A USER FRIENDLY GUARD WITH MOBILE POST-RELEASE ACCESS CONTROL POLICY

Douglas E. Williams, Amgad Fayad, Sushil Jajodia, Daniel Calle
The MITRE Corporation

7515 Colshire Boulevard, McLean, VA 22102-3481, USA

{dewillia,afayad, jajodia, dcalle}@mitre.org

Abstract: Information security guards perform an important function in multillevel security (MLS) environments. To perform their functions correctly, guards must contain data release and sanitization rules that accurately reflect the reclassification or declassification requirements to move data across information security boundaries. The current guards, however, require considerable technical skill to express release and sanitization rules, which data producers typically do not possess. Another limitation of the current guards is that once the data passes through a guard, all access control requirements to that data is lost. In this paper, we propose a high-level language to express release and sanitization rules, as well as post-release access control rules. We also describe a prototype that demonstrates the applicability of our approach.

1. INTRODUCTION

Information security guards regulate the transfer of data across security boundaries in multilevel security (MLS) environments. In addition to an allow or deny release decision, guards frequently perform operations such as sanitization of data by removing portions that are still considered too sensitive to be released. While some complex data will always require manual review, the ability to have automated review capability has the

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35691-4_52

D. Gritzalis et al. (eds.), *Security and Privacy in the Age of Uncertainty*

potential for being very cost effective when considering the large quantities of classified data that government organisations are reclassifying or releasing from classification over a period of time.

Traditionally, the source of data that passes through the guard is called the high side and the destination of the data the low side [5, 8]. On the high side, security guard operations generally involve two groups. The first group, which we will refer to as *Data Producers,* is responsible for producing the data and establishing associated release and sanitization rules. The second group, which we will refer to as *Guard Administrators,* is responsible for interpreting the Data Producers' rules to generate accredited executable release and sanitization rules on the guard. This relationship is a weakness of the current system. It requires accurate, rapid, effective, and consistent communication between the Producers and Administrators. If the Data Producers miscommunicate with the Guard Administrators about the correct policy rules to apply to a data object, the guard will not function correctly. This is further exacerbated in the current trend in dynamic multinational coalitions, where communication is even more difficult. It is therefore a necessity that guard rules must be communicated quickly and accurately between the Producers and Administrators.

Another challenge for writing software security guards is their certification for a formal evaluation of trust. Since guards become the responsible parties for information declassification, they must operate with a very high degree of reliability. To achieve this, the software code must be meticulously inspected for correctness and, if confirmed, certified correct. Therefore, the simpler the guard source code is, the more confident one can be to its reliability to perform correctly.

Finally, in addition to release control rules, Data Producers may wish to request that certain access control rules be enforced after the data is released beyond the guard to the low side. We refer to this capability as *post-release access control*, and could consist of both access provisions and access obligations [1, 6]. Unfortunately, existing guard mechanisms do not provide such a capability, and once a document is released, all control over that document is lost. Providing such control could enhance the security and flexibility of an information system.

Doshi et al. [4] propose a *mobile policy* framework that allows policies to move through the distributed system, accompanying the data it is intended to protect. In this framework, policy administration is separated from policy enforcement: policy is specified and administered at only those elements of

the distributed system authorized to do so (frequently the "owner" of the data), whereas any trustworthy component in the distributed computing environment can enforce the policy.

In this paper we build on the idea presented by Doshi, et al. [4] and Felt, a Guard language currently in use [5, 8], and propose a high-level tool that acts as a front-end to legacy guard systems. We will also explore the use of mobile policy to implement post-release access control.

This paper is organized as follows. Section 2 describes current approaches to guards and their limitations. Section 3 outlines the architecture of our proposed solution. Section 4 describes our proposed guard language. Section 5 provides examples of guard rules using our language. Section 6 describes our prototype implementation. Finally, section 7 outlines conclusions and future directions for this work.

2. CURRENT APPROACHES AND THEIR LIMITATIONS

In this section, we give an overview of current guard systems. As outlined in the Director of Central Intelligence Directive (DCID) 6/3 [3], a guard is defined as a process (or set of controls) that function as a "controlled interface" mediating transfers across security boundaries. Guards typically control the flow of data from a "high" domain to a "low" domain, where the "high" domain is at a higher level of security classification than the "low" domain. However, this is not always the situation, because Guards may also control the "horizontal" flow of data between two domains of equal security classification levels. Finally, a Guard can also control the flow of data to more than one domain as in a one-to-many relationship. The guard is considered part of a domain's security architecture and it enforces a well-defined security policy.

Many variations of software security guards are in existence today. They provide services at security boundaries that include filtering, sanitization, transliteration, and integrity checks. Guard types range from low-to-high, high-to-low, manual review, and fully automated. As expected, many guards today consist of computer software, and one such guard development environment is *Felt* [5, 8] currently used by two popular guards: the Information Support Server Environment (ISSE) and the Command and Control Guard (C2G) of the U.S. Department of Defense.

Felt is a software development environment that was developed by The MITRE Corporation under funding from the Rome Laboratory of the United States Air Force Electronics Systems Command, and the Defense Information Systems Agency (DISA). Felt partly consists of a computer programming language with a special notation for defining data structures. This language has a special-purpose notation for defining message structures and it provides operations that can be used in writing constraint-checking procedures to determine whether a structured message is releasable, or to sanitize portions of it [5, 8]. The other part of Felt consists of an executable pre-processor that converts Felt code into the C programming language code. The code is then inspected by the guard administrators for correctness, is certified if it is, and then compiled with a regular C compiler.

While Felt provides a powerful mechanism for building guard filters and has proven to be quite successful, it has two disadvantages. First, it requires considerable expertise to use because of its power and flexibility. While Guard Administrators typically have the required expertise to use Felt, guard rules frequently originate from the Data Producers who often do not have the necesssary skills or training to accurately define policy for a document. A good solution to this problem is to develop a high-level language that is easier to use by the Data Producers.

Another limitation that Felt has pertains to release rules. In many situations release rules are sufficient, but at times it is useful to be able to control access to a document after it is released by the guard. For example, the Data Producer may wish to specify Role-Based Access Control (RBAC) rules that limit access to the data within the low side to individuals with particular role or rank credentials. Current guard tools, such as Felt, provide good release control, but do not provide any post-release access control functionality. For this additional flexibility a more comprehensive and extendable guard language is needed.

Before using a guard in a Department of Defense (DoD) environment, the guard and its rules must be officially certified to provide assurance that the guard will function properly and protect sensitive data from leaking to less secure domains. The Felt environment has gone through the rigorous, lengthy, and expensive verification program using formal methods and is consequently known to be correct. To replace an existing tool that works, in spite of its limitations, is an undesirable choice. Therefore, we chose to build on the existing Felt system.

3. OUR APPROACH

In this section, we outline our proposed solution to the limitations of usability and post-release access controls that we identified with current guard tools in Section 2 of this document. Figure 1 shows the data and policy flow between Data Producers and Guard Administrators. In order to improve this communication, we propose a high-level tool to bridge the gap between the Data Producers and the Guard Administrators. Using the tool, Data Producers can use a high-level language, called MoPEd, to specify the release and sanitization rules. See Section 4. The tool then translates these rules into Felt so that the Guard Administrator can produce the accredited executable release and sanitization rules to be applied by the guard. In addition to the release and sanitization rules, Data Producers can also use MoPEd to specify post-release access control rules. These rules are attached to the data in the form of a mobile policy [2, 4, 7].

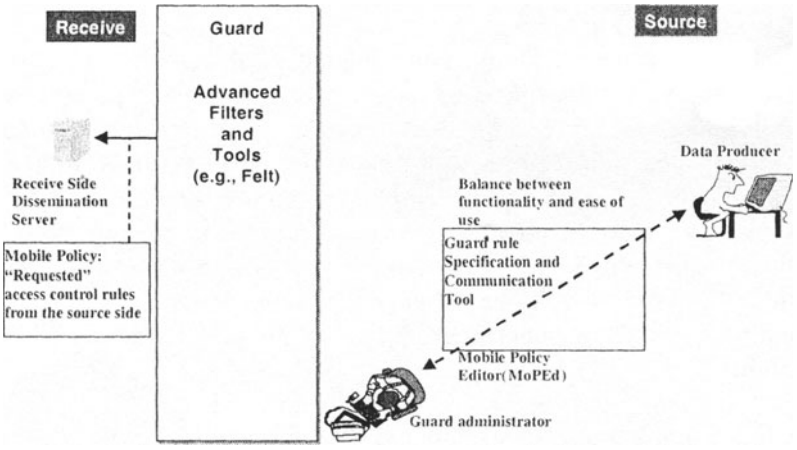

Figure 1. The Guard Policy Tool

The benefits of this approach include the following: First, communications between Data Producers and Guard Administrators about guard rules is easier and faster to use because the policy language more closely approximates human languages than Felt does. A result of this rule simplification should be that guard rule generation is also less prone to human error. Second, guard rule certification can be pipelined. This can be advantageous in two different ways. First, rather than being a wrapper, if a pre-processor approach is used there is no compelling need for the pre-processor to be certified, because all output from the pre-processor must next

be processed by Felt before being put into operation. Second, if one does elect to certify the system using the pre-processor, this is more easily done because the pre-processor is separate and distinct from Felt and consequently can be considered alone. This is especially advantageous in dynamic environments that require quick changes to the software guards, or in environments where cost is an issue.

4. GUARD LANGUAGE

As we have indicated, one of the difficulties in implementing software guards is the problem of constructing reliable and accurate filtering rules for data crossing security boundaries. The ideal language used to describe the filtering rules must be sufficiently powerful to accurately apply security policies, yet also must be user-friendly enough to help, and not hinder, the correct operation of the guard. This is indeed one of the fundamental challenges of information security: making security as transparent to the users as possible.

Earlier we described the Felt language and compiler as a means of generating release filtering code for a guard. While FELT is a powerful and flexible language, it has two shortcomings. First, Felt is a relatively low-level language so it requires well-trained personnel because all details must be explicitly stated. Second, Felt expresses only filtering rules for release control, and does not provide any access control or rule provisions for the data after it is released. To overcome these shortcomings, we first developed a high-level and user-friendly language for generating Felt source code. Next, we extended our language by adding post-release access control features.

It is worth noting that since our language is at a higher level and a lower complexity, it does not provide all the functionality that Felt provides. Our intent is to strike a balance between functionality and ease of use. We do not consider our language is a replacement for Felt, but rather an enhancement to the Felt rule composition process. We believe this is justified, because the majority of guard rules are very similar and we intend to provide constructs in the language to describe these common rules.

The style chosen for this high-level language was similar to that proposed by Doshi, et al. [4], which in turn was similar to the grammar and syntax of the Structured Query Language (SQL) developed by IBM in the 1970s. Our language, named Mobile Policy Editor (MoPEd), was designed to satisfy the

majority of needs for typical guard configuration and can be considered a high-level meta-language for Felt. MoPEd was deliberately kept simple to handle the majority of situations that occur in normal operations, but because it generates Felt code any filtering rule that can not be described using this language can instead still be written in Felt. The syntax of MoPEd is described in a Backus-Naur Form (BNF) style in Figure 2. This language should be considered as under development and may evolve in future releases.

Figure 2. MoPEd Language BNF

```
POLICY      ::= "RELEASE" <OBJECTS> "AT" <CLASS> "APPLY" <FILTER> "WITH" <PRAC>
<OBJECTS>   ::= <obj_regex> <OBJECTS> | <obj_regex>
<CLASS>     ::= "UNCLASSIFIED" | "CONFIDENTIAL" | "SECRET" | "TOPSECRET"
<FILTER>    ::= <SANITIZING> < EXCLUDING> ...
<SANITIZING> ::= "SANITIZE" <san_regex> | <empty>
<EXCLUDING> ::= "EXCLUDE" <exc_regex> | <empty>
<PRAC>      ::= <GRANTING> | <REVOKING> | <empty>
<GRANTING>  ::= "GRANT" <ACCESS> "TO" <SPATTR> "WITH" <PROVISIONS>
<REVOKING>  ::= "REVOKE" <ACCESS> "TO" <SPATTR>
<ACCESS>    ::= <READ> <WRITE>
<READ>      ::= "READ" | <empty>
<WRITE>     ::= "WRITE" | <empty>
<SPATTR>    ::= <USERS> <ROLES> <RANKS> ...
<USERS>     ::= "USER" <user_list> | <empty>
<ROLES>     ::= "ROLE" <role_list> | <empty>
<RANKS>     ::= "RANK" <rank_list> | <empty>
```

A description of the MoPEd language semantics is given below.

- "RELEASE" - A flag denoting the start of a policy statement.
- <OBJECTS> - A list of objects the policy it to affect. It is designated by a regular expression, but in its simplest form could simply be limited by the operating systems file naming constraints.
- "AT" – A flag denoting the start of the security classification the released object is to have.
- <CLASS> - The permitted security classifications
- "APPLY" – A flag denoting filtering to be applied before release of the object.
- <FILTER> - The set of possible release filters. Currently there are only "SANITIZE" and "EXCLUDE", but this list will probably grow in the future. The difference between the two types of filters is that "SANITIZE" means to replace a specific expression with a word such as

"censored", while "EXCLUDE" deletes any paragraph containing a specified expression.

- <san_regex> - The list of expressions to be sanitized from the object. Currently this is interpreted as simply omitting the sanitized part of the object.
- <exc_regex> - The list of expressions to be sanitized from the object. Currently this is interpreted as omitting the part of the object.
- <PRAC> - Post-Release Access Control (PRAC) rules, or those that apply to the object after the object leaves its security domain when a security principal attempts to access it. Currently the two options are "GRANT" and "REVOKE."
- <ACCESS> - The current attributes for the released object. Currently there are only "READ" and "WRITE," but this list will probably grow in the future.
- <SPATTR> - Security Principal ATTRibutes, or people or processes that may interact with the released object. Currently these may be a list of specified users, roles, and ranks.
- "USER" – A flag denoting the start of the list of users affected.
- <user_list> - A user, a delimited list of users, or the empty set of users.
- "ROLE" - A flag denoting the start of the list of roles affected.
- <role_list> - A role or a delimited list of roles, or the empty set of roles. Some examples are "radio operator", "system administrator", or "coalition members only".
- "RANK" - A flag denoting the start of the list of affected ranks.
- <rank_list> - An organisational rank or a delimited list of ranks, or the empty set of ranks. Some examples are "Colonel", "department manager". Whether ranks include those above the specified level, or only those at the specified level, depends on the environment's security policy.
- <prac_regex> - The post-release access control list is a delimited list of environmentally specific restrictions placed on access control outside the object originating boundary. Some examples might be "delete after reading" or "access only after a specified date".

Finally, definitions for release rules, and post-release access control Grant and Revoke rules in MoPEd are given.

Definition 1 (Release Rule): A guard policy release rule is a rule of the following form:

RELEASE <*OBJECT*>
AT <CLASS>

APPLY <*FILTER*>
WITH <*PRAC*>

Definition 2 (Post-Release Access Control Grant Rule): A post-release access control (PRAC) grant rule is a rule that is applied when access to the object is made after its release from the guard. Consequently, the post-release access control grant rule, if one exists, is inserted into the <*PRAC*> field of Definition 1 above. It has the following form:

GRANT <*ACCESS*>
TO <*SPATTR*>
WITH <*PROVISIONS*>

Definition 3 (Post-Release Access Control Revoke Rule): A post-release access control (PRAC) revoke rule is a rule that is applied when access to the object is made after its release from the guard. Consequently, the post-release access control revoke rule, if one exists, is inserted into the <*PRAC*> field of Definition 1 above. It has the following form:

REVOKE <*ACCESS*>
TO <*SPATTR*>
WITH <*PROVISIONS*>

5. LANGUAGE EXAMPLES

In this section some guard policy rules are presented to illustrate the power of the MoPEd system. First, an example demonstrating the economy and user friendliness of the language is given. This is followed by an example demonstrating the power of MoPEd has in integrating release rules and post-release access control rules into a single policy rule.

Consider, for example, that the words "cat" and "dog" must be removed from data and replaced with the word "censored" before the data can be declassified. In Felt, the following lines of program would be required.

```
%{
#include <stdio.h>
#include <string.h>
#include <sys/types.h>      /* Required for regex */
#include <regex.h>
%}

struct main {                       /* This filter */
  fields {                          /* rejects lines */
```

```
    string "\n" line;           /* that contain one */
  }
  action check_line             /* of four words */
};

%{
/* line_checker returns true if its arg matches
   the given regular expression. */
felt_regex_gen(line_checker,
               "cat|dog",
               REG_EXTENDED)
void check_line(ffInstance i)
{
  f_main d = (f_main)i->data;
  if (strstr(d->line->str, "cat")) {
    d->line->str = "censored";
    d->line->len = strlen(d->line->str);
    d->ffattributes.line.string = d->line;
  }
}
void check_line(ffInstance i)
{
  f_main d = (f_main)i->data;
  if (strstr(d->line->str, "dog")) {
    d->line->str = "censored";
    d->line->len = strlen(d->line->str);
    d->ffattributes.line.string = d->line;
  }
}
%}
```

Using the MoPEd language proposed in this paper, the entire previous policy would become:

RELEASE FILE
AT UNCLASSIFIED
APPLY SANITIZE CAT DOG.

Next, a more complex example showing the combination of guard release rules and post-release access control is presented. For given collection of classified data, a Data Producer might create the following guard filtering rule:

RELEASE FILE
AT SECRET
APPLY SANITIZE OVERLORD
WITH GRANT READ
    TO CAPTAINS 82AIRBORNE
    WITH NOPRINT

This guard release policy states that the object file may be released at a secret level, and the codeword "OVERLORD" is sanitized from it. In

addition, the post-release access control rules state that only Captains of the 82nd Airborne Division are permitted to read the file, but they may not print the file. The voluminous Felt code that would be generated from the release control portion of this policy is omitted due to space limitations of this paper.

6. IMPLEMENTATION

As a prototype to process the MoPEd language, we are developing the MoPEd tool. MoPEd functions in the following manner: A Graphical User Interface (GUI) is presented to the Data Producer allowing him or her to specify release and sanitization rules for a data object. The MoPEd tool translator then "translates" the guard release rules into Felt code and in turn the Felt translator converts it into C code, which it attaches along with the post-release access control rules to the data object. The Guard Administrator reviews the attached policy and applies any other relevant global release and sanitization rules. The Guard Administrator then compiles the Felt generated C code into an executable guard, which in turn processes the data object to determine whether it is releasable. If releasable, the data object is released after stripping it of the Felt release rules and leaving attached only the post-release access control rules.

Users have options available for specifying policy using the MoPEd language. For standalone use by a Data Producer (i.e., an environment that does not provide a MoPEd web server), a simple command-line translator is available to transform a policy specified in MoPEd into Felt. This policy can then be sent to the Guard Administrator along with the data itself.

For extended use, the MoPEd Server presents a web-based interface to Data Producers allowing them to specify release and sanitization rules for data items. The server allows users to upload data and to specify additional policy in one of three ways: (1) Upload a text file containing a MoPEd policy specification; (2) Type lines of MoPEd directly into the web interface; or Use the provided GUI to define the policy via "point and click".

For the receiving end, our prototype system uses a Microsoft SharePoint portal to handle the post-release access control requirements. For proof of concept purposes, each receiving realm is represented by a workspace in SharePoint. Each workspace has its own body of Windows users and groups. The post-release access control specifications are implemented with

operating system level access control by using groups corresponding to the various ranks, roles, and other access levels.

7. CONCLUSIONS AND FUTURE WORK

We have proposed a high-level tool that will provide a front-end to low-level guard tools, such as Felt. We defined a language to express release, sanitization, and post-release access control rules. We also developed a prototype to demonstrate the applicability of our approach to Felt-based guard systems. We succeeded in striking a balance between ease of use, and functionality in our language. We plan additional features for our language in order to broaden its power and functionality. Another direction for future work involves the actual binding of the post-release access control policy to the protected object.

REFERENCES

1. Claudio Bettini, Sushil Jajodia, X. Sean Wang, Duminda Wijesekera, "Obligation monitoring in policy management," *Proc. 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, Monterey, CA, June 2002, To appear.
2. S. Chapin, S. Jajodia, and D. Faatz, "Distributed Policies for Data Management Making Policies Mobile," *Proc. 14th IFIP 11.3 Working Conference on Database Security*, Schoorl, Netherlands, August 2000.
3. DCID 6/3, Available at: http://www.fas.org/irp/offdocs/DCID_6-3_20Manual.htm
4. V. Doshi, A. Fayad, S, Jajodia, and R. Maclean, "Using Attribute Certificates and Mobile Policies in Electronic Commerce Applications," *Proc. 16th Annual Computer Security Applications Conf.*, 2000, pages 298-307.
5. Joshua D. Guttman, John D. Ramsdell, and Vipin Swarup, "Felt: A Security Filter Compiler," Revision 2, Technical Report,The MITRE Corporation, 1999.
6. Sushil Jajodia, Michiharu Kudo, V. S. Subrahmanian, ``Provisional authorizations,'' in *E-Commerce Security and Privacy*, Anup Ghosh, ed., Kluwer Academic Publishers, Boston, 2001, pages 133-159.
7. K. Smith, D. Faatz, A. Fayad, and S. Jajodia, "Propagating Modifications to Mobile Policies," *Proc. 17th IFIP 11 international conference on Information Security*, Cairo, Egypt, May 2002, To appear.
8. V. Swarup, "Automatic generation of high assurance security guard filters," *Proc. 17th National Computer Security Conference*, Baltimore, Md., October 1994.