The Pennsylvania State University
The Graduate School

SECURITY ENFORCEMENT IN MOBILE TELEPHONY SERVER

A Thesis in
Computer Science and Engineering
by
Mohamed N Hassan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2008

The thesis of Mohamed N Hassan was reviewed and approved* by the following:

Trent Jaeger
Associate Professor of Computer Science and Engineering
Thesis Advisor

Thomas LaPorta
Professor of Computer Science and Engineering

Mahmut Kandemir
Director of Graduate Affairs and Associate Professor of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

The increasing market share of Linux based mobile phones has motivated developers to make use of the Linux flexible environment to write applications that provide services for mobile users. These services vary from short messaging, calendar, phonebook management, internet browser, mobile games and email clients. These applications require cellular connectivity which is provided via the telephony server API. The telephony server, a part of the phone software architecture, converts these API calls to attention commands (AT) that are sent to the modem. AT commands pose a threat on user data, SIM data and the core cellular network. The current Linux security modules like SELinux enforce access control at the granularity of applications access to the telephony server. Therefore, we intend to design a reference monitor module inside the telephony server that enforces policy over AT+ commands sent to the phone modem. We introduce user level LSM that can decide which classes of application can access which telephony services.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deep and sincere gratitude to my advisor, Dr. Trent Jaeger, who provided me with support and guidance during my research and study at Penn State.

I would also like to express my appreciation to my committee member, Dr. Thomas La Porta, for his helpful and constructive suggestions.

I am thankful to my colleagues Anuj Sawani, Divya Muthukumaran and Sandra Rueda with whom I have had the privilege to work. They have collaborated and provided thoughtful ideas during the course of my research.

I would like to thank Penn State Information Technology Services for providing me graduate assistantship during my two years of study at Penn State.

Finally I would like to thank my parents and family for their support and encouragement during my studies at the university.

Chapter 1

Introduction

1.1 Smart Phones - An Emerging Technology

Improving the computational power of portable devices has become a driving force in the mobile phones market. Cellular providers have extended their range of services beyond basic telephony services providing more services such as internet and media based services. In addition to basic internet browsing, users are now able to access their email, corporate calendars and bank accounts from their phones.

In order to cope with such requirements, mobile phone producers have increased the computational power of smart phones to have nearly the same capabilities as desktop computer. This development was also reflected in both the systems and application domains leading to more complex operating systems and applications. In the systems domain, different flavors of operating systems are developed to run on smart phones. The most common of these operating systems are Symbian, Windows, Mac OS X and Linux. In the applications domain, there are different classes of applications. The first class is signed applications which are usually shipped with the phones. These applications are provided by the operating system producer or third party developers. The process of application signing ensures that the application will not cause any threat to the system. This class of applications is trusted by the operating system allowing them to access and manage phone resources. The second class is untrusted applications that a user can download and install on the phone. The operating system usually prevents such applications from accessing the system resources. Although this is the model for operating systems such as Symbian and Windows, it is not applicable in Linux based phones.

Linux as an open source operating system is more flexible in the terms of what users can install on their phones. This includes open source and unsigned third party applications. Such applications may pose a threat to both the cellular network and user data. For example, a game application downloaded by the user can work as a key logger that logs dialed numbers and key presses. During the idle time this malicious application sends out this data via SMS or GPRS to untrusted sources over the internet. Another example would be malicious worms that read the user phone book and use this as a hit list to spread. It becomes clear that control is required to limit the resources that applications can utilize.

This security threat will not only affect the end user but it could affect the cellular network core. By design the security of cellular networks is dependent on the security of end devices. Eventually, the end devices are becoming less trusted than before. This increase in computational power and application diversity will be pushing the end devices outside the trust domain of the cellular network. The devices have to demonstrate some kind of security in order to protect the core network from malicious behavior.

In the threat models mentioned above, the main assumption is: *Applications that users download are the untrusted subjects not the actual users.* Our assumption is that the system users are not trying to misuse the network. *Therefore, the objective is to protect users from misbehaving applications.*

The security model highly depends on how application interacts with the phone and network resources. The Global System for Mobile communication (GSM) network is our target network. GSM phones are equipped with modems that run both the GSM and GPRS (General Packet Radio Service) stacks. The modem provides an abstraction layer for both the operating system and applications. Modems run their own real time operating system and receive voice/data requests through what is known as attention (AT) commands. The modem multiplexes multiple services such as voice services, data services and use/network data management.

On top of the GSM/GPRS modem runs a telephony server that manages requests sent directly to the modem. The telephony server does the functions of transforming application programming interface (API) calls to AT commands. The commands are submitted to the modem and the response is sent back to the application. There are several design flavors for the Telephony Server. The design used in most Linux phones is client/server architecture. The telephony server listens on a Unix domain socket that clients bind to, and issue API calls.

An application can send a request to the telephony server and the server will create

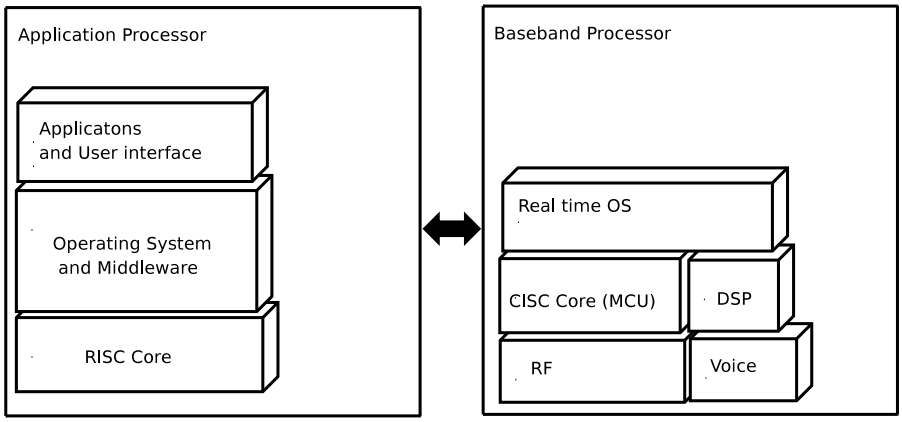

Figure 1.1. Smartphone Hardware Architecture

an AT command and send it to the modem. However, the operating system will not be aware of such AT command because it views the telephony server as a trusted server that issues commands to the modem. Hence, it will not be able to control the flow of AT commands to the Modem.

Security Enhanced Linux (SELinux) is the current Linux Loadable Security Module (LSM) e that provides type enforcement and role based access control. This module is what is provided in version 2.6 of the kernel. However, SELinux will not be able to provide the required access control with the current kernel or user space hooks. For example, if we consider our previous game application as our untrusted subject S and the modem device as the target object O. This modem device provides various operations C. The current security mechanism will say whether S can access O or not, but it will not be able to decide if S can send command C to O or not?

The telephony server interacts with other system entities like the sound subsystem. For example, when a call is created, the analog voice will be forwarded from the modem to the sound chip directly. Untrusted subject can listen to these channels by accessing the sound device files. So, we aim at protecting these subsystems as part of our security goals.

1.2 Thesis Statement and Contribution

To provide such level of access control, we need to intercept the AT command in the telephony server before they are submitted to the modem. In addition, we need to identify objects created by the telephony server's clients and mediate access to these objects. We identify any interaction between the telephony server and other subsystems

and offer a design for protecting these subsystems.

Following is the list of requirements needed to achieve our security goal:

- Understand the phone software and hardware architecture by carrying out a detailed study of the flow of AT commands from applications to the modem and decide the correct hook placement.
- Ensure complete mediation over the services offered by the modem by placing the hooks in the telephony server and protects objects created inside the server.
- Design a user level LSM that can decide which classes of application can perform which operation. The module will use SELinux to check and provide the access decision based on a user defined Mandatory Access Control policy.
- Integrate Security Enhanced Linux in the current Linux Phone distribution to provide policy management and access control decision.
- Design a model to protect system resources, accessed by the telephony server, using the SELinux user level security infrastructure as part of our proposed model.

The thesis statement for this work is therefore: *Design and implement user level references monitor that protect both the phone system and the cellular network against the malicious behavior of untrusted subjects; the reference monitor depends on SELinux for policy management and access decisions.*

1.3 Thesis Structure

Chapter 2 provides a background about the smart phones usage and hardware design, then it gives a brief description of currently available operating systems. In addition to that, the chapter includes background material about reference monitors, the telephony server, SELinux and AT commands.

Chapter 3 covers the related work including network attacks, the threat model and different access control mechanisms that are available in the current phones operating systems. A brief description of sandboxing and different models of server level access control will be introduced in the chapter.

Chapter 4 presents the experiments done to understand the flow of AT commands. Based on this experiment we will show how applications can pose a threat on the phone through the use of privileged AT commands.

Chapter 5 discusses the setup of Openmoko development environment and the porting process of SELinux, reference policy and policy management server.

Chapter 6 covers the hook placement in the telephony server, the user level LSM structure, object manager and sound subsystem security model.

Chapter 7 is the conclusion and future work.

Chapter 2

Background

In the following chapter, we present background material related to the thesis to provide a clear understanding of the phone hardware, operating system, applications and wireless network. Section 2.1 covers the evolution of mobile phones from simple a simple device that provide standard telephony voice function to a handheld personal computer providing a wide range of services. Section 2.2 describes the hardware architecture for different smart phones generations. In the subsequent section, we will survey different smart phones operating systems focusing on Linux based phones.

2.1 Evolution of Smart Phones

Mobile phones evolved from simple devices that provide cellular functions to fully featured personal computer[1]. In addition to basic telephony functions such as calling, phonebook and text messaging, Smart phones run a fully featured operating system with user applications including email client, organizer, PDF viewer and user-installed applications. The consumer demands for such advanced features lead to an increasing market share that grows at a rate of 85% per year. Such growth rate indicates that smart phone will dominate the phone market in the near future.

2.2 The Hardware

The early phones (low tier) used a dual core baseband processor which consisted of a CISC micro-controller (MCU) to manage the RF and a digital signal processor (DSP) to manage voice[2]. The BP runs a real time operating to guarantee high responsiveness for cellular and voice services.

Command	Description
AT+CPAS	Phone activity status
AT+CFUN	Set phone functionality
AT+CPIN	Enter PIN
AT+CBC	Battery charge
AT+CSQ	Signal quality
AT+CPBS	Select phonebook memory storage
AT+CPBR	Read phonebook entries
AT+CPBW	Write phonebook entries

Table 2.1. Example of AT Commands and Their Description

Due to technological convergence another core was required to support smart phones applications requirements. The newly added RISC application processor (AP) has its own memory and flash separate from the BP. In addition to these processors there are a module for other wireless features such as Wi-Fi and Bluetooth. Each module has its own RAM and flash memory.

Some phone companies are planning to move to single core and make use of virtualization technology to run the real time OS on top of the AP.

2.2.1 The Modem and AT Commands

The communication between the AP and the BP is done via attention (AT) commands that are sent through a serial bus. These AT commands provides a various set of functions such as accessing Subscriber Identity Module (SIM) card data, Short Messaging Service (SMS), phonebook management, network registration and calling services. Each vendor supports their own set of AT commands that depends on the modem capabilities and the network type (GSM, WCDMA).

Table 2.1 shows a subset of AT commands that are defined in the GSM specifications 07.07 [3].

2.3 The Operating System

2.3.1 Palm OS

Palm OS is a modular operating system that was designed to support third party development [4]. There is no traditional filesystem supported in Palm OS. It uses memory chunks called records that are dumped into a single database on the device. Applications are single threaded and event-based. Applications can launch another application. Later

versions of Palm OS allow masking of records and encryption.

Palm OS lakes the notion of access control on the database or resources. Data records in the database are accessible by all applications.

2.3.2 Symbian

Symbian has the highest market share (72%) among other operating Systems [5]. It supports preemptive multitasking, multitasking and memory protection.

Symbian uses the concept of Trusted Computing Base (TCB) and Trusted Computing Environment (TCE) [6]. The TCB is the Symbian kernel which has direct access to the hardware and control system processes. The TCE consists of the system server process such as the telephony service. Both TCB and TCE software are provided by Symbian or Symbian trusted providers. Applications that run over the Symbian OS have to be signed in order to have access to the services provided by the TCE and TCB. Unsigned applications are "sandboxed" and cannot access system data or services.

This approach poses limitation on the software developers and users. Developers have to sign the applications they develop and they have to pay for it which in turns hinders open source or cost effective software development.

2.3.3 Microsoft Windows

In Windows mobile applications can run in two modes normal and trusted[7]. A normal process cannot call trusted APIs or modify some register keys. Trusted API includes memory management API, file-based API , SIM API, SMS API, and Extended Telephony Application Program Interface (ExTAP)[8]. Normal mode is designed to prevent error-prone application from accidentally causing damage on the device. However, normal mode cannot prevent malicious code from causing damage to the device. "The primary defense against malicious code is to not run it at all on the device."[7]

In addition to modes of operation, there are three classes of applications Privileged, Unprivileged, and Unsigned. The first two are two classes of application that are signed by a third party. The Privileged applications means that the certificate resides in the privileged certificate store of the device. Same for unprivileged, it exists in the unprivileged store. The third distinction in Windows Mobile Security is One-tier and Two-tier devices. In One-tier devices all applications run in trusted mode. Pocket PC is an example of One-tier devices. In Two-tie devices only privileged applications can run as trusted. A Security Policy defines what application class can run on the device class.

The policy is defined as a key/value pair in *secpol.h*. For example, the key SECPOLICY_UNSIGNEDAPPS represents the "Can unsigned apps run?" policy. The policy value can be yes, no or prompt user.

As mentioned earlier such policy will not prevent malicious applications from accessing the trusted API meaning that malicious code can still access the telephony APIs and call functions that can cause harm to both the user data and wireless network.

2.3.4 Apple Mac OS

The iPhone runs a stripped version of the Mac OS[9]. The security model used in iPhones depends on reducing exposure to the system resources. This policy prevents downloadable applications by MobileSafari to be installed on the phone. It blocks all incoming connections to the phone through TCP/UDP ports. In the first release, Apple doesn't provide an SDK or toolchain to develop and build applications for the iPhone. In the later release, Apple introduced an SDK for third party application developers. In order to include an application in the apple store, developers has to pay Apple Developer Connection. This should provide a security similar to the Symbian OS. However, such security precautions don't prevent attackers from compromising the IPhne through current vulnerabilities in the phone application such as the MobileSafari buffer overflow. Since all process run as trusted any compromised process will have full access to the phone resources.

2.3.5 Java

The Java Virtual Machine (JVM) utilizes a sandboxing approach. The JVM runs unsigned applications in the un-trusted protection domain. Whenever an application requests an access to a resource or a service like HTTP connection it prompts the user for permission. [10]. The JVM is used in both Symbian and Blackberry phones [11]. Blackberry applications that require access to sensitive data or system resources must be signed and registered with Research In Motion (RIM).

2.3.6 Linux

Linux on mobile phones has the second highest share after Symbian with a high growth rate. Several mobile phone producer like Motorola, NEC, Samsung and OpenMoko are adopting Linux Operating System for mobile phones[12]. We have studied two different mobile phones that are running Linux. In the following subsections we will discuss the

architecture and security in each phone.

2.3.6.1 Motorola

The A780 phone uses mainstream kernel 2.4 and a root file system provided by Motorola[13]. The A1200 has upgraded to kernel 2.6 with a modified SELinux access control called MotoAC The kernel is modified to include preparatory device drivers for the GSM modem. On top of the device drivers (/dev/mux*) there is the Telephony Application Programming Interface Server (TAPIsrv) which exports all telephony functions to the upper application layer. In user space there are several applications that use The TAPIsrv such as the dialer, SMS app and Opera browser. In the current architecture, all applications are allowed to access modem mux driver. This is because no access control is employed in the current phones. In order to understand how applications interact with the GSM modem, a cross compiled version of *strace* was used to trace system calls (read/write/ioctl) sent to the modem device.

The traces showed the AT commands that are sent to the modem from the TAPIsrv. As shown below, the network manager on the phone uses two commands AT+CFUN=0 or 1 to turn on or off the GSM stack. The second trace shows the AT commands for applications sending SMS. Another trace shows the GPRS attach in which the phone receives the PDP context from the network. The last trace is for the dialer application.

```
Enabling/Disabling gsm:
write(4,"AT+CFUN=0\r", 10)
write(4,"AT+CFUN=1\r", 10)

Sending SMS:
write(5, "AT+CSCA?\\r", 9)
read(5,"\r\n+CSCA:\"+13123149810\",145", 35)
write(5,"AT+CMGS=17\r",11)
read(5, "\r\n> ", 4)
write(5,"07913121139418F011000A8118840890",51)
read(5,"\r\n+CMGS:2\r\n\r\nOK\r\n", 18)

Establishing GPRS connection:
write(10,"AT+CGDCONT=1,\"IP\",\"wap.cingular\"", 47)
read(10,"\r\nOK\r\n", 6) write(10, "AT+CGQREQ=1,0,0,0,0,0\r", 22)
write(10, "AT+CGQMIN=1,0,0,0,0,0\r", 22)
```

```
read(10,"\r\nG_CONNECT:1,\"10.93.114.134\",\"", 65)

Making phone call:
write(4, "ATD8148800905", 13)
read(4,"\r\nOK\r\n", 6)
```

Listing 2.1. Network Attach Trace

2.3.7 OpenMoko

The second embedded Linux system we studied is OpenMoko. The system runs kernel 2.6 and a gsmd which listens on Linux Domain sockets. Applications use libgsmd which acts like a telephony API to send request to the server. Libgsmd sends the TAPI requests to the gsmd via the domain sockets. The gsmd convert the API calls to AT command that are sent to the modem device.

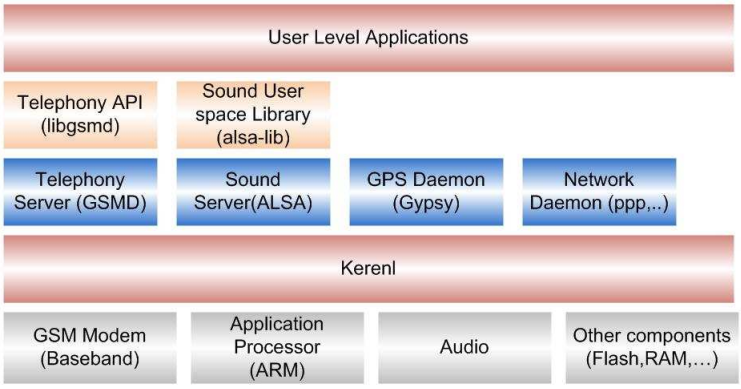

Figure 2.1. Openmoko Softwarex Architecture

2.4 SELinux

SELinux was developed by The National Security Agency and presented in 2001 at the Linux Kernel Summit [14]. Initially, SELinux was developed as a security infrastructure that is integrated in kernel 2.5 through a series of patches. In order to allow for security infrastructure in Linux, Linus Torvalds proposed a set of recommendation to design a Loadable Security Module (LSM) that was later included in mainstream kernel 2.6.

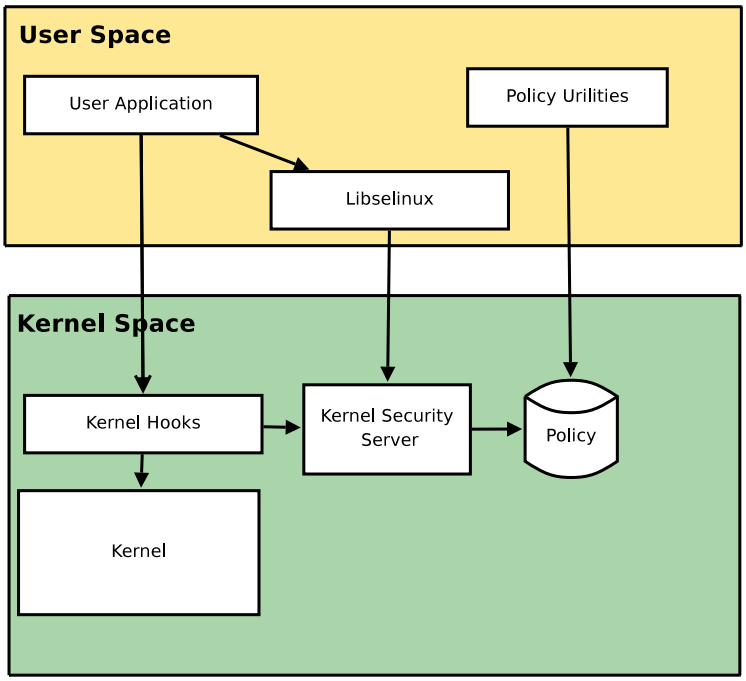

Figure 2.2. SELinux Architecture

2.4.1 SELinux Architecture

The LSM consists of a set of hooks placed inside the kernel to mediate access to kernel objects. In addition, it defines security fields within kernel data structures. Each hook invokes a callback pointer function defined in the *security_ops* structure. The callbacks are dynamically installed during the LSM loading. This allows for loading different implementation of the LSM and chaining different LSMs together.

SELinux as an LSM is based on the Flask architecture[15]. It consists of a security server, object manager and AVC cache. The security server is responsible for making access decisions using the encapsulated security policy. The AVC cache role is storing the security server decisions enhancing SELinux performance. The object managers are implemented inside kernel subsystems such as the networking subsystem and files subsystem. These object managers are responsible for labeling kernel objects with security contexts.

SELinux provides a user space library for userland applications. The library maintains an AVC cache for each application and the AVC cache send decision requests to the kernel security server.

2.4.2 Policy Management and Protection Infrastructure

Tresys is developing an infrastructure for user space application to leverage the access control decision to user space security server and allow for dynamic policy changes[16]. The main goals of the project are:

- Provide fine-grained access control
- support user-space object managers
- Enable user space administration of the policy.

The project is divided into the Policy Management Server (pms) and Userspace Security Server. Both servers communicate policy changes via Inter Process Communication. For more details refer to [16].

We will use the Tresys architecture to propose a design for securing the sound files used by GSMD server and ALSA libraries. The design goal is to dynamically update the policy for accessing these files using the GSMD client security label.

2.5 Threat Model

The variety of services running on the smartphone allows malicious applications, downloaded by the phone users, to exploit weaknesses in the current cellular network architecture and threaten user privacy. Enck et al. showed that with a small volume of SMSes they can deny voice calls to mobile subscribers in a metropolitan area [17]. Bocan and Cretu described a Denial of Service (DOS) attack that can cause local congestion at the base station level [18]. In addition to DOS attacks, malicious apps can send malformed requests or control packets that exploit vulnerabilities in the cellular network nodes like the Home Location Register (HLR) and Mobile Switching Center [19]. Mobile devices can be turned into zombie devices analogous to zombie machines on the internet. These zombie mobile devices can launch timed or coordinated attacks that can take down parts of or the whole cellular network.

Chapter 3

Related Work

In the following chapter we will present an overview of the related work to access control mechanisms and different reference monitor implementations.

3.1 Sandboxing

TRON, a process specific access control, is a capability based access control[20]. It allows users to define capabilities for processes to access files and directories. TRON does not require modification of the program code because it runs the program in a protected domain. The enforcement is carried out by hooks wrapped around existing system calls. When a user runs a process in TRON it will have the same Unix permissions as if it is running outside the domain. These permissions can be extended or limited by the user. TRON introduced the concept of extended permissions such as link and subtree. It introduced the concept of security domains and domain transition.

Internet browser uses helper application in order to process different types of data. For example, acrobat reader is used to view pdf files and media player is used to play media files within the browser. Janus provides a sandboxing mechanism for such helper applications[21]. The motivation is based on a bug in an older version of ghost script helper application that was used by Netscape browser. This version of ghost script allowed malicious programs to execute code that have read/write access to user files. Since application has access to code and data within its userspace, it is required to have the enforcement in a different process (Janus is a user level access control). Any application's system call will be filtered through Janus preventing it from harming the system and isolating it from the reference monitor. The system is divided into a framework and

modules; the framework is the core of Janus and it loads/unloads the security modules. Each module is responsible for filtering a set of system calls. The framework reads a configuration file that defines the module and permissions to be loaded. Janus uses a process tracing facility such as *ptrace* to intercept system calls issued by the untrusted helper applications.

TRON and Janus are example of sandboxing mechanism. Both have drawbacks as they don't provide system wide access control. In Janus the policy is partially hard coded in the module violating the concept of clear separation of policy from enforcement.

Bluebox, a host based intrusion detection and prevention system, is another example of sandboxing in which processes access to kernel resources is controlled via system call interception[22]. Bluebox policy is expressed in human readable format which defines the subject, the resource and the type of access granted. In addition, Bluebox allows for policy inheritance which allows for sharing the same policy between different programs providing an efficient and easy to verify system policy, Systrace is a policy generation tool that allows users to define the policy interactively[23]. The first time an application runs in Systrace, a list of warnings is generated alerting the user about all the system calls the application is trying to execute. The user responses, (allow or deny) to these warning, are used to generate a new policy that will be used for sandboxing the same application when it runs again.

SELinux and AppArmor are another category of sandboxing that use the Linux Security Module (LSM) kernel interface to implement Mandatory Access Control (MAC) restricting applications access to system resources. On the contrary to SELinux, AppArmor uses files path rather than file labels for defining policy. AppArmor developer claims that AppArmor policy is easier to manage than SELinux[24].Rather than that they both provide the same level of protection. Refer to Chapter 2 for more details on LSM and SELinux.

The common feature among the systems, we have studied so far, is mediating access by intercepting and controlling kernel system calls, thus preventing untrusted application from tampering with system resources. Language based sandboxing allow application to run inside a confined environment which is enforced by the language interpreter itself rather than the operating system. Examples of this sandboxing mechanism are java and SafeTcl. More details about sandboxing implementation in Java and SafeTcl are available at [25] and [26] respectively.

3.2 SELinux in Userland Applications

In this section, we will discuss the usage of SELinux in userland application such as X, gconf, and PostgresSQL.

3.2.1 XACE

Xserver is the considered the default graphical user interface engine for Linux and Unix[27]. It is used by various desktops such as GNOME and KDE. The X protocol provides comparably and preference function, but it lacks security requirements.

X is based on client/server architecture in which local clients connect to the server using unix domain sockets and remote clients uses TCP sockets. Clients can communicate with other clients via Xserver shared resources such as the copy/paste clipboard.

X uses client authentication as the only security measure. Once authenticated, clients can perform malicious operations such as tampering with other clients' windows, listening for other clients' keyboard and mouse events, capturing the whole state of the server, sending fake events to other clients and terminating other clients.

X security is divided into two parts: Access to server resources and Control over protocol dispatcher. Server objects or resources such as windows, cursors, fonts and colormaps are labeled by the resource manager by assigning an owner (object creator) label to them. When a client tries to a query any object, a security hook, placed in the lookup function, will mediate access by using both the owner label and client1 (that is trying to access the resource) label as parameters.

The dispatcher uses an array which is indexed with request code, and contains pointer function to serve client request. Hooks are placed at the dispatcher to invoke security access checks providing the request type and request parameters as inputs for the hook function. In addition to the object labeling and dispatcher hooks, there are hooks at the events' sending function to control which events are sent to clients.

3.2.2 Gconf

Gconf is the configuration manger used by GNOME [15]. It stores configuration data for application and send change notification to applications in real time. Same as X, gconf uses a client/server model where clients communicate with the server using the ORBit Object Request Broker.

The threat model resides in the client's ability to read (confidentiality) and modify other clients' configuration, and clients can send fake configuration updates to other

clients (Integrity). From security prospective, the objects of interest in gconf are the configuration pairs (key/value). An object manager is created to label and to maintain these object during creation or modification. Any request, to read or modify configuration data, is being mediated via access control hooks in the gconf server.

3.2.3 SE-PostgresSQL

PostgresSQL is a database management server that utilizes the database access control which is independent of the system [28]. PostgreSQL uses the concept of privileged DB user which grants the user absolute rights for all databases. SE-PostgreSQL will enforce MAC on connected clients eliminating the privileged user concept.

The granularity of objects depends on the DBMS products. For example, some products deal with tables and columns as the smallest object for access control. However, SE-PostgreSQL use rows and columns are the smallest DB object for the purpose of access control.

SE-PostgreSQL implements their own object manager for labeling and managing access to DB objects. In addition, the SE-PostgreSQL has its own AVC cache which connects directly to the kernel security server. Here is an example of SE-PostgreSQL "create table" statment:

```
create table tbl1 (
   id integer primary key
       context = 'system_u:object_r:sepgsql_table_t',
   body text
       context = 'system_u:object_r:sepgsql_secret_table_t'
) context = 'system_u:object_r:sepgsql_table_t:s0:c0';
```

Listing 3.1. SE-PostgreSQL CREATE statement

In the next chapter, we will cover the process of exploiting the telephony sever and analysis of the telephony server that will be used later for designing the reference monitor.

Chapter 4

Analysis of the GSM telephony Daemon

In this chapter, we will show how an untrusted application can make an unauthorized call by issuing AT commands to the modem. In the experiment, we used a Motorola A780 running Linux 2.4 kernel. The aim of the experiment is to show how malicious software can pose a potential threat on both user data and the mobile network.

We extended our study to the Neo1973 open source Linux phones provided by OpenMoko. The OpenMoko phone is capable of running multiple Linux distributions. We will provide an overview of the OpenMoko distribution. We analyze the operation of the telephony server by carrying a static analysis and dynamic analysis. The goal of the analysis is to identify the optimal location to place the access hooks in order to satisfy complete mediation in the telephony daemon.

4.1 Exploiting the Motorola A780

4.1.1 The Phone

The A780 is a quad band gsm phone that was introduced in 2003 [29]. The phone has 3 processors: baseband processor for GSM, application processor for running the Linux OS and a GPS chipset. The phone has a USB serial port that allows users to connect to the phone either in storage mode where they can transfer files to the phone or in modem mode where they can send AT commands to the modem.

The software stack is provided by Motorola. They called the distribution EZX which is a derivative of MontoVista Linux. The main intention of Motorola was to lock down the

phone and allow only application development only using the J2ME Java platform utilizing the sandboxing nature of Java. However, Motorola developers have build OpenEZX, an open source distribution, which runs on the phone without the lock down. In addition, the community provided a loader program called *linloader* which can be used on the Motorola original EZX distribution to unlock the phone [13].

4.1.2 Experiment Setup

We copied *linloader* to the mounted file system and start it from the phone. The loader start networking over USB then it launches the *telenet* daemon which is already installed on the phone (probably for developers to debug Motorola applications). We used the memory card to transfer arm cross compiled malicious code to the phone. The code mimics a trojan program that users can download on the phone.

4.1.3 Experiment

In order to understand how to interact with the modem, we used *strace* on the telephony server while making a real phone call. This trace showed the sequence of files opened and command sent to the device (See background work for an example). Using this trace we wrote the following malicious code:

```
int malicious(){
    int fd=open("/dev/mux8", O_RDWR|O_NONBLOCK);
    if(fd<=0){
        printf("error openning file %i .\n",fd);
    }
    int buffsize,readsize;
    char* rbuffer;
    FD_ZERO(&readfds);
    FD_ZERO(&writefds);
    FD_SET(fd, &readfds);
    FD_SET(fd, &writefds);
    timeout.tv_sec  = 15;
    timeout.tv_usec = 990000;
    if ((nfound = select (FD_SETSIZE, 0,\
         &writefds, 0, &timeout)) == -1){
            perror ("select failed");
```

```
    }
    else if (nfound == 0){
            printf ("select timed out and fd is %i \n",fd);
    }
    else
    {
        printf("got it\n");
        write(fd, "ATD18148800805;\r", 17);
        timeout.tv_sec  = 1500;
        select (FD_SETSIZE, &readfds, 0, 0, &timeout);
        ioctl(fd,FIONREAD,&buffsize);
        rbuffer=(char*)malloc(buffsize);
        read(fd,rbuffer,buffsize);
        printf(" got %i \n",buffsize);
        return 0;
    }
}
```

Listing 4.1. Malicious TAPISrv client

The device driver for the gsm modem in the A780 exports multiple device files to multiplex commands sent to the modem. Each mux receives a specific set of commands. The above code opens the /dev/mux8 which is the device file specific for voicecall commands. Then it issues an ATD command to initiate a voice call. If this command is sent twice it will deny the user from making a call as long as there are 2 active calls.

4.1.4 Results

When the code was executed, it made a successful voicecall to the target party. Repeating that in a loop caused the modem to deny any outgoing or incoming calls. Such code can be embedded inside a malicious game and start issuing voice call commands to premium 900 numbers causing a denial of service on one side and can add charges on the user's bill that may go undetected for a long time.

Further analysis of GPRS operation showed that processes use named pipes (files under /var) to share data with the telephony server. These named pipes contain sent/received data in the clear. Any malicious program can access these pipes and sniff on the connection or inject data in the GPRS stream.

4.2 Analysis of OpenMoko Gsmd

A780 drivers and telephony server are proprietary source. This poses a limitation on both the analysis and code modification. Neo1973 an open source smartphone was a good candidate for the following experiments because the software stack is open source.

The neo1973 has the same application/baseband processor architecture. The software stack is maintained by the OpenMoko team. The stack includes a 2.6 kernel and multiple Linux distribution. The most common distributions are OpenMoko and Qtopia[30].

The OpenMoko distribution contains several applications that were ported from Debian and GNU. The distribution includes Graphical User Interface (GUI), PIM and internet applications that are specific to smartphones.

4.2.1 The GSM Daemon

Gsmd is the equivalent of the telephony server (tapisrv) in the A780. The daemon exports a library called libgsmd which allows application to use the phone services. The *gsmd* listens on a Unix socket for incoming connection from user applications. Once the connection is established, an application can send a request to the *gsmd* that is transformed to an AT command and passed to the modem. Then the response is passed to the caller.

We carried static and dynamic analysis of *gsmd* to observe the flow of commands from user applications to the modem. The result of the analysis is a classification of persistent and non-persistent object (refer to chapter 6) and proposed hook placement.

4.2.1.1 Static Analysis

The analysis involved going through the *gsmd* code to trace different possible paths. In addition, we used cflow to generate a static call graph of different *gsmd* parts.

Figure 4.1 show a sample of usock which represent the callgraph to initialize the *gsmd* user socket. This control flow analysis provides us with some details of the operation

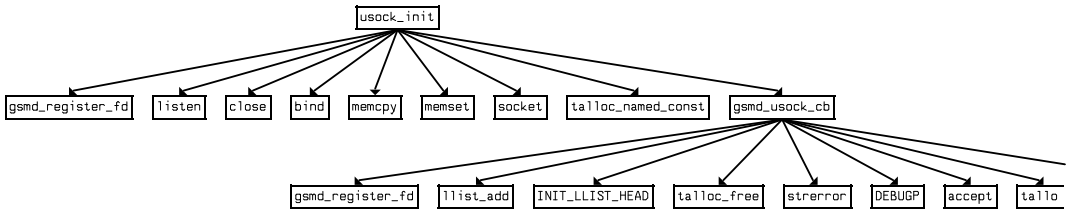

Figure 4.1. Gsmd Static Analysis

of gsmd. The control flow does not provide with sufficient information about *gsmd* execution behavior. Therefore, a dynamic trace was required to study *gsmd* operation with respect to different user inputs.

4.2.1.2 Dynamic Analysis

This analysis required some modification in the build process (discussed in next chapter) of the distribution to include debug information in *gsmd* libraries and binaries.

Using the GNU debugger (*gdb*) we traced the execution path for multiple operations such as initialization, making a phone call, sending SMS and network operations.

Following is a trace of a phone call sent from the dialer application to gsmd:

1. A user request is received on Unix socket causing the main select function to exit.
2. The request is handled by gsmd_select_main in select.c.
3. The callback uf->cb, which maps to gsmd_usock_user_cb, will be invoked.
4. This callback will read a buffer from the fd that contains the request header and data.
5. The buffer is passed to usock_rcv_pcmd in usock.c which will cast the buffer to header and payload and check the headers version. Meanwhile it will determine which payload processing function to call depending on the msg_type in the header. This is performed by calling umh[gph->meg_type].
6. The umh (in this case the type is 003) will result in calling usock_rcv_voicecall in usock.c.
7. An action is taken in a switch case depending on the subtype. Subtypes are dial, hangup, dtmf and answer. In this case the subtype is dial.
8. The dial action will create a command ATD by calling atcmd_fill in atcmd.c which taken ATD as the first parameter and the phone number as the second parameter.
9. A ucmd callback, which handles the AT command response, is added to atcmd command data structure. In this case the response callback is usock_cmd_cb.
10. This cmd is passed to atcmd_submit which will add the atcmd to a list that will be processed later.

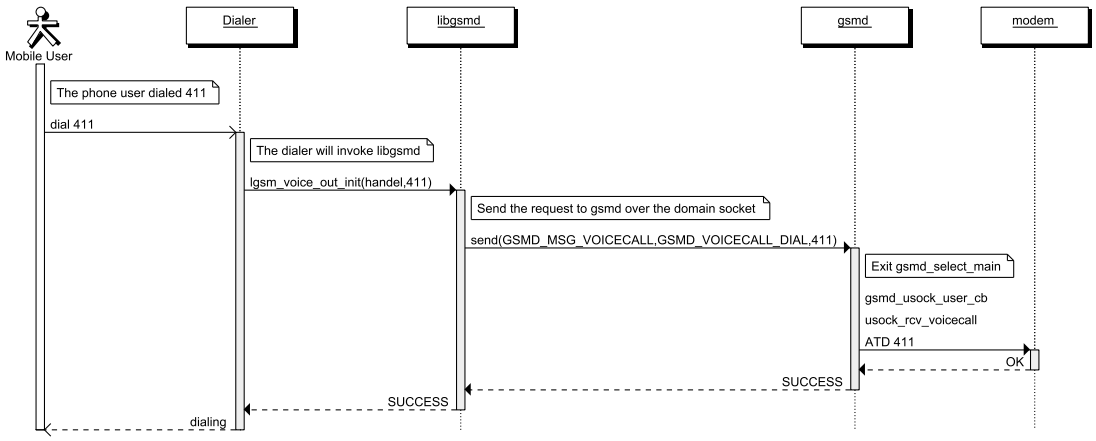

Figure 4.2. Outgoing Voice Call Sequence Diagram

4.3 Results of The Gsmd Analysis

Using the above analysis, we built a different table for each category of operations. Each table describes the AT commands, operation, objects affected and proposed hook placement within the code. In addition to these tables, we will use the analysis to derive the hooks design philosophy discussed later in Chapter 6.

The proposed hook placements took in consideration multiple factors:

- The hook is placed at the point where we have enough information to make the access control decision. This information has to include the subject (user), the object (whether persistent or not) and the operation. This information is passed to the policy server for access decision (Refer to chapter 5).

- The access control information will not change between the point the hook is placed and the point it get executed on the modem ensuring that there is not any security holes after the hooks are placed. Our analysis showed that there is a one-to-one mapping between commands submitted to *gsmd* and AT commands sent to the modem ensuring that the operations are not changing after the security checks.

- All code paths that involve user (application) supplied data or commands must have a security hook as early as possible. There is no action taken that affects the network or user data before the hook.

4.3.1 Security Operations

Following is a list of tables of *gsmd* operations. Each table is preceded by a description of the class of operations in the table, and followed by an example of hook function and a threat example.

4.3.1.1 Voice Calls

Voice calls are requested by the ATD dial command. Once a call is created, a client can halt or hold an existing call. The system supports 2 calls at a time that can be swapped using the call control commands.

AT command	Action	Objects affected	Persistent object	Hook placement
ATD	Dial	Dialing call	Current call	GSMD_VOICECALL_DIAL
ATH	Hang-up a current call	current call	Current call held calls	GSMD_VOICECALL_HANGUP
AT+VTS	Send DTMF in a current call	current call	Current call	GSMD_VOICECALL_DTMF
AT+CLCC	List active calls	incoming call current call	None	GSMD_VOICECALL_GET_STAT
AT+CHLD	Control call hold, release, and multiparty states	Active calls held calls	active call held calls	GSMD_VOICECALL_CTRL

Table 4.1. Voicecall Hooks

```
case GSMD_VOICECALL_ANSWER:
        if(security_voicecall_answer(gu))
        {
                return -EACCES;
        }
        cmd = atcmd_fill("ATA", 4, &usock_ringing_cb,
        gu, gph->id,NULL);
```

Listing 4.2. Example of a voice call hook

Threat: Untrusted applications can use these commands to halt any call created by trusted applications causing a denial of service attack (DOS). In addition, they can create stealthy calls incurring costs toward the user phone bills.

4.3.2 User PIN

The SIM card allows user to provide a user PIN to protect SIM data in case the phone is lost. The client can request setting or changing the PIN.

AT command	Action	Objects affected	Persistent object	Hook placement
AT+CPIN	Query and change the PIN	User PIN	None	`GSMD_PIN_INPUT`
AT+CPIN?	PIN status	None	None	`GSMD_PIN_GET_STATUS`

Table 4.2. User PIN Hooks

```
case GSMD_PIN_GET_STATUS:
        if(security_pin_get_status(gu))
        {
                return -EACCES;
        }
        cmd = atcmd_fill("AT+CPIN?",
        8 + 1, &get_cpin_cb,
        gu, 0, NULL);
        break;
```

Listing 4.3. Example of a PIN hook

Threat: Entering the wrong PIN three consecutive times will lock the SIM card making it unusable. Therefore, these commands can be used to create a DOS attack on the user.

4.3.2.1 Phone

The phone commands are used to turn on or off the modem, retrieve system information and control the phone vibrator functions.

AT command	Action	Objects affected	Persistent object	Hook placement
AT+CFUN	Turn network on,off (values 0,1 respectively)	Phone functionality	None	`GSMD_PHONE_POWERUP` `GSMD_PHONE_POWERDOWN`
AT+CIMI	Get International Mobile Subscriber Identity (IMSI)	IMSI (Network data)	None	`GSMD_PHONE_GET_IMSI`
AT+CGMI	Get Manufacturer	Network data	None	`GSMD_PHONE_GET_MANF`
T+CGMM	Get Model	Network data	None	`GSMD_PHONE_GET_MODEL`
AT+CGMR	Get Revision	Network data	None	`GSMD_PHONE_GET_REVISION`
AT+CGSN	Get Serial Numberr	Network data	None	`GSMD_PHONE_GET_SERIAL`
AT+CBC	Get Battery info	Network data	None	`GSMD_PHONE_GET_BATTERY`
AT+CVIB	Enable/Disable Vibrator	Phone Settings	None	`GSMD_PHONE_VIB_ENABLE/DISABLE`

Table 4.3. Phone Hooks

Threat: Using these commands, untrusted applications can acquire the IMSI from the SIM card and send it to malicious parties. Since, the IMSI is used to identify the mobile subscriber, it can be used to perform attacks on behalf of the user.

4.3.2.2 Network

The network operations control network settings such as network operator and voicemail settings, and they provide options for querying phone number and signal quality.

AT command	Action	Objects affected	Persistent object	Hook placement

AT+COPS	Select operators, to request the current operator details, and to request a list of the available operators.	Network	None	`GSMD_NETWORK_REGISTER`
				`GSMD_NETWORK_DEREGISTER`
				`GSMD_NETWORK_OPER_GET`
				`GSMD_NETWORK_OPER_LIST`

AT+CSVM	Get and set voicemail service	Voicemail settings	None	`GSMD_NETWORK_VMAIL_GET` `GSMD_NETWORK_VMAIL_SET`
AT+CSQ	Get signal quality	None	None	`GSMD_NETWORK_SIGQ_GET`

AT+CPOL	Get or set SIM preferred list of networks	preferred list of networks	None	`GSMD_NETWORK_PREF_LIST`
				`GSMD_NETWORK_PREF_DEL`
				`GSMD_NETWORK_PREF_ADD`
				`GSMD_NETWORK_PREF_SPACE`

AT+CNUM	Get MSISDN (phone number(s)) from the SIM	read MSISDN	None	`GSMD_NETWORK_GET_NUMBER`

Table 4.4. Network Hooks

Threat: Untrusted applications can tamper with voicemail setting or preferred operator setting causing service interruption.

4.3.2.3 SMS

The SMS commands are used to manage the SMS storage on the SIM card, and carry out read/write/list operations on SMS records.

AT command	Action	Objects affected	Persistent object	Hook placement
AT+CMGL	Get message list with status	Messages on SIM	None	`GSMD_SMS_LIST`
AT+CMGR	Read a message	Message on SIM	None	`GSMD_PHONEBOOK_FIND`
AT+CMGW	Write a message	Message on SIM	None	`GSMD_SMS_WRITE`
AT+CMGS	Send a message	Network	None	`GSMD_SMS_SEND`
AT+CMGD	Delete a message	Message on SIM	None	`GSMD_SMS_DELETE`
AT+CPMS	Set and get message storage	Message storage on SIM	None	`GSMD_SMS_GET_MSG_STORAGE` `GSMD_SMS_SET_MSG_STORAGE`
AT+CSCA?	Set and get service center number	Service center number	None	`GSMD_SMS_GET_SERVICE_CENTRE` `GSMD_SMS_SET_SERVICE_CENTRE`

Table 4.5. SMS Hooks

Threat: A malicious program can delete short messages stored on the SIM or can send out spam messages to other subscribers without the user consent.

4.3.2.4 Phonebook

Same as the SMS, the SIM card maintains storage for user contacts. These commands allow clients to perform read/write/list operation on contact records.

AT command	Action	Objects affected	Persistent object	Hook placement
AT+CPBS	Set and get phone book storage	Phonebook storage on SIM	None	`GSMD_PHONEBOOK_LIST_STORAGE` `GSMD_PHONEBOOK_SET_STORAGE`
AT+CPBF	Find a phonebook record	Phonebook on SIM	None	`GSMD_PHONEBOOK_FIND`

AT+CPBR	Read a phonebook record	Phonebook on SIM	None	`GSMD_PHONEBOOK_READ`
				`GSMD_PHONEBOOK_READRG`
				`GSMD_PHONEBOOK_GET_SUPPORT`

AT+CPBW	Write a phonebook record	Phonebook on SIM	None	`GSMD_PHONEBOOK_WRITE` `GSMD_PHONEBOOK_DELETE`

Table 4.6. Phonebook Hooks

Threat: Phonebook entries can be used to create a hit list that a worm can use to spread itself. In addition, these phonebook information can be sent out to untrusted parties to be used for mobile phone and advertising.

The above operations are generated by *gsmd* clients and forwarded to the modem in a synchronous manner. Each command is associated with a response that is forwarded to the requesting client. In the next section we will describe a different class of operation that are created at the modem and forwarded to *gsmd* clients.

4.3.3 Modem Generated Events

These are events generated at the modem to update *gsmd* with any change in the modem status. For example, AT+CSQ command is send to *gsmd* when signal quality changes. Some of these events provide sensitive data such as call status change, network operators change and cipher status change. We decided to place hooks that decided which class of application can receive this information.

4.3.4 Untrusted Legitimate Operations

Untrusted applications can perform some legitimate operations in *gsmd* , For example, a gaming application can query the phone model (AT+CGMM) and battery status (AT+CBC). These operation can be added as an allow rule in *gsmd* security policy granting untrusted application access to these information.

Later in chapter 6, we use the above analysis to design and implement the security hooks that control security operations in *gsmd* .

In the next chapter, we will cover the steps of porting SELinux on the phone. SELinux is required by the userland LSM in *gsmd* to query and provide access decision according to the user defined policy.

Chapter 5

Porting SELinux in the OpenMoko Environment

In this chapter we describe the OpenMoko build environment and OpenEmbedded (OE) toolchain. First, we will present the architecture of the toolchain. Then we will show the steps to create and port SELinux packages.

5.1 The OpenEmbedded Toolchain

OpenMoko uses the OpenEmbedded (OE) tool chain. OE is a development environment that targets building different Linux distribution on various devices It also supports configuration and compilation caching to speed the building process. The toolchain compiles native packages on the host (i386) which are used to cross compile packages for the target system. In our case, the target is the neo173 arm based phone. OE builds three types of packages: host, target and cross.

- The host packages are the cross compiler packages such as glibc, gcc, mtd, flex and zlib. These packages are followed by the suffix "native" such as zlib-native.
- The target packages are cross compiled for arm processors.
- The cross packages, (linux-gnueabi-arm-gcc) and linker, are used to build the target packages.

5.1.1 Version Control

The tool version uses three different tools to obtain and update code from the different source trees: *monotone*, a distributed version control system, is responsible for getting the OpenEmbedded packages directory. Svn and git are used to checkout package sources. These tools are also used to update and merge patches from and to the source trees. Versions of target packages are maintained in a versions file that is specific to each distribution. There are global variable, denied in the main Makefile, specify the revision of OpenEmbedded and OpenMoko that are checked out.

5.1.2 The Toolchain Structure

The MokoMakefile defines various make targets to set up and build the environment. During the setup phase OE is checked out using *monotone* then *bitbake*, a tool for managing build metadata and executing build tasks, is checked out using svn.

A config file build/conf/local.conf defines the target distribution (OpenMoko in our case) and target hardware (arm). In addition to these variables, the config file defines the host architecture, number of build threads and global debugging variables. Below is an example of the config file.

```
MACHINE = "om-gta01"
DISTRO = "openmoko"
BUILD_ARCH = "i686"
INHERIT += "rm_work"
PARALLEL_MAKE = "-j 10"
BB_NUMBER_THREADS = "10"
ENABLE_BINARY_LOCALE_GENERATION = "0"
DISTRO_EXTRA_RDEPENDS += "SELinux-pms-support trunk-pms"
```

Listing 5.1. local.conf file

The build process will create two additional directories: The first one is "build" which is used to maintain the compiled packages, rootfs, distribution image and staging files. The second one is "sources" which includes a copy of all checked out packages.

5.1.3 Building the Local Overlay

In order to build our customized image and packages, we created a local overlay that includes the new packages we have ported into the system and new packages we have

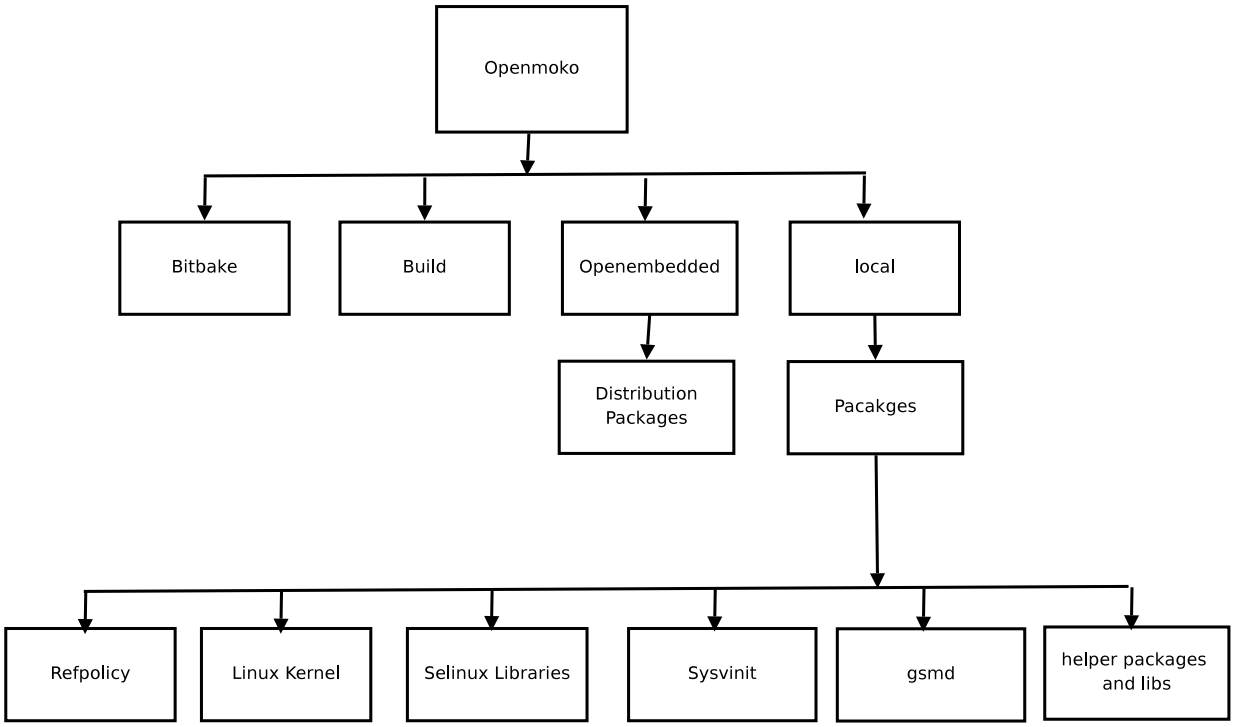

Figure 5.1. Openmoko Directory Structure

developed. We moved existing packages in order to add SELinux support and create our reference monitor. Following is a list of packages moved to the local overlay and the reason we modified these packages:

- linux: We enabled the SELinux paramters in the defconfig to build SELinux support in the kernel
- busybox: In this package, The defconfig was modified to add SELinux support for some shell commands such as ls and added SELinux commands.
- ppp: We modified the init.d scripts to start point-to-point deamon on startup. This facilitates connecting to the emulator.
- sysvinit: We added the SELinux patch to sysvinit to load the policy during boot time.
- gsmd: This is the main package of interest. We will discuss changes to gsmd in the next chapter.

Other libraries related to SELinux and the reference policy were ported under the local overlay. We will explain these packages in details in the Porting SELinux section.

As mentioned earlier, the toolchain config refers to an image as a build target. The image describes what packages are included in the distribution. The next section will describe the image *bitbake* file for the customized distribution.

5.1.4 Customizing the Image

The OpenMoko image is built from a collection of tasks. Each task is a collection of packages that are grouped by functions such as kernel, user interface, networking, games , personal information management (pim) and telephony applications.

To reduce the build time, the customized image excludes the games and pim packages. Following is the image file for the SELinux customized distribution:

```
#----------------------------------
# OpenMoko SELinux Image Recipe
#----------------------------------

export IMAGE_BASENAME = "${PN}"

export IMAGE_INSTALL = "\
    ${MACHINE_TASK_PROVIDER} \
    task-openmoko-linux \
    task-openmoko-net \
    task-openmoko-ui \
    task-openmoko-base \
    task-openmoko-phone \
    task-openmoko-security \
    gdb \
    "

inherit image
```

Listing 5.2. Openmoko-selinux-image

In addition to the OpenMoko predefined tasks, we defined an openmoko-security task that will include the SELinux packages and reference policy.

```
CRIPTION = "OpenMoko: Selinux"
SECTION = "openmoko/base"
```

```
LICENSE = "original"
PR = "r0"
inherit task
DESCRIPTION_task-openmoko-security= "Openmoko: selinux Tresys"
RDEPENDS="selinux-pms-support \
          refpolicy-pms-support\"
```

Listing 5.3. Task security

5.2 Bitbake

Bitbake is a task executor that is used to build packages in OpenEmbedded[31]. It uses three types of metadata for task execution:

- Bitbake.conf: This file provides configuration for the target distribution and versioning configuration.
- Bbclasses: These files defines the tasks to be executed. Classes are based on inheritance. The main class is base.bbclass which defines the base tasks. Tasks will be explained in more details in the next section.
- Recipe: Each package will have a .bb recipe file that defines the location of the src, revision, dependencies and which classes to be used.

5.2.1 Classes and Tasks

A class defines the variables and steps (tasks) for configuration, compilation and installation of packages. For example, native.bbclass uses make with CC=gcc and arch=i386 while auttools.bbclass uses CC=cross-gcc for cross compiling. Here is the list of common tasks that are defined in most classes:

- do_fetch: This task is responsible for fetching source code from source control repositories such as git or svn.
- do_unpack: This task is responsible for extracting files from archives, such as .tar.gz, into the working area and copying any additional files.
- do_patch: This task is responsible for applying any patches to the unpacked source code mainly using quilt.

- do_configure: This task runs the configure script ("./configure ¡options¿").
- do_compile: This task is responsible for running make for compilation.
- do_populate_staging: This task is responsible for copying libraries and headers, to the staging folder, required by other packages to build (for linking).
- do_install,do_package: This tasks are responsible for installing the files under the image or install directory to be shipped later in the image.

Additional tasks can be defined in a new class such as the QA tasks defined in insane.bbclass. In addition, recipe files (.bb) can append, preppend or override the above tasks.

5.2.2 Example Recipe

The following recipe is used for porting SELinux support libraries:

```
DESCRIPTION = "This library provides a set of interfaces for
security-aware applications to get and set process and file
security contexts and to invoke the policy API."
HOMEPAGE = "http://oss.tresys.com/projects/policy-server/"
AUTHOR = "Tresys"
DEPENDS += "libustr flex-native"
RDEPPENDS +="libustr"
SRCREV = "${AUTOREV}"
PACKAGES = "${PN}"
S=${WORKDIR}/selinux-pms-support
SRC_URI = "svn://oss.tresys.com/repos/policy-server/branches/
            ;module=${PN};proto=http\
          file://openmoko_port_Makefile.patch"
export DESTDIR=${D}
FILES_${PN} = "\
    ${bindir}/* \
    ${sbindir}/* \
    ${libexecdir}/* \
    ${libdir}/lib*.so.* \
    ${sysconfdir} \
    ${sharedstatedir} \
```

```
    ${localstatedir} \"
inherit autotools
do_clean_prepend(){
#clean staging
        dir = bb.data.expand("${STAGING_LIBDIR}",d)
        if dir == '//': raise bb.build.FuncFailed("wrong DATADIR")
        bb.note("removing " + dir + "selinux sepol semanage")
        os.system('rm -rf ' + dir + '/libselinux.* '
                    +dir+'/libsemanage.* '+dir+'/libsepol.*' )
{
do_install_append(){
        mkdir ${D}/selinux
}
do_stage() {
  autotools_stage_all
}
```

Listing 5.4. Libselinux recipe

The first set of variables provides package description and name. Then the DEPENDS variable define expected dependencies to be compiled before this package. The SRC_URI is the location of the source files (svn) and additional patch files required for porting. The package uses the autotools class for building target machine packages. We append some tasks, required for porting, to the clean and install tasks.

5.3 Porting SELinux

In the previous section, we have shown an example of a bb file for porting SELinux userspace libraries. We defined a certain process for porting packages that was followed for porting most of the packages.

5.3.1 Porting Process

Steps for porting a package:

1. Find and download the source for the package of interest.

2. Create a basic bb file for the package and inherit autotools as trhe basic class.

3. Define the packages description variables and any required dependencies.
4. Build the package using *bitbake* with debug flags.
5. Some errors will occur due to arm incompatible parameters or missing dependencies.
6. Using quilt modify the makefiles or source code to fix the compilation error. Create a patch of fixes called openmoko_port_*.patch
7. Add the patch to the bb files and recompile.
8. Some packages might require task overriding. For example, the refpolicy required changes in the do_compile is it build pp modules which requires "make conf" before the "make" command.

5.3.2 Packages Ported

We ported/modified different sets of packages. The first set is SELinux libraries including libsepol, libselinux, libsemanage, policycoreutils, checkpolicy and refpolicy. The major fixes for these packages were dependency issues and invalids cross compiler parameters.

The second set of packages is modified from the original. This includes linux, sysvinit and busybox.

- In linux we had to enable SELinux in the kernel defconfig file as follow:

```
CONFIG_SECURITY_NETWORK=y
CONFIG_SECURITY_CAPABILITIES=y
CONFIG_SECURITY_SELINUX=y
CONFIG_SECURITY_SELINUX_BOOTPARAM=y
CONFIG_SECURITY_SELINUX_BOOTPARAM_VALUE=1
CONFIG_SECURITY_SELINUX_DEVELOP=y
CONFIG_SECURITY_SELINUX_AVC_STATS=y
CONFIG_SECURITY_SELINUX_CHECKREQPROT_VALUE=1
```

Listing 5.5. Enabling SELinux in defconfig

- In busybox, we enabled SELinux for some of the shell commands such as ls and ps to add the -Z option for displaying security contexts.

- In order to enable SELinux, sysvinit was patched to load the policy and mount the selinuxfs under /selinux.

Openmoko uses jffs2 which is a log-structured file system intended for embedded devices. By default, extended attributes are disabled in jffs2. Therefore, we had to enable extended attributes which is required to store security context for files. Extend attributes in jffs2 requires porting additional dependencies such as libattr and libacl.

In addition to the package porting some parameters where changed during the phone image creation and flashing processes. This includes adding "–with-selinux" to mkfs.jffs2 command and setting "selinux=1 enforcing=0" as boot parameters.

In conclusion, we have provided an overview of environment setup that is required for integrating SELinux in the Openmoko distribution. In the next chapter we will utilize this setup to secure the telephony server.

Chapter 6

Securing Gsmd

This chapter covers the different types of objects (Persistent and non-persistent) in *gsmd* and the mechanism to mediate access to these objects. We discuss code modification in *gsmd* to enable SELinux. In addition, we propose a design, for interaction between *gsmd* and the sound subsystem, is proposed to mediate access to sound files used by *gsmd*. Finally, we will present an example policy for *gsmd* used in our testing.

6.1 Gsmd Hooks

The target of the thesis is to mediate access to the network and phone resources via *gsmd* hooks. The hooks will prevent untrusted application from issuing commands that can either affect the network or user data. However, these applications can perform some non-critical operations such as querying battery status and revision information.

We used the analysis in Chapter 4 to decide on where to place the hooks in *gsmd*. This analysis is used to derive the SELinux classes and permissions. We focused on the granularity of classes and permissions to help create a flexible policy for each application type. For example, a messaging application can have full permissions for the SMS and phonebook classes.

6.1.1 Design Philosophy

There are two approaches for hook placement in a reference monitor. The first approach is bottom-up in which hooks are placed at every access to an object. This approach will not only ensure complete mediation, but also it will create redundancy in access checks, and will result in a complex policy definition. Especially if we know that the access

parameters (source, target and operation) will not change along the execution path.

The second approach is top-down, where an analysis is carried out to identify security operations and mediate access. For example, if we try to control access of an application to system files, we can trace the application and look for read/write system call, and then identify these system calls as critical operations that require a hook function. The problem with such an approach arises from the fact that the analysis tool might not be able to cover all execution paths allowing for some security holes in the system.

In our design we used a combination of both approaches. First we carried out the static and dynamic analyses to identify critical operations. We found out that all requests, coming from the clients, have to pass through a socket callback function. This callback function will dispatch the commands using a pointer function array indexed by the different request type such as voicecall, sms and phonebook. Inside each callback there is a *switch-case* statement that parses the request subtype (dial,halt,..) and creates the AT command accordingly. In this case, the request represented the object and the request subtype represented the operation. Based on these findings, we placed the hook after the *case* in each function to make sure that we mediate access to each object. Each object can fall into one of two class:

- *Persistent objects*: From *gsmd* prospective, the state of these object doesn't change during runtime. For example, the SMS storage, network and user PIN.
- *non-persistent objects*: Phone calls and GPRS connections fall under this category. This objects are created and destroyed inside *gsmd*

We will discuss this object classification in details in the next subsection.

In addition, our analysis showed that there are unsolicited events generated at the modem that are broadcasted to all clients. Since, some of these notifications are considered sensitive (such as network status and signal quality). We placed hooks at the events callback to mediate access to who can receive these notifications.

The second category of objects is non-persistent objects. Since Access decisions for these objects are based on the security context of the owner who created the object and the security context of the entity trying to manipulate the object, we labeled these objects as *Typed non-persistent objects*.

6.1.1.1 Persistent Objects

These classes of objects do not require dynamic labeling since they are neither created nor destroyed in *gsmd*. Hence we define a static type for these objects, and access

decisions are based on their fixed types. For example we define the following class for the phonebook storage:

```
#define PHONEBOOK_T "system_u:object_r:phonebook_t"
```

Any decision on phone book operations will take the client context as the source context and PHONEBOOK_T as the target context.

The hooks for persistent object are placed directly before the operation execution taking in consideration the constraints defined in chapter 4. Following is an example of hook for the network object:

```
    switch (gph->msg_subtype) {
      case GSMD_NETWORK_REGISTER:
              if(security_network_register(gu))
              {
                      return -EACCES;
              }
              if ((*oper)[0])
                      cmdlen = sprintf(buffer,
  "AT+COPS=1,2,\"%.*s\"",
                                      sizeof(gsmd_oper_numeric),
  (char *)oper);
              else
                      cmdlen = sprintf(buffer,
  "AT+COPS=0");
              cmd = atcmd_fill(buffer, cmdlen + 1,
 &null_cmd_cb, gu, 0, NULL);
              break;
      case GSMD_NETWORK_DEREGISTER:
              if(security_network_deregister(gu))
              {
                      return -EACCES;
              }
              cmd = atcmd_fill("AT+COPS=2",
  9+1, &null_cmd_cb,
  gu, 0, NULL);
              break;
```

```
        case GSMD_NETWORK_QUERY_REG:
                if(security_network_query_reg(gu))
                {
                        return -EACCES;
                }
                cmd = atcmd_fill("AT+CREG?",
    8+1, &network_query_reg_cb,
    gu, 0, NULL);
                break;
```

Listing 6.1. Network hooks

6.1.1.2 Unsolicited Events

These modem generated events are broadcasted to all connected clients. The hooks are placed at the callback to check if the target client is allowed to receive the event or not. Following is an example of unsolicited event hook:

```
int usock_evt_send(struct gsmd *gsmd,
                   struct gsmd_ucmd *ucmd, u_int32_t evt)
{
  struct gsmd_user *gu;
  int num_sent = 0;
  llist_for_each_entry(gu, &gsmd->users, list) {
   if (gu->subscriptions & (1 << evt)) {
   if(usock_evt_security_check(gu,evt))
                return -EACCES;
    if (num_sent == 0)
     usock_cmd_enqueue(ucmd, gu);
```

Listing 6.2. Unsolicited Event Hook

6.1.1.3 Typed Non-Persistent Objects

There objects are created and destroyed dynamically in gsmd. An example of these objects is phone calls; when a client send a dial request, a call will be created in the system and the call will be labeled with the security context of the creator. Later, if

another client, who have voicecall permissions, tries to hold or halt this call, the hook will check whether the policy allow such operation or not.

For example, client 1 with context *system_u:object_r:call_manager_t* dial a call. The system will create this call and assign to it client 1 context. When client 2, who has context *system_u:object_r:dialer_t*, tries to halt this call the hook will check if dialer_t can halt objects of type call_manger_t. In the implementation we defined two non-persistent objects: current call and call on hold. The hooks are placed at voice call operations such as dial, halt, and call control (hold and swap calls).

6.1.1.4 Design Extension

These 2 objects are used for proof of concept. Although 2 is the number of calls that are supported in the current system. A table of calls can be build to hold a larger number of calls. This design will require labeling the calls with call_ids that can be obtained from the modem. Once a client issues a call command to the modem, a timer must be created to query the call id and call state in the modem. In this design, all access requests will be queued in a pending queue. Once a call id is obtained from the modem, and the call state is active, the pending access requests are evaluated. In addition to phone calls, we can include GPRS connection as a class of non-persistent objects. This will allow for protecting data channels from being altered by different clients other then the owner (or whatever the policy defines).

6.2 Adding SELinux Support in Gsmd

6.2.1 Initialization

Usesapce servers uses the interface provided by *libselinux* to query access decision from the kernel security server (refer to background work). Each application that uses SELinux is required to initialize its own cache and to pass handlers for auditing and thread management. If null handlers are passed then *libselinux* will use the default *pthread* handlers for thread management, and all log messages will be forwarded to *dmesg* kernel logging ring buffer.

We define our audit handler to forward audit messages to *gsmd* logger. This will facilitate debugging, testing, and separates kernel security audits from application level audits.

```
Audit handler:
void audit_print(const char *fmt, ...)
{
    /* we use gsmd logging of the default dmesg */
    char buf[1024];
    va_list ap;
    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    DEBUGP("%s",buf);//gsmd logger
}

In gsmd_avc_init:
    avc_entry_ref_init(&(g->gsmd_aeref));
    /* use standard malloc/free for the memory callbacks */
    if (avc_init("gsmd", NULL, &alc, &atc, &akc) < 0) {
        DEBUGP("could not initialize avc");
        exit(1);
    }
```

Listing 6.3. AVC initialization

6.2.2 Defining Security Operations

SELinux defines security operations for the kernel objects such as files, sockets and processes. However, userspace applications define their own operation for finer-grained or additional control. In order to define such operation, we followed the LSM architecture; we define hooks as inline functions. Each function invokes a callback in the *security operation* structure (Security Module).

This design pattern allows for different implementation of the security module. For example, our implementation is focused on Type Enforcement as our security goal. Other implementations can focus on Multilevel Security. Therefore we placed the hooks in *gsmd* without restricting which implementation to use and we leave that for the loaded module.

We register *gsmd*sec_ops as our security module when we initialize *gsmd*. The following code will show a declaration for the security_ops structure and gsmdsec_ops:

```
Security Operations Structure:
struct security_operations {
        int (*voicecall_dial) (struct gsmd_user *gu);
        int (*voicecall_hangup) (struct gsmd_user *gu);
        ...
 int (*phone_powerup) (struct gsmd_user *gu);
        int (*phone_powerdown) (struct gsmd_user *gu);
 ...
 ...
 }
Callback assignment:
static struct security_operations gsmdsec_ops = {
        .voicecall_dial = gsmdsec_voicecall_dial,
        .voicecall_hangup = gsmdsec_voicecall_hangup,
        ...
 .phone_powerup = gsmdsec_phone_powerup,
        .phone_powerdown = gsmdsec_phone_powerdown,
 ...
 ...
 }
Registering gsmdsec_ops:
int gsmdsec_init(struct gsmd *g)
{
         /*avc init*/
         if(gsmd_avc_init(g)!=0)
         {
                DEBUGP("Failed to init avc cache\n");
                return -1;
         }
         /*register security module*/
         if(register_security (&gsmdsec_ops)!=0)
         {
                DEBUGP("Failed to register security module");
                return -1;
         }
```

```
        return 0;
}
```

Listing 6.4. Security Operations

6.3 Userspace LSM Implementation and Access Control

The security module implements the enforcement mechanism that satisfies the application security goal. Type enforcement is used to decide which subject can access which object. This decision is based on allow rules defined in the policy. The subject security is obtained when the client is connected to the *gsmd* server.

6.3.1 Obtaining Client Context and Credentials

Gsmd uses Unix domain sockets to communicate with clients. The server listen on a master socket, once a handshake is established a callback is invoked to setup a gsmd_user data structure. The structure holds information about the client such as the socket file handler and commands masks. In addition to these fields we defined security context, security SID and process credentials (user id, group id and process id). When a client is connected we obtain the security context through the SELinux socket function *getpeercon()*. The context is converted to an SID by calling *avc_context_to_sid()*. The SID is used for future references in permission check functions. The process credentials represent the client information that is used in logging access requests.

6.3.2 Hook Implementation

Hook implementations are located in the loadable security module. Security checks are done via calling the *avc_has_perm* function. If SELinux is in enforcement mode, the hooks will return {-1} in case of access denial and 0 otherwise. Since we are using *libselinux* for *gsmd*, we modified the library to act as if it is in enforcing mode. However, the kernel is still in permissive mode. Below is an example of implementation for the voicecall hook.

```
int gsmdsec_voicecall(struct gsmd_user *gu,int action)
{
        int perm=0;
        char *avc_msg=malloc(1024*sizeof(char));
```

```
        sprintf(avc_msg,"client pid %d",
  gu->pid);
        security_id_t call_sid;
        if (avc_context_to_sid(CALL_T,
    &call_sid) < 0) {
                DEBUGP("could not get current call sid");
                return 0;
        }
        perm=avc_has_perm(gu->sid,call_sid,
   SECCLASS_GSMD_VOICECALL,
   action,&(g.gsmd_aeref),
   avc_msg);
        gsmdsec_log(perm);
        return perm;
}
```

Listing 6.5. Voicecall hook implementation

6.4 Gsmd and The Sound Subsystem

Voice is routed to and from the modem via analog connection. These connections deliver the sound to the speaker and from the microphone using the Wolfson Codec sound chip; this routing happens at hardware and driver level. For userspace, the only exported interfaces for accessing or manipulating the sound are through the /dev/snd files and through the ALSA (Advanced Linux Sound Architecture) library.

This can create a threat to current voice calls as it allows untrusted applications to sniff voice data or inject data in the call stream. Therefore, we will present two different designs that will mediate access to the sound subsystem. The main objective is to dynamically label sound objects using the label of the client who created the call.

6.4.1 Files Labeling Model

In this model, when a voice call is created, *gsmd* will relabel the file with the client security context and store the file original label. Once the call is halted, *gsmd* will restore the file original label.

This model will allow only programs that have the same label as the creator to access

the sound files. If we consider the creator label as trusted then only trusted applications can access the sound files.

A limitation of this model is that some application might request access to the sound file during a voice call. For example, a messaging application may need to send a voice notification of new incoming SMS which is considered a desirable feature in mobile phones.

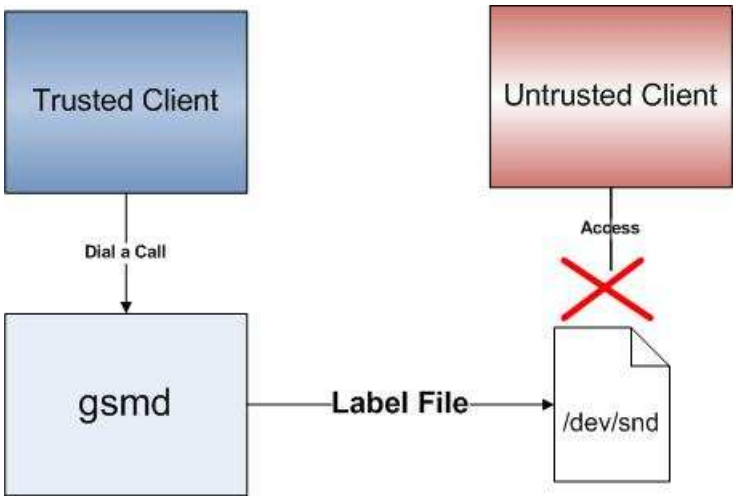

Figure 6.1. Gsmd and ALSA Files Labeling Model

6.4.2 Policy Management Server Model

The Policy Management Server allows userspace application to modify and query the policy dynamically. Assume that there exists a policy that allows only trusted application to access the sound files such as the ALSA sound server; then When a call is created, *gsmd* will update this policy by adding an *allow* rule to permit the dialer application to access the sound files.

We can mediate access via the ALSA libraries. If ALSA has security hooks placed at the ALSA library, then the library can query the policy server to allow only trusted application (defined by the policy) to access the sound files.

The second model is preferable than the first one because it extends the current sound subsystem policy rather than creating an exclusive access to the sound files.

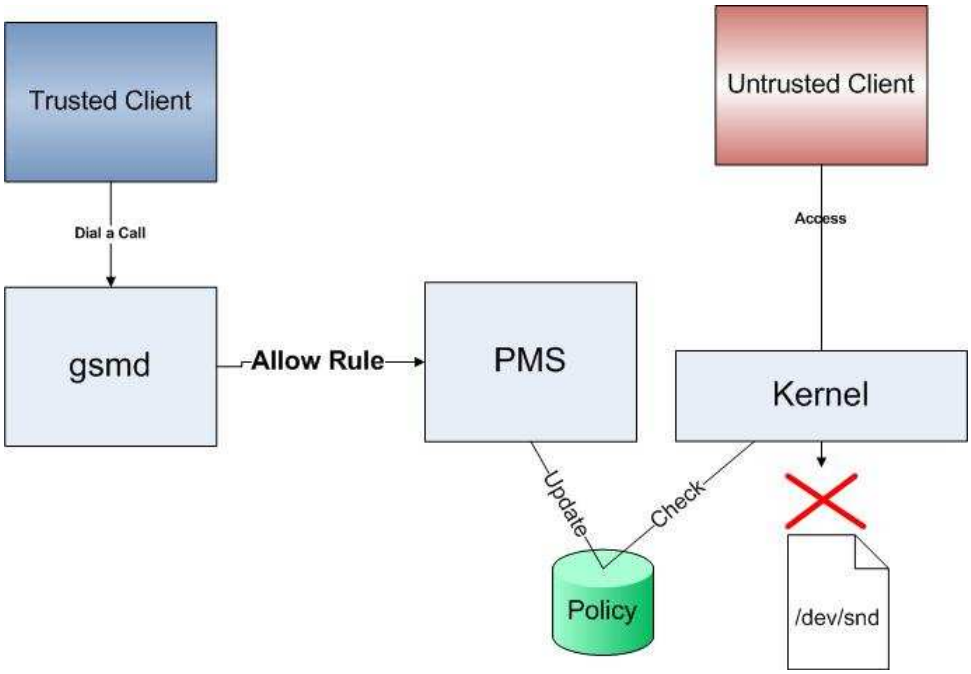

Figure 6.2. Gsmd and ALSA Policy Management Server Model

6.5 Gsmd Policy

We used a non-monolethic refpolicy that was developed by Tresys[32]. This policy is based on NSA example policy and it merges both targeted and strict policy.

Since the current interface does not support defining classes witching module, we have to hard code the classes and permissions in *policy/flask/security_classes* and *policy/flask/access_vectors*. After modifying these files we used a *perl* script that generate header files for *libselinux* to become aware of the newly added classes and permissions.

Using the classes and permissions we defined, we created a sample policy for testing *gsmd*. The policy defines a call manager type, that can carry a voice call and phonebook operations, and it defines an untrusted type that can query phone revision and battery state. Below is a snippet of the allow rules in the policy:

```
allow call_manager_t current_call_t:gsmd_voicecall
            {dial answer hangup dtmf get_stat control forward};
allow call_manager_t phonebook_t:gsmd_phonebook
            {list_storage set_storage find read write delete get_support};
allow call_manager_t phone_t:gsmd_phone
            {powerup powerdown power_status get_imsi get_manf get_model
            get_revision get_serial get_battery vib_enable vib_disable};
```

```
allow call_manager_t ucmd_t:gsmd_general {ucmd_submit};
allow call_manager_t network_t:gsmd_network
            {register deregister query_reg get_operators list_operators
            get_vmail set_vmail get_sigq list_preffered_network
            add_preffered_network del_preffered_network
            space_preffered_network get_phone_number};
allow untrusted_t phone_t:gsmd_phone
            {power_status get_manf get_model
            get_revision get_battery};
```

6.6 Results

We created two different clients: a trusted client C1 that can perform all telephony commands, and untrusted client C2 that can only query phone revision and battery status. Following is the output of the log file when both C1 and C2 connect to gsmd and try to issue a dial command.

```
Fri Aug  4 17:54:53 2008 <1> usock.c:1972:gsmd_usock_cb()
            Handshake with client context system_u:system_r:trusted_t.
Fri Aug  4 17:54:53 2008 <1> usock.c:1978:gsmd_usock_cb()
            Peer's pid=1330, uid=0, gid=0
Fri Aug  4 17:55:30 2008 <1> hooks.c:11:gsmdsec_log()
            granted


Fri Aug  4 17:56:29 2008 <1> usock.c:1972:gsmd_usock_cb()
            Handshake with client context system_u:system_r:untrusted_t.
Fri Aug  4 17:56:29 2008 <1> usock.c:1978:gsmd_usock_cb()
            Peer's pid=1331, uid=0, gid=0
Fri Aug  4 17:57:00 2008 <1> avc.c:16:audit_print()
            gsmd:  denied  { dial } for client pid 1331
            scontext=system_u:system_r:untrusted_t
            tcontext=system_u:object_r:current_call_t tclass=gsmd_voicecall
```

AS the log shows, *gsmd* will obtain the security label of the clients when they establish a connection. When C! Issues the dial command, a security check is preformed and

access is granted as defined in the policy. However, when C2 try to perform the same dial operation, access is denied since there is no allow rule for untrusted subject to make phone calls.

Chapter 7

Conclusion and Future work

The advancement in mobile phones has created new security challenges. The current protection mechanisms in the phone system do not grantee security for both user data and mobile networks. We have illustrated the threat model by exploiting the Motorola phone and running a malicious code that was able to make unauthorized phone calls. Our thesis proposed a reference monitor that can prevent such malicious behavior.

First, we analyzed the structure and control flow of the gsmd server, and then we used this analysis to design our reference monitor. The reference monitor consists of hooks that are placed at security critical points in the code and a userspace LSM that implements the security mechanism for these hooks.

We have shown that the reference monitor and a well defined policy provide a protection for both the telephony services and user information. The hook placement allows the user to define a fine-grained policy that does not only protect against malicious applications, but also it protects the telephony server from faulty application.

Future work would be focused on implementing the sound system protection model discussed in chapter 6. In addition, we can extend the reference monitor to include other telephony services such as GPRS or similar data oriented services.

Bibliography

[1] Weinberg, B. (2006) "Mobile phones: the embedded linux challenge," *Linux J.*, **2006**(148), p. 9.

[2] "Mobile Extreme Convergence: A Streamlined Architecture to Deliver Mass-Market Converged Mobile Devices," `www.freescale.com/files/wireless_comm/doc/brochure/MXCWP.pdf`.

[3] "GSM 07.07 specifications," `http://www.ctiforum.com/standard/standard/etsi/0707.pdf`.

[4] Kingpin and Mudge (2001) "Security Analysis of the Palm Operating System and its Weaknesses Against Malicious Code Threats," in *10th USENIX Security Symposium*, pp. 135–152.

[5] "Symbian Fast facts," `http://www.symbian.com/about/fastfacts/fastfacts.html`.

[6] Heath, C. (2006) *Symbian OS Platform Security*, John Wiley & Sons.

[7] "Windows Mobile 5.0 Application Security," `http://msdn2.microsoft.com/en-us/library/ms839681.aspx`.

[8] "Trusted APIs," `http://msdn2.microsoft.com/en-us/library/ms924486.aspx`.

[9] "Security Evaluation of Apples iPhone," `www.securityevaluators.com/iphone/exploitingiphone.pdf`.

[10] Knudsen, J. (2003), "Understanding MIDP 2.0's Security Architecture," `http://developers.sun.com/techtopics/mobility/midp/articles/permissions/`.

[11] "Blackberry Features," `http://eu.blackberry.com/eng/ataglance/security/features.jsp`.

[12] "Linux Devices," `http://www.linuxdevices.com/articles/AT9423084269.html`.

[13] "OpenEZX project," http://www.openezx.org/.

[14] "National Security Agency. Security-Enhanced Linux (SELinux)," http://www.nsa.gov/selinux.

[15] Carter, J. (2007) "Using GConf as an Example of How to Create an Userspace Object Manager," in *2007 SELinux Symposium*, National Security Agency.

[16] "SE Linux Policy Server," http://oss.tresys.com/projects/policy-server.

[17] Enck, W., P. Traynor, P. McDaniel, and T. L. Porta (2005) "Exploiting open functionality in SMS-capable cellular networks," in *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, ACM Press, New York, NY, USA, pp. 393–404.

[18] BOCAN, V. and V. CRETU (2006) "Mitigating Denial of Service Threats in GSM Networks," in *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, IEEE Computer Society, Washington, DC, USA, pp. 523–528.

[19] Traynor, P., V. Rao, T. Jaeger, P. McDaniel, and T. L. Porta (2007), "From Mobile Phones to Responsible Devices," .

[20] Berman, A., V. Bourassa, and E. Selberg (1995) "TRON: Process-Specific File Protection for the UNIX Operating System," in *USENIX Winter*, pp. 165–175.

[21] Goldberg, I., D. Wagner, R. Thomas, and E. A. Brewer (1996) "A secure environment for untrusted helper applications confining the Wily Hacker," in *SSYM'96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, USENIX Association, Berkeley, CA, USA, pp. 1–1.

[22] Chari, S. N. and P.-C. Cheng (2003) "BlueBoX: A policy-driven, host-based intrusion detection system," *ACM Trans. Inf. Syst. Secur.*, **6**(2), pp. 173–200.

[23] "systrace," http://wiki.openembedded.net/index.php/Getting_Started.

[24] "Apparmor," http://developer.novell.com/wiki/index.php/Apparmor_FAQ.

[25] "Java Security," http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc1.html.

[26] "Safe TCL," http://www.tcl.tk/software/plugin/safetcl.html.

[27] Walsh, E. F. (2007) "Application of the Flask Architecture to the X Window System Server," in *2007 SELinux Symposium*, National Security Agency.

[28] development team, S.-P., "The Security-Enhanced PostgreSQL Security Guide," http://sepgsql.googlecode.com/files/sepgsql_security_guide.20080214.en.pdf.

[29] "Motorola A780," http://www.motorola.com/motoinfo/product/details.jsp?globalObjectId=70.

[30] "OpenMoko Wiki," http://wiki.openmoko.org/wiki/Main_Page.

[31] "Open Embedded," http://wiki.openembedded.net/index.php/Getting_Started.

[32] "SELinux Reference Policy," http://oss.tresys.com/projects/refpolicy/.

[33] Shackman, M., "Symbian OS v9 and Platform Security," http://www.symbian.com/files/rx/file3202.pdf.

[34] Nakamoto, Y. (2004) "Toward mobile phone Linux," in *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation*, IEEE Press, Piscataway, NJ, USA, pp. 117–124.

[35] "Baseband Processors," http://www.analog.com/en/subCat/0,2879,772%255F839%255F0%255F%255F0%255F,00.html.

[36] "Facing the Complexity Challenge of Mobile Phones," http://www.eetchina.com/ARTICLES/2005NOV/PDF/Facing_Complexity_Challenge_Mobile_Phones.PDF.

[37] "AT Command set for GSM Mobile Equipment (ME) 3GPP TS 07.07," http://www.3gpp.org/ftp/Specs/html-info/0707.htm.

[38] Loscri, V. and S. Marano (2006) "A new bi-processor SmartPhone," in *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06)*, IEEE Computer Society, Washington, DC, USA, pp. 104–111.

[39] "Nucleus Real-Time OS - (Real-Time Operating System)," http://www.mentor.com/products/embedded_software/nucleus_rtos/index.cfm.

[40] "Trusted Mobile Platform Specification (IBM, Intel, and NTT DoCoMo)," http://www.trusted-mobile.org/.

[41] Zheng, P. and L. M. Ni (2006) "Spotlight: The Rise of the Smart Phone," *IEEE Distributed Systems Online*, **7**(3), p. 3.

[42] Traynor, P., W. Enck, P. McDaniel, and T. L. Porta (2006) "Mitigating attacks on open functionality in SMS-capable cellular networks," in *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, ACM Press, New York, NY, USA, pp. 182–193.

[43] Wright, C., C. Cowan, J. Morris, S. Smalley, G. KroahHartman, s modules, and G. support (2002), "the linux kernel. In Linux Security Modules: General Security Support for the Linux Kernel," .

[44] Stberg, M., "Radio Jamming Attacks Against Two Popular Mobile Networks," .

[45] Wright, C., C. Cowan, S. Smalley, J. Morris, and G. K. Hartman (2002) "Linux Security Module Framework," in *2002 Ottawa Linux Symposium*.

[46] Sandhu, R. S., E. J. Coyne, H. L. Feinstein, and C. E. Youman (1996) "Role-Based Access Control Models," *IEEE Computer*, **29**(2), pp. 38–47.

[47] Schneider, F. B. (2000) "Enforceable security policies," *ACM Trans. Inf. Syst. Secur.*, **3**(1).

[48] Swift, M. M., A. Hopkins, P. Brundrett, C. V. Dyke, P. Garg, S. Chan, M. Goertzel, and G. Jensenworth (2002) "Improving the granularity of access control for Windows 2000," *ACM Trans. Inf. Syst. Secur.*, **5**(4), pp. 398–437.

[49] Strembeck, M. and G. Neumann (2004) "An integrated approach to engineer and enforce context constraints in RBAC environments," *ACM Trans. Inf. Syst. Secur.*, **7**(3), pp. 392–427.

[50] Kilpatrick, D., W. Salamon, and C. Vance (2003) "Securing The X Window System With SELinux," NAI Labs.

[51] "Trolltech Online Reference Documentation," `http://doc.trolltech.com/`.

[52] "Trolltech Modem Emulator Documentation," `http://doc.trolltech.com/qtopia4.2/modem-emulator.html`.

[53] Covington, M. J., W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd (2001) "Securing context-aware applications using environment roles," in *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, ACM Press, New York, NY, USA, pp. 10–20.

[54] Mayer, F., K. MacMillan, and D. Caplan (2006) *SELinux by Example: Using Security Enhanced Linux (Prentice Hall Open Source Software Development Series)*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

[55] "Security Enhanced PostgreSQL," `http://code.google.com/p/sepgsql`.

[56] "Security Enhanced DBUS," `http://www.freedesktop.org/wiki/Software/dbus`.

[57] Jones, M. T., "Anatomy of Security-Enhanced Linux (SELinux)," `http://www.ibm.com/developerworks/linux/library/l-selinux/`.