The growing potential to combine devices with different capabilities and purposes has led to the development of densely networked, decentralized systems. In these systems, components are becoming increasingly autonomous; they avidly collect data in various forms and adapt to their environment as well as to the users' requirements. Their changing and possibly conflicting demands lead to a continuous negotiation of requirements and ad hoc relationships. These features lay the technical foundation for highly dynamic systems and, thus, for a plethora of new applications building up an infrastructure of services.

When utilized for delivering tailored services to improve customer relationship in a future supermarket [7], for example, dynamic systems do not just allow for the delivery of universal services, such as product finders or information searches. By taking the customers' context into account, such as the products already picked or the current position of the cart, or considering name, age or purchasing history stored on customers' PDAs, the supermarket can offer individualized or even personalized services which adapt to the current needs of the customers [11].

As such systems are continuously becoming an essential feature of everyday life, the consideration of their security and privacy management is highly important. The acceptance of these systems is greatly affected by the capability of the user to keep control over the usage of his personal resources and to observe the systems' adherence to his preferences.

2. POLICIES: USER STILL IN THE LOOP

The ability of autonomous components to operate independently without constant human supervision and their adaptation to current context by automated reconfiguration seem to push the user out of the control loop. For their management, the user interaction appears to be dispensable.

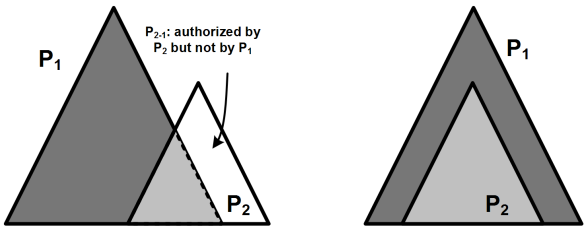
Yet, it is important to realize that the evaluation of security and privacy risks cannot be an issue for those expert security administrators alone, who set up some default guards constraining the system's behavior. Although the border of automation will be moved in such environments, the user cannot be completely released from the responsibility of formulating his goals and preferences as a personal security policy. Trustworthiness and risks cannot exclusively be assessed by central administrators, especially when his privacy is endangered.

Hence, the user needs to transfer his own policy stored on a suitable communication device, such as a personal PDA, to the system. However, his policy will often not be the only clincher for the system's security decisions. If there is a set of global security rules in place guarding the environment, the user will not be able to overrule it. The system will more likely incorporate the user's policy and subordinate its rules to the global ones. Similarly, the resulting policy shared by the system components may be a conjunction of multiple user policies, if the system is used by more than one user at the same time.

In consequence, the high dynamics of such systems forbid this shared policy to be permanently determined and, therefore, to be analyzed for the components' security behavior in advance (technical or legal policy enforcement is not within the scope of this paper; e.g. see [5] for further information). A user cannot predict the future system composition. Its parameters may change so quickly that a determined and subsequently analyzed policy becomes useless after minutes or even seconds.

More importantly, users are often unable to predict even their own future behavior. As surveys have shown, users take a very pragmatic approach to security. Willing to make security and privacy trade-offs according to the current situation or to the current communication partners, they adapt their own policies or spontaneously commit their resources and personal data to a foreign one to reflect the dynamics of such systems [11, 4].

Since users who have a method to observe and preview the result of their interaction make significantly less security-related operating errors [6], they need appropriate support for adapting the often complex policy sets and for evaluating the impact of the changes made. A step in this direction is the definition of the policy difference presented in the following chapter. This difference operator allows the user for the reduction of a modified or updated policy to only those parts that are relevant to him for his further security negotiations and decisions.



(a) $\mathcal{P}_2$ grants further actions, hence the (white) difference $\mathcal{P}_{2-1}$ is not empty.

(b) $\mathcal{P}_{2-1}$ is empty, i.e. $\mathcal{P}_2$ is completely covered by $\mathcal{P}_1$ and weakly refines it.

Figure 1: Difference and refinement of two policies.

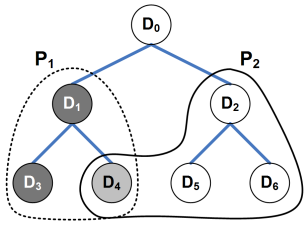


Figure 2: $\mathcal{P}_1$ allows some actions on D_1 and, thereby, D_3 and D_4, $\mathcal{P}_2$ on D_4, D_2, and both its children.

3. ANALYSIS OF POLICIES

Usage control policies integrate the diverse security policy concepts into a unified policy model [8]. With this, the user is not just able to declare who may access his resources, i.e. his own system components and his personal data. By means of obligations, he can also control their usage once they have been handed over to a foreign security domain. Combined with abstracting elements and categorizing them into hierarchies found in many access control languages, such policies are ideal for the intuitive specification of users' requirements in dynamic systems. Featuring both, IBM's policy language EPAL [1] builds the basis for our definition of policy difference, though the underlying concept can be easily transferred to other usage control languages.

For better understanding of the next section, some properties of the abstract syntax and semantic of EPAL are presented, but definitively without the claim of completeness. Hierarchies for users U, data D, purposes P and actions A build the vocabulary Voc together with conditions $C(Var)$, and obligations O. Similar to hierarchies, an obligation model defines an imply-relation on the power set of all obligations indicating that fulfilling a set of obligations implies fulfilling all of its subsets. Given a vocabulary, a policy $\mathcal{P} = (Voc, R, gc, dr, do)$ consists of a global enabler $gc \in C(Var)$, the default dr of ruling $r \in \{+, \circ, -\}$ according to "allow", "do not care", and "deny", a set of default obligations $d\overline{o}$, and an ordered list of rules $R \in \{U \times D \times P \times A \times C(Var) \times r \times \wp(O)\}$. A rule matches a request $q \in \{U \times D \times P \times A\}$, if its conditions are satisfied by the given assignment χ of environment variables, and all elements of the request are equal to or children of their rule counterparts. The first matching rule in $\mathcal{P}$ gives the result $eval_{\mathcal{P}}(q, \chi) = (r, \overline{o})$. For a complete specification, see [3].

3.1 Policy Difference

Whenever the user wants to update his own policy, or the shared system policy is to be changed, e.g. by merging it with a new personal one, the user is faced with the question of whether to accept this new policy version or to decline it and, perhaps, enter a further round of negotiation.

For this decision, both old and new policy versions are to be compared with each other. If the new version is equivalent to or even more restrictive than the old one, an automated process can be used to indicate acceptance. What happens, however, if none of these checks is positive, and the new version is more permissive? In this case, the user needs to be called in to decide on further actions. To prevent him from being overwhelmed by complex policy sets, the difference in the new and old policy reveals to him only those permissions granted by the former, but not by the latter, i.e. a reduction to only those permissions relevant for consideration at this time.

Taking up the shop example, a non-empty difference indicates to the customer the extra permission that is not covered by the regulations $\mathcal{P}_1$ he has set for the treatment of his data, but that the shop demands for the usage and tailoring of its services (cf. Fig. 1(a)). Otherwise, he can verify by an empty difference that the shop's policy $\mathcal{P}_2$ is more restrictive and, thus, safe to use (cf. Fig. 1(b)).

Fig. 2 shows an example reduced to the consideration of the data hierarchy: $\mathcal{P}_1$ allows some actions on data D_1, whereas $\mathcal{P}_2$ allows the same actions on D_2 and D_4. Computing the difference, written $\mathcal{P}_{2-1}$, reveals that $\mathcal{P}_2$ extends the permission to D_2 and its children D_5 and D_6, but not to D_4, as it is already covered by the permission on D_1 by $\mathcal{P}_1$. From the user's point of view, the permission on D_1 and D_3 taken by $\mathcal{P}_2$ (i.e. $\mathcal{P}_{1-2}$) are not of primary interest, because no possibly unintended permission is inserted.

Owing to the limited space, we assume a single vocabulary for both policies. In the style of [2], the notion of policy difference is defined as follows:

Definition (Policy Difference)**.** *Let two policies $\mathcal{P}_1$ and $\mathcal{P}_2$ be given with $\mathcal{P}_i = (Voc, R_i, gc_i, dr_i, d\overline{o}_i)$ for $i = 1, 2$ and equal vocabulary. Then, the rule set of the difference $\mathcal{P}_{2-1}$ gives the same request results as $\mathcal{P}_2$, but it is only enabled for those requests q and assignments χ of condition variables, for which one of the following statements holds, where $(r_i, \overline{o}_i) = eval_{\mathcal{P}_i}(q, \chi)$ for $i = 1, 2$:*

- $gc_1(\chi) = false \quad and \quad (r_2 \in \{+, \circ\} \quad or \quad \overline{o}_2 \neq \varnothing)$
- $\overline{o}_1$ *does not imply* $\overline{o}_2$
- $r_1 = \circ$ *and* $r_2 = +$
- $r_1 = -$ *and* $r_2 \in \{+, \circ\}$

A first implementation can be made using a brute force approach. One simply evaluates both input policies for any possible request q and any possible assignment χ and compares their rulings and obligation sets. A rule $(q, C_2, r_2, \overline{o}_2)$ generated from a given q and χ is part of $\mathcal{P}_{2-1}$, if policy $\mathcal{P}_1$ is globally disabled and, therefore, not active in context χ, but $\mathcal{P}_2$ does not prohibit the requested action or commits the requester to future actions. For all other q and χ, the generated rules are part of the difference, unless $\overline{o}_2$ does not contain any new obligation and is a subset of $\overline{o}_1$. In this case, it is up to the comparison of the rulings only. If r_1 is dominated by r_2, i.e. r_2 is more permissive, the corresponding rule is part of $\mathcal{P}_{2-1}$, too.

As this brute force approach tests all valid combinations of requests and condition variables, their amount grows exponentially with the number of vocabulary variables. Thus, a more efficient algorithm is desired, e.g. by grouping requests with the same matching rules or neglecting rules that will always be hidden by matching rules with higher priority, as Backes et al. propose for their efficient algorithm to test on policy refinement [2]. Exploiting the semantic of the policy language or rather its evaluation function to reduce the difference problem to only the relevant combinations is part of future work.

4. RELATED WORK

Currently, the comparison of such policies is made by a refinement check [3], which enables the automated verification that one policy also fulfills the other one or, in case of weak refinement, that the former is less permissive. For this test, an efficient algorithm is given in [2], but in case of verification failure, the user gets no hint of its cause. Furthermore, weak refinement can be mapped to the evaluation of the policy difference: if $\mathcal{P}_{2-1}$ is empty, then $\mathcal{P}_2$ is more restrictive and weakly refines $\mathcal{P}_1$ (cf. Fig. 1(b)).

For the limited name-space of P3P, the Privacy Bird [4] implements a refinement test. Preferences are specified as a selection from a predefined set of high-level rules the bird tests separately against the site's policy. If one of them is not refined, its description tag is shown to the user. Using policy difference, the conflicting parts are indicated to the user without any clever name-space separation ex ante.

There is ongoing work on the evaluation of policy changes and their compliance with meta rules, such as "separation of duty" [12], but research is focused on RBAC and corresponding algebras only.

Finally, there is an approach using model checking with dynamic logic [9]. While this approach allows for an elegant representation of policy changes, a formal model of the system is needed, which will be difficult, if not impossible to acquire in a highly dynamic system.

5. CONCLUSION AND FURTHER WORK

In this paper, we put forward the user's point of view on policies in dynamic systems. We argue that despite their autonomous behavior the components still need user's supervision. While the user adapts his usage control policies over time, he needs support for observing the possible impact of the changes. For this, we define the difference between two policy versions, which reveals exactly those permissions additionally granted by the new policy.

This definition is just the first step in the direction of usable policy management, and there is considerable work to be done. First, an efficient algorithm for computing the difference is to be developed. Subsequently, it will be transfered to syntax and semantic of NAPS [10], which enhances EPAL's algebra by closure under composition and conjunction, both of them being essential policy operations in dynamic systems. Finally, it still remains open as to how the difference can be presented in a suitable form to the user in order to assist him in his further security decisions.

6. REFERENCES

[1] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (EPAL 1.2). Submission to W3C, 2003.

[2] M. Backes, G. Karjoth, W. Bagga, and M. Schunter. Efficient comparison of enterprise privacy policies. In *Proc. of 2004 ACM Symposium on Applied Computing*, pages 375 – 382, 2004.

[3] M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In *Proc. of 8th European Symposium On Research In Computer Security (ESORICS)*, 2003.

[4] L. F. Cranor, P. Guduru, and M. Arjula. User interfaces for privacy agents. *To appear in ACM Transactions on Computer-Human Interaction*, 2006.

[5] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proc. of 10th European Symposium On Research In Computer Security (ESORICS)*, pages 98 – 117, 2005.

[6] J. Kaiser. Besteht eine Beziehung zwischen Nutzbarkeit und Sicherheit? *PIK "Sicherheit"*, 26(1):48 – 51, 2003.

[7] METRO Group. Future store initiative. `http://www.future-store.org/`, 2006.

[8] J. Park and R. Sandhu. The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128 – 174, 2004.

[9] R. Pucella and V. Weissman. Reasoning about dynamic policies. In *Proc. of 7th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS'04)*, 2004.

[10] D. Raub and R. Steinwandt. An algebra for enterprise privacy policies closed under composition and conjunction. In *To appear in Proc.of Int. Conf. on Emerging Trends in Information and Communication Security (ETRICS)*, pages 132 – 146, 2006.

[11] S. Sackmann, J. Strücker, and R. Accorsi. Customizing services in privacy-aware highly dynamic systems. *To appear in Comm. of the ACM*, Sept 2006.

[12] A. Schaad and J. D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proc. 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 13 – 22, 2002.