Cross-Layer Data-Centric Usage Control

Enrico Lovat

enrico.lovat@in.tum.de
Technische Universität München, Germany
Supervisor: Prof. Dr. Alexander Pretschner, TUM

Abstract. Usage control (UC) is concerned with what happens to data after access has been granted, and is usually defined on the grounds of *events* that, somehow, are related to data. Data may assume different representations within the system, possibly at several levels of abstraction. This research aims at extending a generic event-driven UC model and its language [1] by the explicit distinction between *data* and *representations* of data. The resulting system will be able to enforce policies for single representations (e.g., delete f1.txt after thirty days) as well as for all representations of the same data at once (e.g., if f2.txt is a copy of f1.txt, also f2.txt will be deleted). To this end, different data-flow tracking approaches will be investigated. The result will be implemented, instantiated and evaluated in terms of security, precision and performance.

1 Introduction

The ever-growing sharing of digital content in the last decades demands better technologies to control what may happen to data after access to it has been granted. Usage Control (UC) [2] is an extension of access control that addresses exactly this issue. Examples of UC requirements are "delete this mail after 30 days", "don't copy this picture" or "notify owner of data upon access". In literature, UC enforcement mechanisms are defined on the ground of *events*, leveraging techniques like complex event processing [3] and runtime verification [4].

With this work, I want to investigate a different aspect of the problem: the *data* dimension. To enforce a requirement such as *"don't copy this picture"* one must consider that *this picture* may exist in multiple representations within the system, potentially residing at different levels of abstraction, like network packets, Java objects, window pixmaps, data base records, or files. Some UC requirements, usually related to *confidentiality* properties, address all the representations of the same data at once ("this picture cannot leave the local system", no matter which copy), whereas other requirements, usually related to *integrity* properties, refer to one (or some) specific representation(s) ("content of file /etc/passwd cannot be modified", but a copy of it could). The general research question addressed by this work is *"How can usage control requirements be expressed and enforced on different representations of data within a system, at and across different layers of abstractions?"*. I plan to answer it in three steps:

1) Extending a generic UC model [1] with Data Flow Tracking (DFT) features to relate the different representations of data within the system. To this

end, I plan to leverage existing work [5, 6] and taint-propagation tracking techniques from the information flow analysis area. The result will be a generic model instantiatable at arbitrary layers of abstraction in the system.

2) Define a generic model for tracking flows of data across different instances of the model defined in step 1, possibly at different layers of abstraction (*cross-layer data-flow tracking*). This will support integration of enforcement mechanisms at different layers to achieve system-wide usage control.

3) Refine the results of steps 1 and 2, in terms of precision e.g. considering DFT approaches that accounts for structural or quantitative aspects of data. This is motivated by preliminary results showing that the overapproximation introduced by basic taint-propagation becomes quickly prohibitive. In summary, **The goal of this research is to define, formalize, implement and evaluate a usage control model which supports data flow tracking at and across different layers of abstraction.**

The remaining of this paper is organized as follows: in §2 I compare my work to existing solutions; in §3 I describe a use case of my system; in §4 and §5 I discuss the different steps of my research plan and some preliminary results.

2 Related Work & Expected Contribution

The subject of my research is the combination of data flow monitoring with usage control, a policy language, and a prototype enforcement infrastructure.

A number of policy **languages**, like [1, 7, 8], has been proposed before, but none of them addresses the data dimension like mine will do; with them one can define UC restrictions for specific representations of data rather than all of them, and their semantic models ignore data flows. **Enforcement** of UC requirements has been done at many levels, like the Operating System (OS) level [5,9], the X11 level [6], for Java [10] and machine languages [11], for dedicated applications [12] and in the context of DRM [13]. These works focus on just one dimension of the problem, either data flow tracking or event-driven usage control. My approach, in contrast, tackles both at the same time, and since it is layer-independent, it can possibly be instantiated at each of these layers.

In terms of general-purpose **usage control** models, I see similarities with the models underlying XACML [14], Ponder2 [8] and UCON [2]. The first two, however, do not provide formalized support for cardinality or temporal operators (free text fields exist, but the respective requirements are hard to enforce). UCON supports complex conditions and has been used in applications at different level of abstraction [15], but assumes that data never leaves the data provider realm, implicitly making UCON policies device-dependent.

Concerning **data flow** tracking, my approach restricts the standard notion of information flow analysis, which also caters to implicit flows and aims at non-interference assessments [16]: my system detects only flows from one representation to another. This explains the choice of the term "data" flows rather than "information" flows. Moreover, even if I plan to leverage results of static analyses, like [17], I want to monitor these flows at runtime. Similar implementations have been realized for OS [5], X11 [6] and Java [10]; these works, however,

only address DFT without full usage control. Finally, I plan to generalize the idea for quantitative DFT presented in [18] to multiple layers of abstraction.

Gap Analysis and Contribution. Every related work I am aware of is comparable to my approach either in terms of usage control (e.g. policy expressiveness) or data flow tracking: none copes with both *at the same time*. Moreover, in contrast to mine, every "layer-independent" solutions I found in literature that consider data representations at multiple layers of abstraction, in facts tackle the problem only at *one* very low level, like the CPU or the hypervisor. I see my research as the first model and language for general purposes data-centric usage control that supports specification and enforcement of policies for all the representations of data within a system across several layers of abstraction.

3 Use Case Scenario

This scenario relies on instantiations of the model for a mail application (Mozilla Thunderbird, TB) and its underlying OS (MS Windows). Imagine Alice wants to send some sensitive content (text and pictures) to Bob via mail, while setting some restrictions on its usage such as *"The content of this mail cannot be printed, forwarded nor saved on removable devices and it must be deleted within 30 days"*.

When Bob opens the mail, he tries to print it, but the action is forbidden; similarly, the "Forward" button is disabled. Bob then creates a new message and drag&drops some content from Alice's mail to it. This action is allowed, but in order to prevent violations of the "no-forward" policy, the new mail is also not allowed to be sent. This is enforced thanks to the data-flow tracking component, which recognizes the two mails as representations of the same data. **(Step 1)**

Afterwards, Bob tries to save the attached picture to his USB stick. The policy that forbids it cannot be enforced by TB, because file management is an OS duty. Hence, after Bob chooses where the file should be saved (but before the actual saving), the monitor for TB notifies the OS monitor about the policy. Only then TB starts to save, but, as required, the OS denies it. **(Step 2)**

Bob then saves the picture to a local folder. When he browses it with the file explorer, the icon of the picture file is replaced by its thumbnail preview, which is automatically generated by Windows and stored in a hidden file (Thumb.db) in the same directory. After 30 days, all the copies of the picture, identified by the data-flow tracking system, are deleted. Thumb.db, however, containing only a *small amount of data* of the original picture (plus the thumbnail of many other files), is not considered as a copy and thus it is not deleted. **(Step 3)**

Note that, under different assumptions on the environment, usage control can be used against *deliberate* attackers (like Bob) or to mitigate *unintended* disclosures (e.g. Alice preventing herself from disclosing sensitive data by accident).

4 Research description

Step 1: Usage Control & Data Flow Tracking In the example above, Alice formalizes her requirements in a language that supports UC constraints specifi-

cation, like "delete within 30 days", while, at the same time, addressing abstract data (the content of a mail) in lieu of specific representations of it (a mail, a file, a window, etc.). She may also want to express data-dissemination restrictions, like "this data should never leave the local system", regardless of which sequence of events leads to the disclosure. The definition of such language and the respective system model are the goals of this part of my research, which addresses the question "*How can we express and enforce data-centric UC requirements?*"

To answer it, I will augment a policy language [1] with special parameters to distinguish data from representations, plus additional operators to constrain *data-dissemination*. The semantics of these new artifacts will be grounded on a simple model for taint-based DFT. I also plan to generalize the result into a modular architecture for generic UC systems, that supports component replacement, e.g. one DFT analysis with another (cf. Step 3). This model will then be instantiated for multiple layers, possibly extending existing solutions ([5, 12]).

Step 2: Cross-level model Tracking data flows across different layers is fundamental, because the system should never lose control over data once it changes representation. Imagine a policy like *"only one copy (=file) of picture p can exist in the system"*; at the OS level, every *write* system call that (potentially) transfers part of p to a new file must be forbidden, including those generated by a user saving to file f a screenshot of window w where p is rendered.

Thus, the second step of my thesis will be the definition of a UC model that relates representations of data at different levels, like f and w. The problem is far more complex than "just connecting" data representations at different levels. A relation between events must also be defined, because the execution of an action (like "save as" in Thunderbird) may result in several events at another level (e.g. "open","write" and "close" system calls at the OS level). This introduces additional problems, e.g synchronization and race-condition issues.

The fundamental question addressed at this step is *How can we relate data representations and events at different levels of abstraction?* I plan to tackle it with a bottom-up strategy: firstly, I will connect two specific existing implementations (for Thunderbird and Windows), looking for a common shared knowledge-base on top of which the cross-level relationship can be defined. Then, I will extend this model by adding, one-by-one, other layer-specific instantiations of the model of step 1. Existing implementations offer the advantage of knowing a-priori when and where data flows from one level to another.

Finally, I will try to leverage the result to generalize the model for arbitrary number and type of levels. The expected outcome is a framework for dynamically connecting arbitrary instantiations of the model; new monitors and enforcement mechanisms can be added at any time and share usage information with existing components, working together for a single system-wide enforcement.

Step 3: Data Flow Tracking Refinement Preliminary results (§5) show that the overapproximation induced by the taint-based data-flow tracking introduces enough false positives to compromise the overall functionality of the system. That is why, as third step, I will refine my data-flow tracking precision. The fun-

damental questions addressed here are: "*How can we measure the exact amount of data flowing within the system? How should such value be interpreted?*"

To answer this question I plan to a) augment the taint-based analysis of step 1 with quantitative aspects and b) consider the structure of data representations. In terms of a), the benefit of measuring the amount of data stored in each representation is twofold: on one hand it allows specification of *quantitative* policies such as *"if a file contains more than 5% of sensitive data, then delete it upon logout"*; on the other hand, it can be used as a possible declassification criterion.

Note that usually quantitative measurements are used to determine how many bits of information flow from inputs to outputs in a program [18]. These approaches cater to both explicit flows and control-flow dependencies. Mine, in contrast, applies to generic reactive systems where control-flow information may not be available, and thus it copes with explicit flows only. This does not make quantitative analysis a trivial task: consider, for instance, a representation resulting from the merging of other two; a precise estimation of the amount of data stored in it may require the whole history of transfers of the secret across the system [19]. And even though every kind of information can be measured in bits, different encodings may result in significantly different sizes for representations of the same data, especially if they reside at different levels of abstractions.

In terms of b), I want to refine the taint-propagation by considering the structure that data exhibit when stored by particular events in special types of containers (e.g. a ZIP archive file). In this case, the taint-propagation in case of a split-event (e.g. decompress), will introduce a lot of useless overapproximation.

Evaluation and Limitations At the end of each step, I will evaluate the results in terms of security, precision and, when possible, performance. According to preliminary results, the overhead of the monitoring infrastructure is above two orders of magnitude. Given the expected complexity of the outcome, my goal is to achieve, at the end, a usable system. Hence, fine-grained tuning and extreme performance optimizations will be analyzed, but their implementation is out of scope. Similarly, policy management and policy deployment, as well as details on distributed aspects of usage control and guarantees (enforcement mechanisms are up and running, not tampered with, etc.) will not be part of this research.

5 Preliminary results & Remaining Objectives

Preliminary results have been published in [20, 21] and cover most of the work described in step 1. A model for data flow tracking refinement based on quantitative measurements (step 3a) has also been developed and accepted for publication [19]. Currently, I am working on a generic model for cross-layer data flow tracking (step 2) and on a model for structured data-flow tracking (step 3b). I plan to be done with these two tasks by spring 2014. By summer 2014 I plan to have a complete usage controlled system, with instances of the generic model developed in step 1 for the operating system and for some specific applications, all working together according to the cross-layer model developed in step 2. Meanwhile, I am writing my thesis, which I plan to defend in fall 2014.

Acknowledgments This work was done as part of the DFG's Priority Program SPP 1496 "Reliably Secure Software Systems," ref. numbers PR-1266/1-2.

References

1. M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In *Proc. ESORICS '08*, pages 531–546.
2. J. Park and R. Sandhu. The UCON ABC usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004.
3. David Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. In *LNCS*, volume 5321, pages 3–3, 2008.
4. Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.
5. M. Harvan and A. Pretschner. State-based Usage Control Enforcement with Data Flow Tracking using System Call Interposition. In *Proc. NSS '09*, pages 373–380.
6. A. Pretschner, M. Buechler, M. Harvan, C. Schaefer, and T. Walter. Usage control enforcement with data flow tracking for x11. In *Proc. STM '09*, pages 124–137.
7. Multimedia framework (MPEG-21) – Part 5: Rights Expression Language, 2004. ISO/IEC standard 21000-5:2004.
8. Kevin Twidle, Emil Lupu, Naranker Dulay, and Morris Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *Proc. POLICY'08*.
9. Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. In *SOSP '05*, pages 17–30.
10. M. Dam, B. Jacobs, A. Lundblad, and F. Piessens. Security monitor inlining for multithreaded java. In *Proc. ECOOP*, pages pp. 546–569, 2009.
11. B. Yee, D. Sehr, G. Dardyk, J. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proc IEEE Symposium on Security and Privacy*, pages 79–93, 2009.
12. P. Kumari, A. Pretschner, J. Peschla, and J.M. Kuhn. Distributed usage control for web applications: a social network implementation. In *Proc. CODASPY '11*.
13. Adobe LiveCycle Rights Management ES. `http://www.adobe.com/products/livecycle/rightsmanagement/indepth.html`, August 2010.
14. E. Rissanen. Extensible access control markup language v3.0, 2010.
15. Gabriela Gheorghe, Paolo Mori, Bruno Crispo, and Fabio Martinelli. Enforcing UCON policies on the enterprise service bus. In *Proc. OTM'10*, pages 876–893.
16. J.A. Goguen and J. Meseguer. Security policies and security models. In *Proc. of IEEE Symposium on Security and Privacy*, pages 11–20, 1982.
17. Bruno P. S. Rocha, Mauro Conti, Sandro Etalle, and Bruno Crispo. Hybrid static-runtime information flow and declassification enforcement. *IEEE Transactions on Information Forensics and Security*, 8(8):1294–1305, 2013.
18. Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In *Proc. PLDI '08*, pages 193–205.
19. Enrico Lovat, Johan Oudinet, and Alexander Pretschner. On quantitative dynamic data flow tracking. In *CODASPY 2014 (to appear)*.
20. A. Pretschner, E. Lovat, and M. Büchler. Representation-independent data usage control. In *Proc. SETOP/DPM*, pages 122–140, 2011.
21. Enrico Lovat and Alexander Pretschner. Data-centric multi-layer usage control enforcement: a social network example. In *Proc. SACMAT '11*, pages 151–152.