KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 25–33

AN OVERVIEW OF DISTRIBUTED USAGE CONTROL – EXTENDED ABSTRACT –

ALEXANDER PRETSCHNER

Abstract. Usage control generalizes access control to what happens to data in the future ("delete after thirty days," "do not copy," "notify owner upon access.") Distributed usage control is about defining and enforcing usage control requirements on data after giving it away. It is relevant in the areas of data protection, the management of intellectual property, the management of secrets, and compliance with regulations. In this extended abstract, we provide an overview of the field. We introduce fundamental concepts, requirements, policy specifications, policy analyses, dissemination models, the enforcement of usage control requirements at different levels of abstraction, and the challenges ahead.

1. Introduction

Ever increasing amounts of digital data require procedures and mechanisms for secured access to and usage of that data. Because we live in an interconnected world, this problem not only extends to the original provider of a data item, but also to all those parties who have, transitively, received a—possibly modified—copy of this data item. The subject of *usage control* [13, 6] is the definition and enforcement of access control requirements that relate to actions and state information before data is released, and also to usage control requirements that relate to actions and state information after the data is released. We distinguish between two kinds of requirements, *provisions* that reflect access control requirements, and *obligations* that reflect requirements on the future usage [6]. Usage control requirements include permissions ("song may be played at most twice," "document must not be printed," "financial statements must be retained for at least five years") and duties ("data owner must be notified upon each access," "song must be paid for after listening five times," "data must be deleted after thirty days"). While both kinds of requirements are stipulated in *usage control policies*, we will only be concerned with obligations in this extended abstract.

Key words and phrases. Access Control, Usage Control, Enforcement, Policies, Trust.

This work was supported by the FhG Internal Programs under Grant No. Attract 692166 as well as by the EU under the project FP7-IST-IP-MASTER.

©2007 Babeş-Bolyai University, Cluj-Napoca

1.1. **Relevance.** Usage control is relevant in at least the areas of privacy, the management of intellectual property, the management of secrets, and compliance with regulations. In terms of privacy, users may want, among other things, their medical information, loyalty card records, telecommunication connection records, and banking information to be kept under wraps (sometimes they may not, as the current immense popularity of largely unprotected social network sites shows). At least in Europe, in addition to EU-wide legal requirements, current developments clearly indicate a potential or actual political fallout when large sets of citizen or customer data become available.

In terms of intellectual property, business processes are increasingly implemented in a distributed manner, fueled by an increasing trend to outsourcing. To create added value, this requires the exchange of usually confidential information, including blueprints, results of experiments, and the like. Companies may have an interest that such data is solely used according to their expectations. As far as digital rights management is concerned, artists may be granted a vested interest in receiving royalties for their artwork.

In terms of secrets, most administrations, state departments, intelligence agencies and the military may want to have control over how specific information is disseminated.

Finally, usage control is becoming increasingly mandatory. Regulations such as the EU directive 95/46/EC require data to be collected and used according to a specified purpose; the Sarbanes-Oxley Act (SOX) requires specific data to be retained for five years; and the US Health Insurance Portability and Accountability Act (HIPAA) stipulates strict documentation requirements on the dissemination of data [9].

1.2. **Big Picture.** We assume the following scenario. A *consumer* wants to get access to data. To do so, negotiations on the terms and conditions of usage take place with the *provider*. The negotiations result in a *policy* that encodes rights and duties related to the data item and that reflects regulations, non-disclosure agreements, or any legally binding contract. After checking if the consumer can enforce the policy, the provider sends both data and policy to the consumer. In order to prevent simply throwing away the policy, this is likely to involve some encryption mechanism. At the consumer's side, the data is stored so that only well-specified *enforcement mechanisms* can access it. This, again, is done by cryptographic means; such ideas are currently implemented at different levels of abstraction, including the operating system where so-called data caging takes place. The mechanisms are configured by the policy. Consumers attempt to use (render, process, execute, disseminate) the data. The enforcement mechanisms then either check if this is allowed, by turning the attempted usage into an actual usage, or if a policy violation has occurred, report the violation.

1.3. **Overview.** Challenges related to usage control roughly encompass the specification of requirements, including their evolution upon dissemination; the enforcement of respective policies; and the assessment of how easily these enforcement mechanisms can be circumvented. These issues are considered in the remainder of this paper, mostly from the perspective of our own work, which is explained by the very nature of this article.

2. Requirements, Policies, Analysis, and Evolution

Usage control requirements can be classified into permissions and duties. Both kinds usually specify conditions in which data may be used or in which actions need to be taken. Conditions relate to time ("within thirty days"), cardinality ("copy at most three times"), purpose ("personal use only"), events ("upon access"), and both the technical ("Windows RMS must be installed") and organizational ("virus scanner databases need to be updated every week") environments [7, 16].

Many policy specification languages have been defined (including [1, 4, 24, 23, 7]), a few of them also with formal semantics.[1] This formal semantics, by its very nature, however is restricted to the aspects that are captured by temporal, modal, and first-order logics. Propositions ("print," "copy," "delete") are, essentially because of their high level of abstraction, much harder to define precisely.

In current languages, policies are mostly specified in terms of events ("play," "copy," "delete"). For well-specified rendering devices in a DRM context that make use of a standardized ontology, this is often sufficient. However, as we will see, usage control requirements can be enforced at different levels of the software stack. One level includes the operating system; relevant events are then system calls. Exhaustively defining the notion of "deletion" in terms of sequences of system calls (unlink, mv to a null device, overwrite, ...) seems like a Sisyphean endeavor. One possible remedy is to track data flow through the system, and encoding as abstract state the (overapproximated) mapping from data containers to data items. Deletion in state-based terms means that in a given state, no container may contain the data item. Similarly, prohibiting dissemination can be expressed by stating that the a data item is in at most one data container.

It is useful to distinguish between three kinds of policies. *Specification policies* declaratively state what should be the case ("no non-anonymized data must leave the system"). The decision of how to enforce a policy—by modification, inhibition, or execution—is done in *implementation policies*. The example policy can be enforced in at least two ways: by modifying the data record's name, birthdate, and address fields into blanks; and by simply blocking all data packets which are not anonymized in the sense that their name, birthdate, and address fields are not blanks (which of course is a very rough definition of anonymization).

[1] Note that we tacitly assume policies to be data-centric rather than server-centric; this is not a conceptually fundamental distinction, however.

Implementation policies usually come in the form of classical operational Event-Condition-Action rules, where the condition must be specified over the present and the past rather than the future (otherwise it could not be checked). Finally, *configuration* policies are rights objects that can directly be understood by implementations of enforcement mechanisms. In model-driven engineering terms, configuration policies are platform-specific while specification and implementation policies are platform-independent.

At the abstract level, checking if an implementation policy makes sure that a specification policy is fulfilled amounts to checking entailment of logical formulae. Respective reasoning technology can also be used to assess the consistency or subsumption relationships between policies [19].

When data is disseminated, some policy will have to be associated with it as well. In some sense, this policy should at most be a "strengthened" version of the original policy—otherwise, a subject could send the data item to itself, together with a relaxed policy that allows everything and requires nothing. Strengthening permissions can be done by restricting them; strengthening duties should then, dually, mean that they are increased. By ordering events in lattices and specifying lower and upper bounds for both permissions and duties, one can uniformly express the strengthening of policies as a combination of logical entailment and interval reduction [20, 19].

Widely applicable usage control frameworks must cater to the problem of policy management as well, including overwriting and revoking policies as well as handling conflicting policies.

3. Enforcement

The fundamental problem of enforcing usage control requirements is that a data provider usually has no control over nor inspection into the IT infrastructure of a data consumer (note that the roles dynamically change upon re-distribution of data). In order to prevent simple interceptions or retrievals of sensitive data, data must be stored and transmitted in encrypted form. Moreover, at the consumers' side, tamper-proof and trustworthy monitoring and control devices must be in place. These devices can come as special rendering software (such as the Adobe Acrobat Reader in conjunction with the respective rights management system), or as add-ons to a system, similar to malware intrusion detection systems.

3.1. Reactive and Preventive Enforcement.

One way of enforcing usage control requirements is *by observation*, or reactive. Provided that adherence to policies can be monitored, one can at least detect the violation of a policy and react by undoing the violating action, by penalizing the wrong-doer, or by performing compensating actions. This is similar to how human law enforcement works [14].

Another way of enforcing respective properties is *by control*, or preventive. The goal here is to prevent a policy violation from happening. Rather than observing

events post factum, one must usually observe *requests*, anticipate the events that would be a consequence of these requests, and then either inhibit the request, modify it, or execute some action. For instance, a policy "no non-anonymized data may leave the system unnoticed" can be enforced by dropping the request that asks for a non-anonymized data item (inhibition); by anonymizing the data item (modification); or by logging the event that the data item was released (execution) [17, 16]. Whether a reactive or a preventive enforcement strategy is to be chosen cannot be decided in general; this depends on the trust relationship of provider and consumer, and also the value of the data items that are exchanged [18].

Particularly enforcement by control can quickly become impossible. This is often the case when media breaks occur. Once a song is played, i.e., transformed into sound waves and has thus left the scope of a controlled (the boundaries of which need to be defined), one can externally record and replay it. Similarly, a document can be printed or photographed from a screen, thus rendering most control devices useless. Watermarking schemes have been developed for these situations. Their purpose is to subject such data to the possibility of more or less random observation mechanisms, and hoping that non-rightful possessors of data will be deterred.

On the other hand, enforcement by control may turn out to be too intrusive. If usage control is applied to Java API calls, for instance, then one can of course block calls that are considered sensitive, e.g., calls to text message APIs in mobile phones. However, unless the original system is programmed defensively and always anticipates the potential failure of a method invocation, this is likely to lead to a crash—many exceptions tend to be caught only at the bottom-most stack frame. This problem turns out to be particularly challenging in asynchronous communication infrastructures such as service-oriented architectures.

In any case, there never is complete security in life or it would be too restrictive or too expensive anyway. We believe that increasing the barriers for accessing and using data in non-permitted ways is sufficient in a large majority of cases. Security in the sense of usage control must be subject to risk analysis, and we believe in the idea of *just-right-security*, similar to the analogous concept of *just-right-reliability* [12]. Needless to say, we are of course aware that any solution is most likely to encompass both technical and organizational means.

3.2. **Signaling, Monitoring, and Enforcement proper.** We have alluded above that enforcement mechanisms can abstractly be perceived as sets of Event-Condition-Action rules. Conditions relate to the current state of the system (which in many cases encodes past events); events are triggers; and actions define whether or not an event is inhibited (which requires the distinction into requests and actual actions), modified, delayed, or if another action is to be executed. From an architecture perspective, this necessitates three components: *signalers* that make events visible so that a *monitor* can check if the condition is true, and *enforcement components*

that perform the respective action. Consider a policy that requires a movie to be played at most three times before it is paid for. A signaler must provide information on payments and whether the movie is played or attempted to be played; a monitor must count the number of times the movie is played; and the enforcer must then issue a payment or block the attempt to play the movie. Note that these components can but need not necessarily reside on one single machine [15].

Monitoring can be achieved by many different technologies, including rewriting logics, state machines that implement policies, and complex event processing technologies (where it is then usually called aggregation). For signalers, it is likely that the subsystem that generates events must be instrumented; this is sometimes but not always the case for monitors.

3.3. **Levels of Abstraction.** Enforcement, which from now on we understand to include signaling and monitoring, can be implemented at different levels of the software stack. Consequences include both the universe of discourse of the policy language (for instance, messages vs. files and natural language terms such as deletion vs. unlink()) and—of course depending on the chosen trust or attacker model—the guarantees that can be given. It is likely that a combination of several—rather than one single—enforcement mechanisms at different levels will be necessary to provide guarantees (e.g., dissemination requirements relate to both files and screenshots).

The lowest level at which software-based usage control can be applied is the *CPU/virtual machine*: specific calls or references to memory cells can be blocked or modified. At this level, data flow within an application (or business logic) or between processes can be controlled or at least monitored. If the content of a file is written to a specific memory location, for instance, then subsequent reading operations can be detected.

The next level is the interface to the *operating system*. At this level, system call interposition can take place: system calls that access specific files, for instance, can be monitored and possibly modified or blocked. Re-distribution at a rather rough level can be controlled: if a process has accessed a file, it is possible to forbid all subsequent communication with other processes, file systems, networks, etc. This is likely to be considered an impediment by the users. Examplary base technologies include systrace [22] for OpenBSD and the Detours framework for Microsoft Windows [8].

At the level of the *runtime system*, calls to libraries can be monitored, blocked, and modified. Specific library calls and their parameters for communication can be prohibited, for instance, which provides a more fine-grained control than prohibiting all future communication. One example is the Polymer system that modifies Java byte code on the go [5].

At the level of dedicated *application wrappers*, applications can be controlled on the grounds of abstractions that relate to the specific application. As an example,

the UNO framework for OpenOffice allows to control copying text between specific documents rather than generally prohibiting copy and paste.[2]

At the level of *applications*, all kinds of usage control can be implemented. However, the question arises how the guarantees can be assessed. This is of course simpler if pre-defined components are installed at the consumer's side and data is only given to the client if the respective components are in place.

If we distinguish between end-user applications and *infrastructure applications*, such as the X11 server or the window manager, then control can also be exercised at the level of infrastructure applications. Controlling access to clipboards, for instance, can also take place at the level of the X11 server. Another example are *data base systems* with usage control mechanisms in place [2].

Usage control can also be implemented at the levels of wrappers for *services* [9, 3, 18]. The AXIS framework, for instance, allows the definition of handlers that intercept incoming and outgoing messages. Wrappers can also be written in an ad-hoc manner and are likely to operate at the level of messages (but need not necessarily, depending on the implementation).

Finally, usage control that operates on messages can be implemented at the level of the *enterprise service bus*. This is the least intrusive yet most vulnerable approach: XML tags can simply be compromised. At the level of an orchestration engine for service-oriented architectures, reactions to a policy violation can be initiated.

Note that in general the step from a lower-layer enforcement framework to a higher-layer enforcement framework is possible but non-trivial, particularly so if information flow is to be taken into account. One can monitor both the explicit and the implicit flow of sensitive data through a Java program. However, if this data is to be rendered on a screen, an AWT or Swing method will be invoked which will then call some native routines which, in case of a Unix system, will invoke X11 libraries. Respective interfaces that help identifying the flow of sensitive data across the levels must be defined.

3.4. **Assurance.** Among other things, the above logical architecture requires signalers to be complete and trustworthy. There must be means to make sure that signalers are not simply switched off. Monitors must be guaranteed not to miss events, and they must also be guaranteed to run as long as usage control is, according to the provider, to be exerted. Finally, the actions executed by a mechanism must be definitive and not easily overwritten.

Data providers likely want to know if a suitable enforcement framework is in place at the consumer's side [17]. Upon negotiation [21] over a data item to be released, the provider wants to know if the consumer can enforce the requirements

[2]The latter is all that can be expected at the level of the X11 server because the X11 server knows about windows, possibly mapping them to processes, but not about single documents that are simultaneously opened by one process.

as specified in the policy. At the logical level, this again boils down to an entailment problem [17]. At the more technical level, we believe that remote attestation on the grounds of trusted computing technology [3] is a promising candidate technology in this respect.

A further challenge obviously relates to the management of cryptographic keys that are the prerequisite for avoiding the simple interception and unconstrained usage of data before it enters the respective enforcement mechanisms.

4. Conclusions

Distributed usage control is about making sure that a data consumer handles data according to the rules as set forth by the data provider. Without entering the moral discussion of whether or not this is always desirable, we see usage control as an indispensable enabler when it comes to data protection or privacy, the management of intellectual property in distributed business processes, the management of secrets, and compliance with data-related regulations.

In this article, we have provided a big picture of the field, ranging from the model world (requirements, policies, analysis problems) to implementations of enforcement mechanisms that, in addition to fulfilling their functional specifications, must also be tamper-proof.

There are many open research, engineering and business problems left. Starting with the latter, it is not entirely clear what adequate business models are, and who is going to pay for usage control—the idea of usage control as an enabler must also be translated in business terms. Research problems in particular include problems related to suitable trust models; policy management schemes in different organizational settings (enterprises; the Internet); the conceptually clean connection between different levels of abstraction for enforcement; the difficult problem of de-classification when information flow is tracked; a better understanding of information flow, including quantitative measures [11, 10]; and the question of how we can, qualitatively or quantitatively, measure the guarantees that a usage controlled system can provide. Finally, engineering problems relate to efficient signaling, monitoring, and enforcement technologies at different levels of abstraction, secure key storage, and the question of how it can be assured that a particular mechanism is in place at the consumer's side.

5. Acknowledgment

We would like to thank T. Walter and C. Schaefer from DOCOMO Euro-Labs for many fruitful discussions. D. Hupel, C. Lorini, O. Maschino, M. Radulescu, and S. Willenbrock commented on an earlier version of this article.

References

[1] Open Digital Rights Language - Version 1.1, August 2002. odrl.net/1.1/ODRL-11.pdf.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *VLDB*, pages 143–154, 2002.
[3] B. Agreiter, M. Alam, R. Breu, M. Hafner, A. Pretschner, J. Seifert, and X. Zhang. A Technical Architecture for Enforcing Usage Control Requirements in Service-Oriented Architectures. In *Proc. ACM workshop on Secure Web Services*, pages 18–25, 2007.
[4] M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In *Proc. ESORICS*, LNCS 2808, pages 162–180. 2003.
[5] L. Bauer, J. Ligatti, and D. Walker. Composing Security Policies with Polymer. In *Proc. PLDI*, pages 305–314, 2005.
[6] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proc. ESORICS*, Springer LNCS 3679, pages 98–117, 2005.
[7] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A Policy Language for Distributed Usage Control. In *Proc. ESORICS*, pages 531–546, 2007.
[8] G. Hunt and D. Brubacher. Detours: Binary Interception of Win32 Functions. In *Proc. USENIX Windows NT Symposium*, pages 135–143, 1999.
[9] V. Lotz, E. Pigout, P. Fischer, D. Kossmann, F. Massacci, and A. Pretschner. Towards Systematic Achievement of Compliance in Service-oriented Architectures: The MASTER approach. *Wirtschaftsinformatik*, 50(5):383–391, October 2008.
[10] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In *Proc. PLDI*, pages 193–205, 2008.
[11] C. Mu. Quantitative information flow for security: a survey. Technical Report TR-08-06, Department of Computer Science, King's College London, September 2008.
[12] J. Musa. *Software Reliability Engineering*. AuthorHouse, 2004.
[13] J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.
[14] D. Povey. Optimistic security: a new access control paradigm. In *Proc. workshop on new security paradigms*, pages 40–45, 1999.
[15] A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *CACM*, 49(9):39–44, September 2006.
[16] A. Pretschner, M. Hilty, C. Schaefer, F. Schütz, and T. Walter. Usage Control Enforcement: Present and Future. *IEEE Security and Privacy*, 6:44–53, July/August 2008.
[17] A. Pretschner, M. Hilty, C. Schaefer, T. Walter, and D. Basin. Mechanisms for Usage Control. In *Proc. ASIACCS*, pages 240–245, 2008.
[18] A. Pretschner, F. Massacci, and M. Hilty. Usage Control in Service-Oriented Architectures. In *Proc. TrustBus*, pages 83–93, 2007.
[19] A. Pretschner, J. Rüesch, C. Schaefer, and T. Walter. Formal Analyses of Usage Control Policies. In *Proc. AReS*, 2009.
[20] A. Pretschner, F. Schütz, C. Schaefer, and T. Walter. Policy evolution in distributed usage control. In *Proc. 4th Intl. Workshop on Security and Trust Management*, pages 97–110, 2008.
[21] A. Pretschner and T. Walter. Negotiation of Usage Control Policies—Simply the Best? In *Proc. AReS*, pages 1035–1036, 2008.
[22] N. Provos. Improving host security with system call policies. In *Proc. SSYM*, pages 257–272, 2003.
[23] W3C. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, 2005.
[24] X. Wang, G. Lao, T. DeMartini, H. Reddy, M. Nguyen, and E. Valenzuela. XrML – eXtensible rights Markup Language. In *ACM workshop on XML security*, pages 71–79, 2002.

Fraunhofer IESE and TU Kaiserslautern, Germany
E-mail address: alexander.pretschner@iese.fraunhofer.de