A Specification Language for Information Security Policies

Juan Manuel Garcia Garcia,
Division de Estudios de Posgrado, FIE,
Universidad Michoacana de San Nicolas de Hidalgo,
Morelia, Mexico,
Email: jgarciagarcia@acm.org

March 31, 2009

Abstract

On an information system, a security policy specifies constraints on resources accessed by processes and information flow among them, and also constraints on external access by outsiders. In order to enforce an information security policy, system administrators face two main problems: First, security policy is often stated informally, leading to ambiguity, inconsistency and incompleteness, and in second place, security policy constraints must be translated on several low level specifications such as operating system access control rules, firewall filtering rules, etc. Is a difficult task to verify if those low level specifications actually enforce the security policy. In this paper we present an information security specification based on process calculus which could be translated to low level specifications.

1 Introduction

On an information system, security policy addresses constraints on processes and information flow among them, constraints on access by external systems including programs and access to data by users. Sound security policies architecture protects organization from attacks as well as accidental internal leakage of information, and data mishandling [1]. Security policies have strategical relevance, since its non-compliance not only threatens the integrity of the system, it also costs the organization a significant amount of money due to loss of information or mans-hours spent to fix the problems.[9]

In order to make effective a security policy, it must be enforced through security controls and its compliance must be verifiable. Most approaches for enforcing and verifying security policies are far from been useful due to informal specifications that might lead to ambiguity, inconsistency, and incompleteness.

Security policies are often stated in natural language. In [4] a paradigm is discussed to extract knowledge from natural language texts such as information security standards, information security policies and security control descriptions, so it could be formalized as logical forms contained on an ontology for information security.

Another issue faced by system administrator in order to enforce a security policy is that it needs to be translated to low-level specifications such as firewall rules [6], operating systems access control specifications [12], authentication mechanism specifications, and so on. Complexity of this task make difficult to verify the security policy compliance.

Nevertheless, relevant progress have been made on access control specification, one central aspect of a security policy [3]. Traditionally, access control models includes discretionary access control (DAC) and mandatory access control (MAC). Mandatory access control had been implemented on several operating systems including Linux [10, 7]. One of the access control models most widely adopted is role based access control (RBAC) [5]. A RBAC architecture for achieving secure inter operation of services in an open, distributed environment, named OASIS, is presented in [2]. Related to OASIS, has been developed XACML, one access control language based on XML [8].

2 Specification Language

As we stressed before, in order to enforce a security policy, system administrator faces a complex task in-
volving heterogeneous setup of security controls such as operating systems access controls, firewall filtering rules, authentication mechanisms, cryptographic protocols, etc., etc. Informal specifications of the security policy make this task even more difficult and also it complicates verify if the security policy is being compliant.

The purpose of our work is the design and implementation of a high level language to specify security policies which can be translated in an automated or semi-automated way to low level specifications as those mentioned above. This language must allow specify constraints on resources access by processes and also constraints on processes intercommunication and external access. In this section we present a sketch of such language.

Since our language must be formulated in terms of processes and communication among them, it seems logical take as a base some kind of process calculus, such as π-calculus [11]. Our specification language could be taken as syntactically sugared π-calculus but this subject is beyond the scope of this paper.

The main declaration of our language is which defines a process. As is showed in figure 1, the process proc is declared followed by the (optional) set \{resource1, ..., resourceN\} of local resources required for proc in order to do its function. Each resource is followed by a mode, specifying the kind of access to resources that is required for the process. As can be observed this part closely resembles the access control lists used on the grsecurity linux kernel hardening project [13]. This is not fortuite because this declaration will be translated to access control list for a hardened linux.

Now, two process communicates thru a channel, which is specified like the figure 2 shows, where proc1 and proc2 are the names of the communicating processes, and parameter1, parameter2, ..., parameterN are N possible communication parameters, such as, authentication method, encryption suite used, etc.

```
channel proc1 proc2 {
    parameter1;
    parameter2;
    ...
    parameterN;
}
```

Figure 2: channel declaration

Then, our language main constructs are processes, which are constrained explicitly to some local resources, and process communicating channels, which are explicitly restricted to some communication parameters. Constraints on processes must be translated to underlying operating system access control rules and channel specification to network access and tunneling specifications.

## 3 Translation to low level specifications

In our implementation, one specification written on the language above explained, must be translated to the following low level specifications:

1. Access control list for the grsecurity RBAC system [13].
2. Firewall rules for linux kernel packet filtering IPtables.
3. Tunneling specification for SSH.
4. Integrity check rules for an integrity verifier like Tripwire.

First, from the processes specification is extracted the list of resources required for each process and translated to access control list for the grsecurity system. If no local resources list is given, then is used the learning mode of the gradm tool [13] to obtain the list of local resources accessed by each process. The access mode of each resources is translated to integrity verification rules for the integrity verifier.

From the channel specifications are obtained the tunneling specifications assuming that all communications should be transmitted through a secure channel such as, in our case, a SSH tunnel. From the channel
specification, the required information to set up firewalling rules also can be obtained.

As an example, we show at figure 3 a security specification for a key distribution center (KDC) of Kerberos V.

```
krb5kdc proc {
    allow_conn 192.168.111.0/24;
    auth none;
}
```

Figure 3: KDC declaration

From this simple KDC specification, the access control rules shown on figure 4 are deduced by means of the learning mode of the gradm tool.

```
subject /usr/local/sbin/krb5kdc o {
    /h
    /dev  h
    /dev/urandom r
    /etc  h
    /etc/krb5conf rw
    /etc/ld.so.cache r
    /lib  rx
    /usr  h
    /usr/lib/libcrypto.so.0.9.8 rx
    /usr/local rxw
    /var  h
    /var/log ra
    /var/log/krb5kdc.log rwcd
    -CAP_ALL
    +CAP_NET_BIND_SERVICE
    +CAP_NET_ADMIN
    bind 0.0.0.0/32:dgram udp
    bind 192.168.211.131/32:750 dgram udp
    connect 192.168.211.0/24: dgram udp
}
```

Figure 4: Access control rules for KDC server

From the access control rules showed at figure 4, is trivial to extract the firewall rules that are show at figure 5.

```
iptables -A input -p udp -s 192.168.211.0/124
         -d 192.168.211.132 –dport 750 -j ACCEPT
```

Figure 5: Firewall rules

### 4 Ongoing and Future Work

Now, we are working on two directions:

1. Refining and extending our specification language so it would be expressive enough but without bother with low level details.

2. Implementing translation to low level specifications to obtain practical insights about required elements on our specification language.

Our current implementation often needs of manual tuning to obtain functional low level implementations generated from the high level specification. Also, another issue is the conflicting integrity verification rules generated: some processes access resources in read-only mode and then corresponding integrity verification rules rise false alerts specially when system processes access those resources. We consider that this issue could be solved if another approach such as kernel level auditing is used instead of signatures based integrity verification.

### 5 Conclusions

We present in this paper a specification language prototype for security policy specification, loosely based on process calculus, that could be easily translated to low level system specifications. We present a simple example of how a high level specification in our language can be translated to several low level specifications. So far, our implementation requires some manual tuning of the low level specifications to make it fully functional. Anyway, our approach seems feasible and useful to facilitate sysadmin work.

Further work must be done to reach full process automation and to solve conflicting integrity verification rules generated.

### Acknowledgments

Author wish to thanks to Juan Flores and Pedro Chavez Lugo for insightful discussions.
References


