Making Middleware Secure on Embedded Terminals

Yoshiharu Asakura, Atsushi Honda, Satoshi Hieda, Hiroshi Chishima and Naoki Sato

NEC Corporation
1753, Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211-8666, Japan

Abstract. Recently more embedded terminals have begun to use a general-purpose OS such as Linux. These terminals can perform various functions, such as downloading applications. Since these applications maybe malicious, it is necessary to protect terminals against them and to ensure stability of services provided by the terminals. We have proposed a security enhanced middleware model for embedded terminals based on Linux (SEMMETL). The SEMMETL offers client identification, access control for each application and resource control for each application. The security enhanced X server (SEN XServer) is an example of our proposed SEMMETL. By applying the SEMMETL to middleware, we can enhance the security of embedded terminals and ensure stability.

1 Introduction

Recently more embedded terminals have begun to use a general-purpose OS such as Linux [1] . These terminals have functions that include downloading content data and applications. Since the content or applications downloaded may be malicious, they can have an adverse affect on terminals. Therefore, it is necessary to protect embedded terminals against such problems and to ensure the stability of services they provide.

Access control and resource control are functions that ensure the stability of services provided by terminals. The former is a function to restrict the kinds of resources that applications can access. The latter is a function to restrict the amount of resources that applications can consume. Server-type middleware such as the X Server[1] accepts connections from client applications and provides services for them. Therefore, middleware needs to identify each client application.

Access control – To implement access control, it is a good idea to introduce a secure OS such as SELinux[2]. Although a secure OS can control accesses to resources provided by the OS itself, it cannot control accesses to resources provided by middleware. Therefore, middleware needs access control for its resources.

Resource control – Many embedded terminals have only limited resources. Server-type middleware allocates resources for client applications to provide services for them. Hence, if one client application consumes a large amount of

[1] All trademarks and registered trademarks referenced herein are the property of their respective owners.

resources, other client applications cannot be allocated sufficient resources[3]. Therefore, middleware needs resource control to restrict resource consumption for each client application.

In this paper, we focus on making middleware for embedded terminals based on Linux secure. This is because Linux has become popular in embedded terminals recently. In Sect. 2, we propose a model that enables middleware for embedded terminals based on Linux to enhance security. In this paper, we call this model *Security Enhanced Middleware Model for Embedded Terminals based on Linux* (SEMMETL). In Sect. 3, we give an example of the SEMMETL being applied to the X Server.

2 SEMMETL

The SEMMETL ensures the services provided by an embedded terminal are stable. The SEMMETL is a middleware model that satisfies the following three requirements:

R1: client identification The SEMMETL must identify each client application.

R2: access control The SEMMETL must check whether a client application has permission to access resources.

R3: resource control The SEMMETL must restrict resource consumption for each client application and each kind of resource.

Figure 1 shows middleware to which the SEMMETL is applied. The middleware accepts connections from client applications and identifies client applications to control access and resources. When the middleware receives and processes requests from client applications, the middleware controls accesses to its resources and restricts resource consumption for each client application.

2.1 R1: client identification

There are several kinds of connections, such as the Internet domain connections and the UNIX domain connections between the SEMMETL and client applications. In the case of a UNIX domain connection, the SEMMETL can obtain a process ID of a connected client application via a UNIX domain socket[4]. However, the UNIX domain connection can only be utilized for internal communication within a terminal. If the SEMMETL needs to accept connections from outside a terminal, a new reliable communication method to obtain client identifiers is required. For example, labeled IPSec is a candidate for this purpose[5]. This functionality enables terminals to control communication with applications on other terminals based on the security label defined in SELinux. This security label can be used as a client identifier.

After obtaining a client identifier, the SEMMETL performs the following two activities:

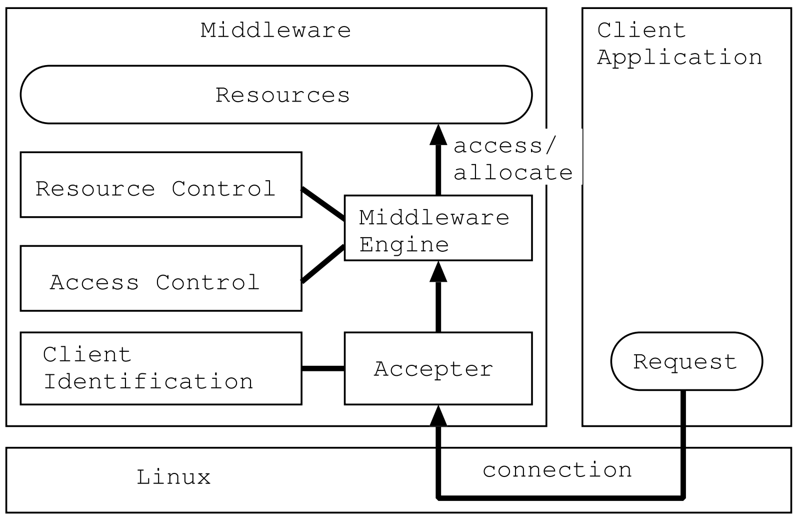

Fig. 1. Middleware to which the SEMMETL is applied

- **A1:** deciding whether the SEMMETL accepts a connection according to a given criteria, e.g. connection types.
- **A2:** classifying the accepted client application into domains according to the client identifier.

To process **A1** and **A2**, the SEMMETL must define rules that specify relations between client identifiers and domains in advance. We call these rules *a domain policy.*

2.2 R2: access control

The SEMMETL provides services to client applications and allows access to none or more of its resources depending on the service required. Therefore, the SEMMETL can implement access control at different levels. The SEMMETL implements **AC1** or **AC2** or both of them.

- **AC1:** restricting use of services provided by the SEMMETL
- **AC2:** restricting access to resources provided by the SEMMETL

In **AC1**, the SEMMETL controls whether client applications have permission to use services. In **AC2**, the SEMMETL controls whether client applications have permission to access resources. The SEMMETL must define rules that specify services or resources that client applications have permission to use or access. We call the rules *an access control policy.* To reduce the size of an access control policy, we specify an access control policy in a domain unit. Permission for a client application is given by the domain to which the client application belongs.

The flow of access control is as follows:

1. The SEMMETL receives a request from a connection and identifies a domain by the connection.
2. The SEMMETL checks whether the domain has permission to use the services or access the resources according to its access control policy.
3. If the SEMMETL concludes that the domain has permission, the SEMMETL processes the request. If not, the SEMMETL does not entirely process the request, that is, the SEMMETL processes only the permitted part of the request or discards the request.

2.3 R3: resource control

The SEMMETL allocates resources according to a request that a client application has sent. To implement resource control, the SEMMETL must identify which client applications possess the resources. We define the following three resource types from the point of view of which possess the resources:

RT1: the resources possessed by one client application
RT2: the resources possessed by several client applications
RT3: the resources possessed by middleware

The SEMMETL must classify allocated resources into the above three resource types, which are described in more detail below:

RT1 Some kinds of resources are allocated to one client application, that is, the resources are accessed by the client application. In this resource type, we consider that the client application that has sent the request possesses allocated resources. The SEMMETL sums the amount of allocated resources in respective client applications and respective kinds of resources.

RT2 Some kinds of resources are shared and accessed by several client applications. In this resource type, we consider that the domain to which those client applications belong possesses allocated resources or middleware possesses them as a shared resources whichever client application has sent the request. The SEMMETL sums the amount of allocated resources in respective domains and respective kinds of resources or as shared resources.

RT3 The SEMMETL allocates resources to itself to work and manage client applications and no client application can access this resource type. In this resource type, we consider that middleware possesses the allocated resources.

For **RT1** and **RT2**, the SEMMETL must define rules that specify maximum amount of resources in each domain and at each kind of resource. We call the rules *a resource control policy.* The maximum amount of resources a client application has is that of the domain to which the client application belongs.

3 Security Enhanced X Server

The X Server is middleware that provides GUI services to X clients. When an X client sends an X request to the X Server, the X Server allocates and accesses X resources, such as windows and pixmaps, while the X Server processes the X request. However, as the X Server has no access control and resource control, an X client can access and consume X resources unrestrictedly. Hence, a malicious X client can obstruct services provided by an embedded terminal by accessing X resources iniquitously or consuming a large amount of a resource. In order to protect against these attacks, we use the SEMMETL on the X Server. We call this X Server *the Security Enhanced X Server* (SEN XServer).

We define one concept as "an owner of an X resource" here. We can classify X resources into a non-shared X resource, which is classified as **RT1**, and a shared X resource, which is classified as **RT2**. An example of the former is a window that is allocated to one X client. An example of the latter is a font that is allocated to all X clients. Therefore, we define the owner of a non-shared X resource as the X client that has sent the X request and the owner of a shared X resource as the SEN XServer.

The SEN XServer satisfies the three requirements described in Sect. 2. The policy file is composed of a domain policy, an access control policy and a resource control policy.

3.1 Client identification in the SEN XServer

The SEN XServer only accepts UNIX domain connections that satisfy **R1**. Since few embedded terminals are required to accept connections outside of terminals, we consider that this restriction is acceptable.

The SEN XServer identifies an X client as follows. The SEN XServer obtains a process ID of an X client via a UNIX domain socket first, and then obtains a full pathname of the X client via a /proc/[process ID]/exe link file. After obtaining the full pathname, the SEN XServer processes **A1** and **A2** as follows. The SEN XServer only accepts the UNIX domain connections as **A1** and classifies the X client into a domain as **A2**. In order to classify the X client into a domain, we specify relations between full pathnames of X clients and domains in a domain policy. If the full pathname is not classified as any domain, the SEN XServer regards the X client as an invalid client and refuses the connection.

3.2 Access control in the SEN XServer

The X Server defines several kinds of access to each X resource. When the X Server receives an X request from an X client, the X Server accesses none or more X resources to process the X request. To implement access control, the X Server must conclude whether a received X request needs processing, that is, whether the X client that has sent the X request has permission to access necessary X resources. Namely, **AC2** is suitable for the SEN XServer.

We define the kinds of access in respective X resources as a primitive access control unit. We call the primitive access control unit *the operation.* Table 1 shows part of the operation for an X resource. We define operations that an X request needs in respective X requests. That is because the SEN XServer must conclude whether an X client has permission to execute the necessary operations for an X resource to process the X request. Table 2 shows part of the sets of

Table 1. Operations (extract)

operations	explanations
Window:addchild	append a child window to a parent window
Window:destroy	destroy a window
Window:map	display a window on the screen
Drawable:draw	draw on a drawable
Drawable:copy	copy pixels from a drawable
Cursor:assign	relate a cursor to a window

Table 2. Sets of X request and operations for an X resource (extract)

X requests	operations	X resources
CreateWindow	Window:addchild	a parent window
	Drawable:copy	using a drawable
	Cursor:assign	using a cursor
CreatePixmap	nothing	–

the X request and the operations for X resources. We specify operations that domains have permission to execute for an X resource in an access control policy.

The SEN XServer processes an X request only if a domain to which an X client that has sent the X request belongs has permission to execute all operations that the X request needs for an X resource. For example, CreateWindow needs three operations: Window:addchild for the parent window; Drawable:copy for the using pixmap; and Cursor:assign for the using cursor. The SEN XServer checks whether the domain has these three operations for the X resource. If the domain has permission to execute the three operations, the SEN XServer processes CreateWindow.

The SEN XServer processes an X request as follows.

1. Identifying a connection from which the SEN XServer receives the X request.
2. Identifying a domain by the connection.
3. Identifying all X resources.
4. Checking whether the domain has permission to execute all the operations that the X request needs for the X resource.
5. If the domain has permission, the SEN XServer processes the X request. If not, the SEN XServer discards the X request.

3.3 Resource control in the SEN XServer

The SEN XServer manages X resource usage as a resource control. Since the X Server consumes memory when the X Server creates X resources, the SEN XServer manages memory consumption for each X client. For resource control, we specify the maximum amount of memory in each domain in a resource control policy. We classify the memory types into the following three types as described in Sect. 2.3.

Non-shared memory (RT1): the memory that is consumed when the SEN XServer creates non-shared X resources.

Shared memory (RT2): the memory that is consumed when the SEN XServer creates shared X resources.

Working memory (RT3): the memory that is consumed when the SEN XServer works and manages X clients.

For the non-shared memory, when the SEN XServer creates a non-shared X resource, the SEN XServer adds a given amount of the non-shared memory to the memory consumption of the owner of the non-shared X resource. If memory consumption of an X client exceeds the specified maximum amount of memory, the SEN XServer does not allocate memory to the X client.

For the shared memory and the working memory, since the owner of a shared X resource is the SEN XServer and the working memory is also used for the SEN XServer, the SEN XServer adds a given amount of the shared memory and the working memory to the memory consumption of the SEN XServer.

3.4 The policy file in the SEN XServer

The policy file is composed of a domain policy, an access control policy and a resource control policy. Hence, the policy file defines the following three elements.

1. Relations between full pathnames of X clients and domains (a domain policy)
   The policy file defines full pathnames of X clients in each domain. The SEN XServer can identify a domain to which an X client belongs by its full pathname.
2. Operations that X clients have permission to execute (an access control policy)
   The policy file defines operations that domains have permission to execute for an X resource. We call a domain that has permission to execute operations *a source domain.* We also call a domain to which an owner of an X resource belongs *a target domain.* The policy file defines operations for a target domain on each source domain.
3. Maximum amount of memory (a resource control policy)
   The policy file defines the maximum amount of memory in each domain per byte unit.

Figure 2 shows an example of a policy file. In this example, the policy file defines 2 domains, the SYSTEM domain and the DOWNLOAD domain. In this policy, /usr/X11R6/bin/xcalc and /usr/X11R6/bin/xclock belong to the SYSTEM domain and /usr/local/bin/dlbrowser and /usr/local/bin/dlmessenger belong to the DOWNLOAD domain. xcalc and xclock can each consume memory up to 2097152 bytes. In the same way, dlbrowser and dlmessenger can also each consume memory up to 1048576 bytes. xcalc and xclock have permission to execute Window:addchild for X resources possessed by the X clients that belong to

```
[SYSTEM 2097152] /usr/X11R6/bin/xcalc /usr/X11R6/bin/xclock
[DOWNLOAD 1048576] /usr/local/bin/dlbrowser
                   /usr/local/bin/dlmessenger

domain SYSTEM {
    {SYSTEM Window:addchild}
    {SYSTEM Drawable:copy}
    {SYSTEM Cursor:assign}
    {DOWNLOAD Window:addchild}
    {DOWNLOAD Drawable:copy}
    {DOWNLOAD Cursor:assign}
}
domain DOWNLOAD {
    {DOWNLOAD Window:addchild}
    {DOWNLOAD Drawable:copy}
    {DOWNLOAD Cursor:assign}
}
```

Fig. 2. An example of a policy file

the SYSTEM and the DOWNLOAD domains. dlbrowser and dlmessenger have permission to execute Window:addchild only for X resources possessed by the X clients that belong to the DOWNLOAD domain. If the SEN XServer receives CreateWindow that uses a drawable possessed by xclock from dlbrowser, the SEN XServer discards CreateWindow because dlbrowser does not have permission to execute Drawable:copy for the drawable possessed by xclock.

4 Conclusion

In this paper, we have discussed a proposed model called the SEMMETL. The SEMMETL enables the enhancing of security of middleware by satisfying three requirements: client identification, access control for each application, and resource control for each application. Moreover, we have given the SEN XServer as an example of the SEMMETL. By applying the SEMMETL to middleware, we can enhance the security of the middleware.

References

1. X.Org Foundation. http://www.x.org/
2. NSA: Security-Enhanced Linux. http://www.nsa.gov/selinux
3. Satoshi Hieda et al., "Resource Management in Linux for Mobile Terminals", IPSJ Transactions on Computing System, Vol.46 No.SIG3, IPS Japan, pp.1-10, January 2005.
4. David A. Wheeler, "Secure Programming for Linux and Unix HOWTO", http://www.linux.org/docs/ldp/howto/Secure-Programs-HOWTO/index.html

5. Trent R. Jaeger et al., "Leveraging IPSec for Mandatory Access Control of Linux Network Communications", Technical Report RC23642 (W0506-109), IBM, June 2005.
6. Doug Kilpatrick, Wayne Salamon and Chris Vance: Securing The X Window System With SELinux. http://www.nsa.gov/selinux/papers/X11_Study.pdf

This article was processed using the LaTeX macro package with LLNCS style