Towards a Policy Enforcement Infrastructure for Distributed Usage Control

Florian Kelbert
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
florian.kelbert@kit.edu

Alexander Pretschner
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
alexander.pretschner@kit.edu

ABSTRACT

Distributed usage control is concerned with how data may or may not be used after initial access to it has been granted and is therefore particularly important in distributed system environments. We present an application- and application-protocol-independent infrastructure that allows for the enforcement of usage control policies in a distributed environment. We instantiate the infrastructure for transferring files using FTP and for a scenario where smart meters are connected to a Facebook application.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Information flow controls; D.4.6 [**Security and Protection**]: Access controls

General Terms

Security

Keywords

Distributed Usage Control, Policy Enforcement, Security and Privacy, Sticky Policies

1. INTRODUCTION

Usage control [8] has been proposed and discussed with the goal to overcome one shortcoming of traditional access control models: the loss of control after access to data has been granted. *Distributed* usage control [4] is concerned with the usage of data in distributed system environments. Policies express what may or may not happen to usage-controlled data [3, 5]. They must be enforced at and across all systems storing, processing, and distributing data. For this reason, an infrastructure is needed that allows for (1) inter-system data flow tracking and (2) the enforcement of both globally and locally enforceable usage control policies. Examples for policies are "do not process my data with application X" and "not more than two instances of document Y may be opened simultaneously". While the compliance with the former can be enforced locally, this is not the case for the latter, since the document may be opened on different systems at the same time.

We present an infrastructure that supports (1) application- and protocol-independent data flow tracking across different operating system instances, (2) sticking policies to data upon sending it to another system, and (3) policy enforcement at the receiving site. We implement our infrastructure at the operating system layer and focus on TCP/IP and locally enforceable policies. The infrastructure integrates into an existing usage control infrastructure for independent systems [6]. Security aspects, provided guarantees, and the corresponding assumptions are out of the scope of this work.

We show two instantiations of our infrastructure: the File Transfer Protocol (FTP) and a scenario where a smart meter is connected to a Facebook application in order to share energy usage data with online contacts.

2. INFRASTRUCTURE

The core components of our distributed enforcement infrastructure are a distribution-enhanced Policy Information Point (PIP) and a Policy Management Point (PMP). The infrastructure is distributed in that its components must be deployed on any system, i.e. an operating system instance, that is expected to enforce usage control policies. The task of the PIP is to hold the information flow state of the system on which it is deployed, i.e. information about the distribution of data. At the operating system layer this is essentially the information which data is stored in which files [1, 6]. The PMP manages all usage control policies for data entering, leaving, and residing in the respective system. We equipped both the PIP and the PMP with the capability to communicate with their respective counterparts on other systems, therefore allowing for the exchange of usage control relevant information (namely inter-system data flow tracking and usage control policies) once data flows between systems.

Our infrastructure integrates into an existing usage control solution for single independent systems [6]. The latter consists of a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), and a local Policy Information Point (PIP). We will now show how these two infrastructures integrate.

Initially, the PMP deploys the policy in the system (Fig. 1, step 1). In this work we assume policies to be formulated in terms of system calls as described in [3, 6]. The PEP is tailored to one system layer [6] (in our case the operating system); it intercepts attempted and actual events within

SACMAT'12, June 20–22, 2012, Newark, New Jersey, USA.

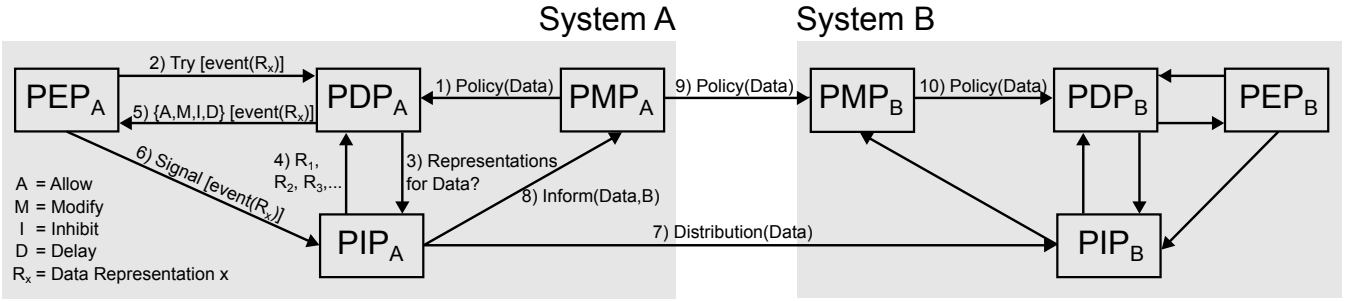

Figure 1: Interplay of the infrastructure's main components.

the layer (at the operating system layer such events have been identified as system calls [1]) and signals them to the PDP (step 2). In order to take a usage control decision, the PDP queries the PIP for additional information about data distribution (step 3). The PIP therefore replies with a list of representations the data has taken (step 4) (e.g., data may be contained in files, pipes, or in some processes' memory). The PDP then decides on the grounds of the policy and data distribution whether to allow, modify, inhibit, or delay the event [5] and sends the decision to the PEP (step 5). The PEP enforces the PDP's decision. If an event actually happened, the PEP signals it to the PIP (step 6) that then updates the system's information flow state.

Likewise any event corresponding to sending usage controlled data to another system over the network is temporarily blocked by the PEP. Our infrastructure then takes care of tracking the inter-system data flow and sticking the corresponding usage control policies to the data.

For example, if PEP_A (X_A denotes component X on system A) intercepts an event of sending data d to system B and if PDP_A decides that releasing data d complies with the policy (otherwise there is no inter-system data flow), then both PIP_A and PMP_A communicate with their respective counterparts on system B: PIP_A communicates to PIP_B that data d is about to be received on a specific network socket (step 7), therefore accomplishing inter-system data flow tracking. PIP_A then informs PMP_A about the data flow to system B (step 8) and PMP_A communicates the usage-control policy associated with data d to PMP_B (step 9), therefore implementing the sticky policy paradigm. PMP_B then deploys the received policy on system B (step 10). After the PIPs and PMPs finished their communication, the original event is unblocked and the actual data transfer proceeds. Once data d is received on system B, its components are already aware of the previous inter-system data flow and the corresponding policy. Consequently they enforce the policy on the received data. Note that the example is analogous if PEP_B intercepts a corresponding event.

3. IMPLEMENTATION

Our implementation leverages an existing PEP for the OpenBSD operating system [1]. This implementation observes and intercepts system calls using the tool *Systrace* [7]. We extended the implementation with TCP/IP-related system calls, the most important ones being *socket*, *accept*, *connect*, and *write* (and all equivalents like *send*). Notably all system calls related to sending data through a TCP connection (*write* and equivalents) are intercepted and handled as described in §2. The remote communication between the PIPs and PMPs has been realized using XML-RPC [9]; though conceptually different, our current implementation bundles the two remote procedure calls for performance reasons.

4. USE CASES

We instantiate our generic infrastructure for FTP in §4.1 and for a smart meter connected to a Facebook application in §4.2. Note that our infrastructure is independent of application-level protocols, applications, and implementations. It may be used with any application building upon TCP/IP. Videos of our use cases are provided online[1].

4.1 File Transfer Protocol (FTP)

In this use case we consider users Alice and Bob on two different operating system instances. Alice owns the usage-controlled file "AlicesFriends.txt" within her home folder; the corresponding policy states that "the content of 'AlicesFriends.txt' may not be opened using the 'mousepad' text editor". Technically, this policy is specified as event-condition-action (ECA) rule [3]:

```
1  <controlMechanism>
2   <id>DenyMousepad</id>
3   <triggerEvent>
4    <id>open</id>
5    <param name="filename" value="/home/alice/
         AlicesFriends.txt" type="dataUsage"/>
6   </triggerEvent>
7   <condition>
8    <XPathEval>
9     /triggerEvent/param[@name='command']/@value='
          mousepad'
10   </XPathEval>
11  </condition>
12  <actions>
13   </inhibit>
14  </actions>
15 </controlMechanism>
```

Therefore, if Alice tries to open the file "AlicesFriends.txt" (or any local copy of it, since the infrastructure for single independent systems tracks local data flows) using mousepad, the corresponding *open* system call is denied and mousepad fails to open the file (cf. Fig. 2).

Then, Alice decides to share this file with Bob and starts an FTP server (we used a standard vsftpd implementation) that is configured to make Alice's home folder readable. Now, Bob runs an FTP client (gftp standard implementation) and logs in to Alice's FTP server. Therefore, Bob is able to list and read the contents of Alice's home folder; he transfers the usage-controlled file "AlicesFriends.txt" to his home folder. Although having read permissions on the

[1] `http://zvi.ipd.kit.edu/english/26_422.php`

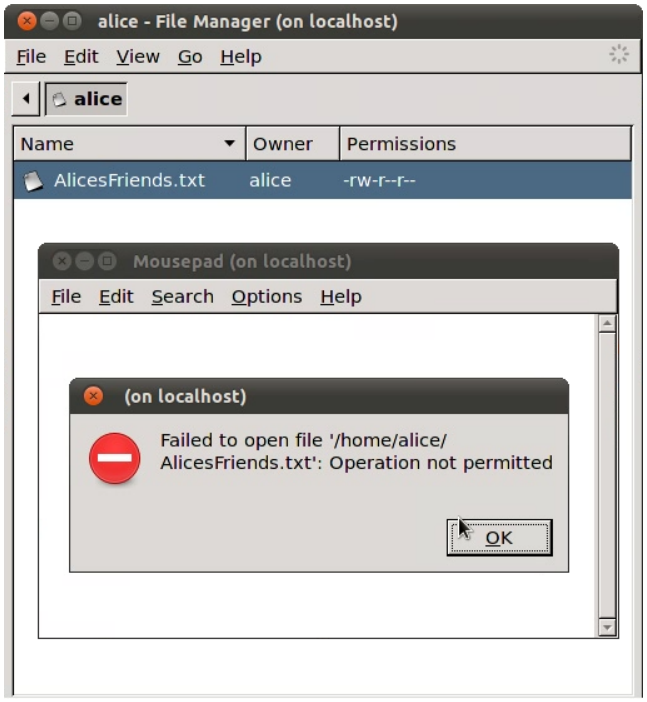

Figure 2: Mousepad fails to open the protected file.

transferred file, Bob is not able to open the file, nor any copy of it, using the mousepad text editor.

This is because the PIPs and PMPs of the two systems took care of (i) tracking the inter-system data flow of the content of file "AlicesFriends.txt" through the TCP channel established by FTP and (ii) transferring and deploying the policy to Bob's system before the actual data transfer happened. Note that we didn't modify the implementation of vsftpd, gftp, or mousepad. Any other equivalent tools would behave identically.

4.2 Smart Meter connected to Facebook

Our second use case is an instantiation of our infrastructure to smart meters connected to Facebook [2]. In this scenario, both Alice and Bob have (simulated) smart meters installed in their homes and opted in to a Facebook application that allows for the sharing and comparison of the energy consumption measured by the smart meters.

The smart meters send their readings to a trusted third party (meter reader) that accumulates the data for further services like billing. The meter reader also releases the energy usage data to other services, like the Facebook application, upon receiving appropriate credentials (that must be provided by the corresponding smart meter user). Since both Alice and Bob provided their credentials in the registration phase, the Facebook application is allowed to request their energy usage data and share it for comparison in forms of graphs and avatars. If no usage control policy is specified, data may be used, stored, and shared unrestrictedly once it has been released by the meter reader.

The meter reader gives users the ability to specify usage control policies for their energy usage data. While in our current implementation users have to specify policies as ECA rules, future instantiations of this work may integrate a more user-friendly policy specification tool as described in [3]. Once Alice decides to deploy the policy "Facebook application must delete all data older than 14 days", the policy is sent to the Facebook application along with the next request of energy usage data. The policy is enforced and the graphs and avatars shown are based on the data of the last 14 days only.

Technically, the meter reader and the Facebook application communicate using HTTP. The meter reader runs a (standard Apache) HTTP server that is queried regularly for the readings by a (simple self-written) script run on the Facebook application server. We instantiated our enforcement infrastructure at both of these sites, therefore expanding data flow tracking and policy enforcement of the energy usage data to the Facebook application once Alice deploys a policy. Each further data request after policy deployment is then temporarily blocked until the meta communication introduced by our infrastructure (between the respective PIPs and PMPs) has finished.

Again, our infrastructure is independent from the applications used.

Acknowledgment. This work was funded by a Google Focused Research Award on Cloud Computing.

5. REFERENCES

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

[2] P. Kumari, F. Kelbert, and A. Pretschner. Data Protection in Heterogeneous Distributed Systems: A Smart Meter Example. In *Proc. Workshop on Dependable Software for Critical Infrastructures. GI Lecture Notes in Informatics*, Oct. 2011.

[3] P. Kumari and A. Pretschner. Deriving Implementation-level Policies for Usage Control Enforcement. In *Proc. 2nd ACM Conference on Data and Application Security and Privacy*, pages 83–94, Feb. 2012.

[4] A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *Communications of the ACM*, pages 39–44, Sept. 2006.

[5] A. Pretschner, M. Hilty, D. Basin, C. Schaefer, and T. Walter. Mechanisms for Usage Control. In *Proc. 2008 ACM Symposium on Information, Computer and Communications Security*, pages 240–244, Mar. 2008.

[6] A. Pretschner, E. Lovat, and M. Büchler. Representation-Independent Data Usage Control. In *Data Privacy Management and Autonomous Spontaneus Security*, volume 7122 of *Lecture Notes in Computer Science*, pages 122–140, 2012.

[7] N. Provos. Improving Host Security with System Call Policies. In *Proc. 12th USENIX Security Symposium*, June 2003.

[8] R. Sandhu and J. Park. Usage Control: A Vision for Next Generation Access Control. In *Computer Network Security*, volume 2776 of *Lecture Notes in Computer Science*, pages 17–31. 2003.

[9] D. Winer. XML-RPC, http://xmlrpc.scripting.com/, 1998.