Monitors for Usage Control

M. Hilty[1], A. Pretschner[1], D. Basin[1], C. Schaefer[2], and T. Walter[2]

[1] Dept. of Computer Science, ETH Zürich, Switzerland
{hiltym,pretscha,basin}@inf.ethz.ch
[2] DoCoMo Euro-Labs, Munich, Germany
{schaefer,walter}@docomolab-euro.com

Abstract. Distributed usage control is concerned with controlling how data may or may not be used after it has been given away. One strategy for enforcing usage control requirements is based on monitoring data usage and reacting to policy violations by imposing penalties. We show how to implement monitors for usage control requirements using run-time verification technology.

1 Introduction

The vast amount of data collected in digital form necessitates controlling the usage of sensitive data. The use of *personal data* is governed by data protection regulations. Likewise, the protection of *intellectual property* such as copyrighted artworks or trade secrets is in the financial interest of the data owners. Usage control [8, 9] is an extension of access control that not only addresses who may access which data, but also what may or may not happen with the data afterwards. We study usage control in the context of distributed systems where the participating subjects can take the roles of data providers (who give data away) and data consumers (who request and receive data). When a data provider gives data to a data consumer, the latter must adhere to obligations, which are conditions on the future usage of data. Examples include "do not distribute document D," "play movie M at most 5 times," "delete document D after 30 days," and "notify the author whenever document D is modified."

There are two basic strategies that data providers can employ for enforcing obligations. With *control mechanisms*, they can restrict the usage of objects or ensure that certain actions are executed at a certain point in time, thus preventing obligation violations. The second strategy is based on *monitoring* whether an obligation is violated and penalizing the data consumer when this happens. This is similar to law enforcement where the police cannot always prevent people from breaking the law but fine or sentence delinquents when catching them. This strategy is implemented by *observation mechanisms*, which consist of two parts. Provider-side *obligation monitors* are used to decide whether an obligation is adhered to, and consumer-side *signaling mechanisms* notify the data provider about events that happen at the data consumer's side.

In this paper, we present the implementation of an obligation monitor that adapts run-time verification techniques. The obligation monitor has been de-

signed to monitor a wide range of usage control requirements that were identified in earlier studies. This is the first application of run-time monitoring to usage control to the best of our knowledge.

2 Background

Obligations The Obligation Specification Language (OSL) [6] is a language for expressing obligations in usage control. OSL is a temporal logic similar to LTL and includes constructs for expressions that frequently occur in usage control requirements and that are difficult to express with standard LTL operators. In particular, OSL is able to express cardinality conditions, conditions on the accumulated usage time, and both permissions and prohibitions. An obligation expressed in OSL formulates a property that every execution trace of a data consumer must satisfy.

We call an obligation o *fulfilled* in a trace t at time n if t with every possible extension after n satisfies o. In contrast, o is *violated* in t at time n if t with every possible extension after n does not satisfy o [5]. Violation and fulfillment correspond to the notions of bad and good prefixes introduced by Kupferman and Vardi [7]. We use them to decide the points in time when penalties should be triggered (i.e., when an obligation is violated) or when the monitoring of an obligation can be stopped (when it is either violated or fulfilled).

Related Work Monitoring as an enforcement strategy for obligations has been proposed by Bettini et al. [1]. There are many different approaches to the monitoring of temporal formulae. These approaches differ depending on the expressiveness of the input language and the techniques used for monitoring. General overviews of such systems are given in [2, 3]. In terms of the implemented techniques, we can differentiate between rewriting-based and automata-based approaches. Automata-based algorithms have an initialization overhead that results from building the automata, and this overhead is usually exponential in the length of the monitored formula. However, they are only linear in the size of the formula at run-time, whereas the rewriting-based algorithms do not have any initialization overhead but are less efficient at runtime [10].

3 Observation Mechanisms

An observation mechanism consists of a provider-side *obligation monitor* and a consumer-side *signaling mechanism*. The signaling mechanism observes the actions of a data consumer and informs the monitor about these observations by sending dedicated *signals*. If the monitor detects the violation of an obligation, the corresponding penalty is triggered. If an obligation is fulfilled or violated, its monitoring is stopped. The signaling mechanism may send signals corresponding to single actions or sequences of actions. In the latter case, obligation monitors

can also be deployed at the consumer side. In the extreme case, the whole monitoring functionality may be included in the signaling mechanism, which then informs the data provider in case an obligation is violated.

The signals received by the data provider must be trustworthy in the following sense: (1) the observations of the signaling mechanism must correspond to what really happens at the data consumer (correctness and completeness); (2) the signaling mechanism itself must not be tampered with; and (3) the signals must not be altered or blocked during transmission. How these guarantees can be provided is outside the scope of this paper. However, monitoring also makes sense even when only some of them hold. If signals cannot always be transmitted, for example, then the monitoring functionality can be integrated into the signaling mechanism. This way, the signaling mechanism can go online periodically and tell the data provider whether violations have occurred.

4 Obligation Monitors

Monitoring Algorithm The process of finding good and bad prefixes for temporal formulae has been studied in the discipline of run-time verification [2, 3]. An important criterion for selecting an algorithm was efficient support for all operators of OSL. While all OSL formulae can be translated into LTL formulae, direct translation results in poor performance. For instance, OSL can express exclusive permissions such as the obligation that a given data item may only be sent to two subjects. Enumerating all subjects that are not allowed, which is necessary when translating permission expressions to LTL, not only makes the resulting monitors potentially very large, but also poses problems when new subjects are dynamically added to the system. Thus, such exclusive permissions should be directly supported by the monitoring algorithm as well.

The algorithm we have chosen is based on the work of Geilen and Dams [4]. It is a tableau construction that constructs a timed automaton for every obligation. When a transition is not defined, this indicates the violation of the obligation, and when a special state is reached, then the obligation is fulfilled. We have extended the algorithm to efficiently support cardinality conditions by introducing dedicated counters, which are similar to timers. Further, we have introduced special support for the permission operators of OSL.

Implementation We have prototypically implemented the obligation monitor in Java. Tests have shown that the following factors impact the monitoring performance: the number of simultaneously monitored obligations, the size of the obligations, the frequency at which signals are received, and the duration of the clock cycle. The number of actions that are prohibited by an exclusive permission does not have an effect on the performance of the monitor, and neither does the size of the number in a cardinality condition. Determining the exact performance of the monitor is outside the scope of this paper, especially as the implementation itself is not yet optimized towards heavy workloads.

However, tests on a Pentium M with 1.6GHZ and 1GB of RAM where 1000 obligations were monitored simultaneously and signals were sent every 150ms on average showed that the monitor was able to process all signals within less than 100ms. The fact that we have achieved this performance on a low-end system, without dedicated performance optimizations of the code and the platform, suggests that creating industrial-strength obligation monitors is not an illusion.

5 Conclusion

Observation mechanisms are an important means for usage control enforcement. A widespread adoption of observation mechanisms will lead to a wider applicability of usage control enforcement. We have characterized those mechanisms and have shown how to adapt existing run-time verification techniques for obligation monitoring in usage control. We have also built a prototype of an obligation monitor based on these ideas. Future work includes creating a performance-optimized implementation of the monitor, determining how trustworthy signaling mechanisms can be built, and integrating observation mechanisms into business information systems.

References

1. C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *Journal of Network and System Management*, 11(3):351–372, 2003.
2. S. Colin and L. Mariani. *Model-Based Testing of Reactive Systems*, chapter 18: Run-Time Verification, pages 525–555. LNCS 3472. 2005.
3. N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, 30(12):859–872, 2004.
4. M. Geilen and D. Dams. An on-the-fly tableau construction for a real-time temporal logic. In *Proc. 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1926, pages 276–290, 2000.
5. M. Hilty, D. Basin, and A. Pretschner. On obligations. In *10th European Symposium on Research in Computer Security*, LNCS 3679, pages 98–117, 2005.
6. M. Hilty, A. Pretschner, C. Schaefer, and T. Walter. A system model and an obligation lanugage for distributed usage control. Technical Report I-ST-20, DoCoMo Euro-Labs, 2006.
7. O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19:291–314, 2001.
8. J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.
9. A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *Communications of the ACM*, September 2006.
10. G. Roşu and K. Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engineering*, 12:151–197, 2005.