Data-centric multi-layer usage control enforcement: A social network example

Enrico Lovat
Karlsruhe Institute of Technology
Karlsruhe, Germany
enrico.lovat@kit.edu

Alexander Pretschner
Karlsruhe Institute of Technology
Karlsruhe, Germany
alexander.pretschner@kit.edu

ABSTRACT

Usage control is concerned with how data is used after access to it has been granted. Data may exist in multiple representations which potentially reside at different layers of abstraction, including operating system, window manager, application level, DBMS, etc. Consequently, enforcement mechanisms need to be implemented at different layers, in order to monitor and control data at and across all of them.

We present an architecture for usage control enforcement mechanisms that cater to the data dimension, grasping the distinction between data (e.g a picture or a song) and its representations within the system (e.g a file, a window, a network packet, etc.). We then show three exemplary instantiations at the level of operating system, application, and windowing system. Our mechanisms enforce data-related policies simultaneously at the respective levels, offering a concrete multi-layer enforcement and laying the grounds for a combined inter-layer usage control enforcement.

In this demo, we consider a use case from a social network scenario. A user can, on the grounds of assigned trust values, protect his data from being misused after having been downloaded by other users. In particular, our mechanisms prevent sensitive data in the browser window from being printed, saved or copied to the system clipboard, avoid direct access to the cached copy of the file and forbid taking a screenshot of the window where data is shown.

1. INTRODUCTION

Usage control is an extension of access control concerning actions that can and have to be performed over data after access has been granted. Usage control requirements, such as "delete after 30 days" or "notify owner upon access", are expressed in so-called policies. These policies can and should be enforced [4] at different levels of abstraction (LoA) in the system. Among others, this has been done at the levels of operating system [1], X11 [3], Java, enterprise service bus, social network [2] or in the context of DRM.

The reason for this variety of enforcement mechanisms is that the data that has to be protected comes in different representations: as network packets, as attributes in an object, as window content, etc. In principle, all of them eventually boil down to some representation in memory, but it turns out to be more convenient to protect them at higher LoA. For instance, taking screenshots is easily inhibited at the X11 level; disabling the print command is easily done at the browser level; prohibiting dissemination via a network requires little effort at the operating system level; etc.

In this work we discuss a generic architecture for usage control mechanisms that can be instantiated to arbitrary LoA. We present three exemplary instantiations at the level of operating system, browser application and windowing system and describe a use case in a social network scenario.

SACMAT'11, June 15–17, 2011, Innsbruck, Austria.
ACM 978-1-4503-0688-1/11/06.

2. ARCHITECTURE

Our architecture is built on top of three main blocks: a *Policy Enforcement Point* (PEP), which observes and possibly modifies events in the system; a *Policy Decision Point* (PDP), where usage control decision are finally taken; and a *Policy Information Point* (PIP), which provides additional information to the PDP.

The role of the PEP is to intercept desired and actual events, forward them to the PDP and, according to the response, allow, inhibit or modify them. On the grounds of such events, the PDP evaluates the policies using standard runtime verification techniques. The PDP component represents the usage control logic of the architecture and due to its generic behavior, its implementation can be the same for several LoA (provided that the semantic binding (events in the system - events in the policies) is given).

PDP decisions sometimes may require additional information about distribution of data among different representations. For this reason the PDP queries the PIP. The PIP is a data-flow tracking system, that simply groups all the representations of the same data. Every time an event happens, the PIP updates its mapping according to a predefined semantic. For instance, if event "copy file1 to file2" is detected, then the PIP records the information that file2 is another representation of the same data stored in file1.

The three components are designed to communicate using a network protocol, so they can be deployed in a distributed way. However, due to their level-dependent nature, PEPs are bound to specific LoA. The inclusion of other components, e.g. for policy management, is work in progress. In particular, so far we can only have independent instantiations of our architecture at different LoA. A future work will be a PIP that "vertically" connects representations of the same data at different LoA (i.e. a synergy of PIPs at different levels). For the purpose of this demo, these aspects have been hard-coded (see Section 4).

3. IMPLEMENTATION

We instantiate our generic architecture at three different LoA: operating system, application level and windowing sys-

tem. Note that identical implementations of PDP and PIP, due to their layer-independent nature, have been used in all the examples and are not presented here for space reasons.

At the application level, we implemented a usage control mechanism for the Firefox web browser [2]. In this context, events are user commands, like "save the page", "copy & paste", "print", etc., and data is represented by the content of the web page. We implemented a Firefox extension that replaces the original mapping between user interface and core functionalities with a javascript instantiation of our PEP. Each user command is hence intercepted and, according to the PDP response, allowed, modified or inhibited (and eventually additional actions executed).

At the operating system level, relevant events are system calls, performed over data representations such as files, network packets, memory regions, etc. Hence, we implemented our second PEP leveraging *Systrace*, a tool for system call interposition in Unix-like operating systems [1].

A third instance of our architecture was deployed for the X Window System (X11, [3]), a distributed system and a protocol for GUI environments on Unix-like systems. In X11, events are network packets that contain requests, replies, events and errors, and are invoked on specific X11 resources (potentially carriers of sensitive information), like windows, pixmaps (memory areas for drawing functions), attributes and properties (variables attached to windows), etc. For this implementation, we developed a wrapper for the X server based on *Xmon*, an interactive X protocol monitoring tool.

4. USE CASE

We consider a use case from a social network scenario where the usage of profile data is restricted according to the relationship between the requester and the owner. If a user logs in and visits another user's page, the server delivers the profile data together with a policy defined on the grounds of their relationship ("best friend" vs "friend" vs "acquaintance", etc.). Such a policy is received and enforced by the visitor's system. Integrity measurements and guarantees that mechanisms are in place and have not been tampered cannot be discussed here because of space restrictions.

In our use case, described in Figure 1, Bob, being Alice's best friend, can see her profile page, print it, save it and, in general, do everything he wants with it (case 1). After a dispute, Alice declassifies Bob from her "best friend" to just an "acquaintance" (case 2(a)). Now Bob still sees Alice's profile page, but he cannot print, save, or copy and paste from it anymore. Moreover he can't take a screenshot of the browser window (case 2(b)) nor directly access a browser cache file (case 2(c)) if these contain (part of) the page.

The policy we enforce is *"Any part of this page cannot be printed nor copied to the clipboard (not even in form of a screenshot) nor saved to disk and its cached version can be used only by Firefox."*. The implementation-level policies in XML are not provided here for space reasons.

If Bob requests Alice's profile page, Firefox downloads it together with the aforementioned policy. Upon reception by the web browser, the web page's content takes new representations at different LoA: it is rendered as a set of pixels in the browser window, it is cached as a file, and it is internally represented by the browser as a node in the DOM tree. To protect all these representations, the Firefox monitor adds some runtime information (like the name of the cache file and the id of the window), deploys tailored policies to the

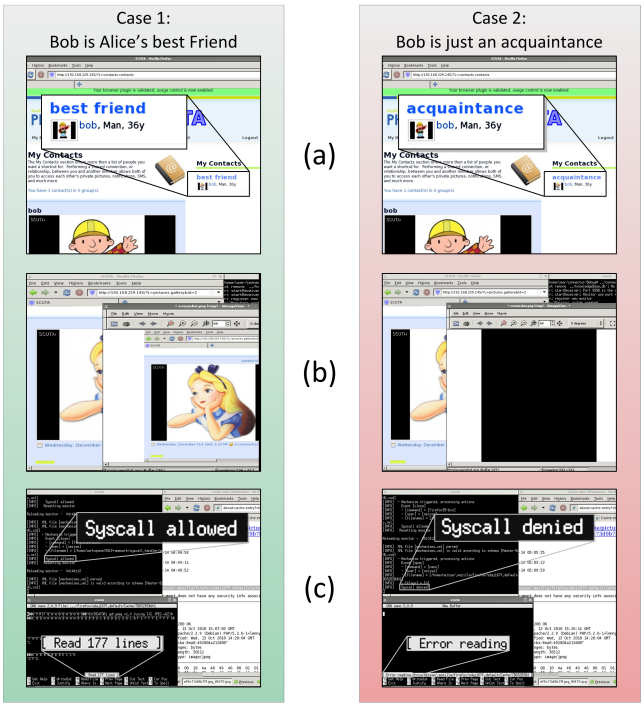

Figure 1: Cross-layer policy enforcement. Row (a) is a view from Alice's account. Rows (b) and (c) are from Bob's perspective, trying to take a screenshot of the page (b) or open a cache file directly (c).

operating system and X11 layers. This is the hard-coded part mentioned above. Afterwards, the Firefox monitor allows rendering the page and creating the cache file.

From now on, several instantiations of the policy are enforced at different LoA, as shown in the following usage attempts. First, Bob tries to print the page; the attempt is intercepted, evaluated against the policy and forbidden. In the second example, taking a screenshot of the browser window is intercepted by the X11 monitor; according to the policy, the request is modified to returning a black rectangle. The last example shows how opening the cache file of a page element (e.g., a profile picture), is prohibited because the the system call is denied unless the caller process is Firefox.

5. REFERENCES

[1] M. Harvan and A. Pretschner. State-based Usage Control Enforcement with Data Flow Tracking using System Call Interposition. In *Proc. 3rd Intl. Conf. on Network and System Security*, pages 373–380, 2009.

[2] P. Kumari, A. Pretschner, J. Peschla, and J.-M. Kuhn. Distributed data usage control for web applications: a social network implementation. In *Proc. 1st ACM Conf. on Data and application security and privacy*, pages 85–96, 2011.

[3] A. Pretschner, M. Buechler, M. Harvan, C. Schaefer, and T. Walter. Usage control enforcement with data flow tracking for x11. In *Proc. 5th Intl. Workshop on Security and Trust Management*, pages 124–137, 2009.

[4] A. Pretschner, M. Hilty, D. Basin, C. Schaefer, and T. Walter. Mechanisms for Usage Control. In *Proc. ACM Symposium on Information, Computer & Communication Security*, pages 240–245, 2008.