Measuring Integrity on Mobile Phone Systems

Divya Muthukumaran
Systems and Internet Infrastructure Security Laboratory
Pennsylvania State University
University Park, PA 16802
dzm133@psu.edu

Anuj Sawani
Systems and Internet Infrastructure Security Laboratory
Pennsylvania State University
University Park, PA 16802
axs1003@psu.edu

Joshua Schiffman
Systems and Internet Infrastructure Security Laboratory
Pennsylvania State University
University Park, PA 16802
jschiffm@cse.psu.edu

Brian M. Jung
Secure Systems Group
Samsung Electronics Co., Ltd.
Suwon-City, Gyeonggi-Do,
Korea, 443-742
brian.m.jung@samsung.com

Trent Jaeger
Systems and Internet Infrastructure Security Laboratory
Pennsylvania State University
University Park, PA 16802
tjaeger@cse.psu.edu

ABSTRACT

Mobile phone security is a relatively new field that is gathering momentum in the wake of rapid advancements in phone system technology. Mobile phones are now becoming sophisticated smart phones that provide services beyond basic telephony, such as supporting third-party applications. Such third-party applications may be security-critical, such as mobile banking, or may be untrusted applications, such as downloaded games. Our goal is to protect the integrity of such critical applications from potentially untrusted functionality, but we find that existing mandatory access control approaches are too complex and do not provide formal integrity guarantees. In this work, we leverage the simplicity inherent to phone system environments to develop a compact SELinux policy that can be used to justify the integrity of a phone system using the Policy Reduced Integrity Measurement Architecture (PRIMA) approach. We show that the resultant policy enables systems to be proven secure to remote parties, enables the desired functionality for installing and running trusted programs, and the resultant SELinux policy is over 90% smaller in size. We envision that this approach can provide an outline for how to build high integrity phone systems.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection — *Access Control*

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.

General Terms

Security

Keywords

Integrity Measurement, Mobile Phones, SELinux

1. INTRODUCTION

Cellular communication is changing. Mobile phones have become smaller, lighter, and more powerful, and support a wide variety of applications, including text messaging, e-mail, web surfing and even multimedia transmissions. Smart phones that are a hybrid of cell phones and PDAs that can handle voice and data communications, in essence functioning as a "tiny computer." This transformation motivated the transition from small, custom operating environments to more powerful, general purpose environments that are based on personal computer environments, such as Windows Mobile [33] and Linux phone OS projects [19, 23].

Third-party developers now provide many multimedia applications that users can easily download onto these powerful new phones. The flexibility of supporting third-party applications presents security concerns for other applications that handle critical user data. For example, mobile banking applications have been created for such phones [2], providing attackers with a valuable target. Worm attacks [16, 4] have been launched against the market-leading Symbian mobile platform [27], a variety of vulnerabilities on this platform have been identified [7, 29], and a large number of users (over 5 million in March 2006 [11]) download freeware games (i.e., potential malware) to their mobile devices. As a result, it seems likely that mobile phones, including Linux and Windows phones, will become targets for a variety of malware.

Security architectures for phone systems are emerging, but they make no concrete effort to justify critical application integrity. The Symbian security architecture distinguishes between its installer, critical applications, and untrusted applications. The Symbian approach has been effective at protecting its kernel, but some critical resources, such as phone

contacts and Bluetooth pairing information, can be compromised by untrusted applications [24]. A mandatory access control framework has been developed for Linux, the Linux Security Modules (LSM) framework [34], but LSM-based approaches (e.g., SELinux [22] and AppArmor [21]) do not ensure integrity. The SELinux LSM focuses on enforcing *least privilege*, and its policies on personal computer systems are too complex to understand integrity completely. The AppArmor LSM focuses on confining network-facing daemons, which may prevent integrity problems from untrusted network requests, but not from untrusted programs running on the system.

Our goal is to protect the integrity of critical phone applications from the untrusted code and data of downloaded third-party applications. The mobile banking application above is one critical phone application. The aim is to install and execute such trusted applications under the control of a phone policy for which precise integrity guarantees can be made. We believe that mandatory access control policies are the foundation for providing such guarantees, but the policies developed thusfar are inadequate because they are too complex or are focused on the wrong goal.

In this paper, we define a MAC policy for a Linux phone system and enable a remote party to verify the integrity of our phone systems using integrity measurements. We use the SELinux LSM as the starting point, but we reduce the policy to focus on integrity goals. In designing our phone policy, we use the CW-Lite integrity model [25], a weakened, but more practical, version of the Clark-Wilson integrity model [6] to define our precise integrity goals. Focusing on integrity, we find that the SELinux LSM policy can be reduced dramatically, by over 90% in size thusfar, although we believe that much greater reductions are possible. We also show that the resultant policy is suitable for justifying the integrity of such critical applications to remote parties using the PRIMA integrity measurement architecture [13]. PRIMA measures the trusted code and the information flows generated by the MAC policy to ensure that the integrity of the trusted code is protected from low integrity inputs according to the CW-Lite integrity policy. We envision that this approach can provide an outline for how to build high integrity phone systems in the future.

The structure of the paper is as follows. In Section 2, we review the background of phone systems, SELinux, formal integrity models, and integrity measurement that form the basis for this work. In Section 3, we define the phone system architecture, outline our policy design goals, and show that these goals satisfy integrity requirements while permitting the necessary function. In Section 4, we describe the implementation of our system on an evaluation board using to prototype phone software. We show how our policies are implemented, and how integrity measurements are generated for this system. We also provide results showing the performance of the system, when performing integrity measurement. In Section 5, we specify other related work, and we conclude with Section 6.

2. BACKGROUND

In this section, we provide background for phone systems security, SELinux, integrity models, and integrity measurement approaches that motivate our work.

2.1 Mobile Phone Security

Historically, mobile phone systems have been standalone devices with custom operating systems. These consumer electronics devices were installed with software in the factory and no user interfaces were provided for typical users to update the software.

As more functional, "Smart" phones began to appear, the operating system functionality requirements increased. A consortium of phone manufacturers created the Symbian operating system [27], a general-purpose, embedded operating system targeted specifically at the phone market.

The Symbian operating system is most noteworthy for not having a known kernel compromise in its history, but it also implements an interesting security model. The Symbian system defines three distinct subjects: the installer, Symbian-signed subjects, and untrusted subjects [28]. Each process is assigned to one of these three subjects depending upon which of the three categories the originating program file belongs. The three subjects essentially form a Biba hierarchy with installer being the highest integrity level. However, the choice of how files are assigned to integrity-levels is somewhat ambiguous. For example, some system files, such as the Bluetooth pairing database can be modified by untrusted code, permitting untrusted devices to upload files unbeknownst to the user [24]. Although we like the small number of subjects, the integrity protections provided are insufficient.

Recently, Windows and Linux-based phone systems have begun to emerge, eating into the Symbian market share, although it is still the operating system in over 50% of the phone devices sold. Windows and Linux systems bring both applications and security issues to the phone market. Security in the initial versions of these phones was nearly nonexistent. For early Linux phones, if an attacker could get a user to download her malware to the phone, it would be trivially compromised. But, most modern phones provide users with easy mechanisms to upload new programs. As a result, many phone system vendors are seeing that they need to add security enforcement. Motorola Linux phones, such as the A1200, include a mandatory access control module called MotoAC [19] and Samsung Research has explored SELinux on phones [35].

The challenge for phone security is becoming similar to the personal computer. Do the phone system vendors provide so much flexibility that the phones become impossible to manage? Or can a model of security that permits the secure use of untrusted code be created? We explore the answers to these questions in this paper.

2.2 SELinux

SELinux is a reference monitor for the Linux operating system [22]. SELinux enforces a mandatory access control policy based on an extended Type Enforcement model [3]. The traditional TE model has subject types (e.g., processes) and object types (e.g., sockets), and access control is represented by the permissions of the subject types to the object types. All objects are labeled with a type. All objects are an instance of a particular class (i.e., data type) which has its own set of operations. A permission associates a type, a class, and an operation set (a subset of the class's operations). Permissions are assigned to subject types using an `allow` statement.

SELinux also permits domain transitions that allow a pro-

cess to change its label (e.g., when it executes a new program). Domain transitions are important because an unprivileged program could not invoke a privileged program without such transitions. For example, `passwd` would not be able to change a user's password in the `/etc/shadow` file when called from a user's shell unless a transition permitted `passwd` to invoke its own rights. Domain transitions are also relevant to security because a privileged program that does not protect itself from invocations by untrusted subjects will be a security liability to the system. In SELinux, a subject type must have a `transition` permission to the resultant subject type in order to effect a domain transition.

SELinux provides a fine-grained model in which virtually any policy could be defined. As a result, we believe that the SELinux model can be used to implement a policy that we can use to verify the integrity of critical phone applications. However, the development of SELinux policies to date have focused on defining least privilege permissions to contain services. Also, SELinux policies have grown to be very complex. A typical SELinux policy is approximately 3MB in size containing over 2000 types and between 50,000 to 100,000 permission assignments. While there have been efforts to shrink the SELinux policy, we believe that a different view of policy and function is necessary for the phone system. If we can get a simple SELinux policy that provides effective functionionality, then we might get a handle on security before the phone systems get out of control. We believe that to do this we need to focus on the integrity protection of critical applications.

2.3 Integrity Models

Protecting the integrity of critical system applications has always been a goal of security practitioners. However, the integrity models that have been proposed over the years seem not to match the practical environment. Our challenge in the development of phone system policies is to find a practical integrity model.

The Biba integrity model [15] assigns integrity labels to processes and relates these labels in an integrity lattice. Biba integrity requires that normal processes not read data at labels of lower integrity in the lattice. Also, Biba integrity does not permit normal processes to write data at labels of higher integrity in the lattice. As such, no lower integrity data could reach our critical, high integrity application in a Biba system. Unfortunately, many critical applications, including software installers, read some low integrity data.

Efforts to allow processes to read lower integrity data without compromising their integrity have not found acceptance either. LOMAC [8] requires that a process drop its integrity level to that of the lowest integrity data it reads, but some critical phone processes, such as the telephony servers, must be permitted to accept commands from low integrity subjects, but execute at high integrity. In general, we find LOMAC too restrictive, although we implement a variant of it for software installers (see Section 4.2). Clark-Wilson integrity [6] provides a more flexible alternative, by permitting subjects to read low integrity data if the immediately discard or upgrade the data, but Clark-Wilson requires full formal assurance of such processes.

We have previously proposed a compromise approach to integrity, called the *CW-Lite* integrity model. CW-Lite is weaker than Clark-Wilson in that it doesn't require full formal assurance, but CW-Lite requires processes to have filtering interfaces that immediately upgrade or discard low integrity data as Clark-Wilson prescribes. The focus then moves to identifying where low integrity data may be read and ensuring that programs use filtering interfaces to read such data. We aim to apply this view of integrity to phone systems.

2.4 Integrity Measurement

Given the inherently untrustworthy nature of remote parties, it is desirable to be able to validate that a system is of high integrity. More specifically, there should be some guarantee that the remote machine is only running programs that are trusted to behave properly and that the security policy is correct. A proposed method of establishing these guarantees uses integrity measurement [26, 17, 10, 5, 12]. Integrity measurements consist of cryptographic hashes that uniquely identify the components that define system integrity (i.e., code and data). Remote parties verify the integrity of a system by verifying that the integrity measurements taken are consistent with the remote party's view of integrity. Such measurements are conveyed in a messages signed by an authority trusted to collect the measurement, and a signed integrity measurement is called an *attestation*.

The secure storage and reporting of these measurements are typically reliant upon a root of trust in hardware like the Trusted Computing Group's Trusted Platform Module (TPM) [32]. This commodity cryptographic co-processor has facilities storing hash chains in a tamper-evident fashion. It can also securely generate public key pairs that are used to sign attestations and identify itself to remote parties. Samsung demonstrated a phone with a hardware TPM, called the Mobile Trusted Module [31], at the CES conference in Las Vegas in January 2008 [1].

Several architectures exist to gather integrity measurements such as the Linux Integrity Architecture (IMA) [10]. It obtains run-time integrity measurements of all code that is memory-mapped as executable. This facilitates the detection of any malware present on a system.

However, the IMA approach is too simplistic for phone systems for two reasons. First, if any untrusted code is run on the phone system, such as a third-party game, then an IMA verification will result in the entire phone being untrusted. Second, if an attack can modify a data file used by a trusted process, then the remote party may be tricked into thinking that a compromised phone is high integrity because IMA only measures the code and static data files. We aim to enable a phone system to run some untrusted code as long as the MAC policy enables verification that the trusted code is protected from inputs from such untrusted code.

2.5 PRIMA

The Policy-Reduced Integrity Measurement Architecture (PRIMA) [13] addresses the problem of run-time integrity measurements by additionally measuring the implied information flows between processes from the system's security policy. This way, a verifier can prove that trusted components in the system are isolated from untrusted and potentially harmful inputs. Moreover, PRIMA's CW-Lite integrity enforcement model only requires the trusted portions of a system to be measured and thus reduces the number of measurements required to verify a system.

In addition to the basic integrity measurements of code and static data, we identify the following set of measure-

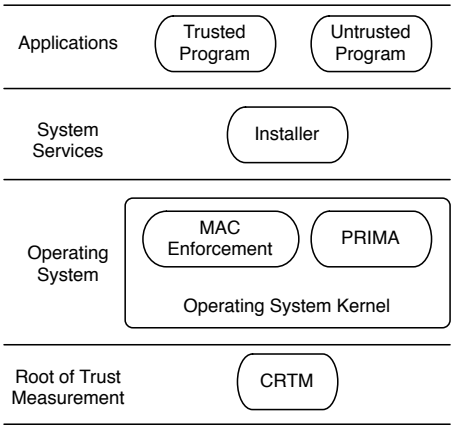

Figure 1: Software architecture for a phone system

ments necessary for a remote party to verify CW-Lite integrity:

1. **MAC Policy**: The mandatory access control (MAC) policy determines the system information flows.
2. **Trusted Subjects**: The set of *trusted* subjects (TCB) that interact with the target application is measured. The remote party must agree that this set contains only subjects that it trusts as well.
3. **Code-Subject Mapping**: For all code measured, record the runtime mapping between the code and the subject type under which it is loaded. For example, `ls` may be run by normal users or trusted administrators; we might want to trust only the output of trusted programs run by trusted users. If the same code is run under two subject types, then we take two measurements, but subsequent loads under a previously-used subject type are not re-measured.

At system startup, the MAC policy and the set of trusted subjects is measured. From these, the remote party constructs an information flow graph. The remote party can verify that all edges into the target and trusted applications are either from trusted subjects (that are verified at runtime only to run trusted code) or from untrusted subjects via filtering interfaces (recall that we extended the MAC system to include interface-level permissions).

Next, we measure the runtime information. Due to the information flow graph, we only need to measure the code that we depend on (i.e., trusted subjects' code). All others are assumed untrusted anyway. Also, we measure the mapping between the code loaded and the trusted subject in which the code is loaded, so the remote party can verify that the expected code is executed for the subject. This is analogous to measuring the UID a program runs as in traditional UNIX.

By measuring how the code maps to system subjects, PRIMA enables a remote party to verify that the system runs high integrity code, perhaps with acceptable filtering interfaces, in its trusted subjects, and that these subjects are protected from information flows from untrusted subjects by the MAC policy.

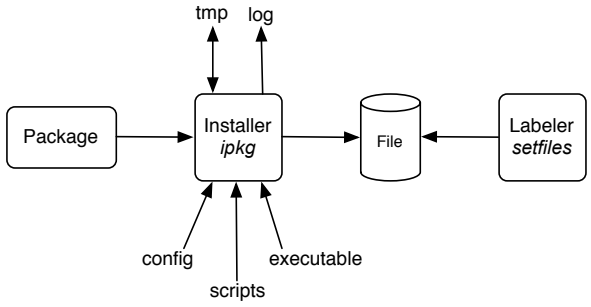

Figure 2: The software installation process

3. APPROACH

3.1 System Architecture

Figure 1 shows the software architecture for a phone system. First, the phone system contains some root of trust which is the basis for integrity in a phone system. An example is the Mobile Trusted Module [31] proposed by the Trusted Computing Group.

Second, the phone system has an operating system kernel that supports mandatory access control (MAC) and integrity measurement using the PRIMA approach. The MAC policy of the kernel will be used to define the system's information flows. The MAC policy is enforced by a reference monitor in the kernel that mediates all the security-sensitive operations of all the user-level programs. SELinux is an example of a kernel with such MAC enforcement (see Section 2.2). The PRIMA module measures the information flows in this MAC policy as well as the code that is loaded in the system, as described in Section 2.5, to enable verification that the trusted subjects are protected from untrusted subjects.

Third, the phone system has a software installer for installing both trusted and untrusted software packages. Most phone systems permit the phone users to install new software packages. In many cases, such installations require confirmation from the device user, but that is not always the case. Also, some software packages may include signed hash files that enable verification of the originator of package and the integrity of its contents.

Fourth, the packages loaded on a phone system may include trusted packages, such as a banking client, and untrusted packages, such as a game program. While some phone systems only permit the installation of signed packages from trusted authorities (e.g., Symbian-signed packages [28]), we envision that ultimately phone systems will also have to support the use of arbitrary packages. However, the trusted components of the system, such as the banking client and the installer itself, must be provably protected from such software.

3.2 Software Installation and Execution

Figure 2 shows the process of software installation. A software installer is a program that takes a software package consisting of several files and installs these files into the appropriate location in the phone's file system. Since the software installer may update virtually any program on the phone system, it is entrusted with write access over all soft-

ware in the system. As a result, the integrity of the system is dependent on the integrity of the software installer.

The software installation process also determines the labels of the installed files. Typically, this is not done by the software installer, however, but a MAC labeling service, outside the kernel, that labels the files based on a specification in the MAC policy. In SELinux, a program called `setfiles` interprets the MAC policy specification for file labeling to set the correct labeling for the newly-installed package files. The MAC labeling service must also be trusted, but unlike the software installer, it need not interact with any untrusted package files directly.

When the software installer executes, its executable uses information in a variety of other files to implement installation. Such files may include installer configurations, scripts, logs, and temporary files. Installation configurations, scripts, and the installer executable itself are rarely modified (e.g., only on installer upgrades), so these can be assumed not to be written on the loading of untrusted software. Other files, such as logs and temporary files may be updated on each installation. That is certainly the intent of the log file, which is designed to collect information from each installation. Temporary files may or may not be used depending on the installation process. In designing an access control policy, in the next section, we must consider the use of these files in designing policies that protect system integrity properly.

After the software packages are installed, the programs included in these packages may now be executed (i.e., as processes). In order to protect the integrity of the system, trusted processes, such as the banking client, must be protected from untrusted processes, such as the game program. As we identified in the PRIMA background in Section 2.5, a process's integrity depends on its code and the data that it depends on. The banking client should be isolated from untrusted programs, so it should not depend on data that can be modified by untrusted processes. However, the installer clearly receives input from untrusted processes (e.g., the untrusted programs themselves) which is necessary for correct functioning. Thus, integrity must be justified while allowing some access to untrusted data, but we also want to minimize the amount of untrusted data that installers must access.

3.3 System MAC Policy Design

The system MAC policy must enable practical justification of integrity for the software installer and the trusted packages that it installs. Here, we sketch the requirements for the MAC policies for trusted programs and the installer. The actual policies are defined in Section 4.

For trusted packages, such as the banking client, we believe that a conservative model of integrity is practical. Biba integrity [15] (see Section 2.3) can be enforced for the banking client because its files can be isolated from all untrusted programs. Since there is only one user on the phone system, there is no need to have separate principals for different banking client data files. We envision that many trusted programs, such as those used to maintain phone books, service configurations, etc., will be isolated from untrusted programs, and generally one another.

For the installer, isolation from untrusted programs is not possible. As a result, only the more liberal justification of

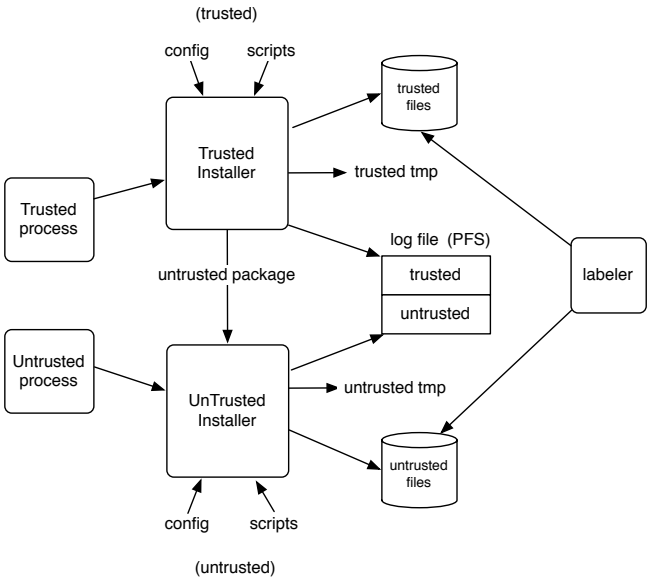

Figure 3: The modified installer process showing filtering interface to handle untrusted input

CW-Lite integrity [25] (see Section 2.3) is possible[1]. In addition, in the design of MAC policy for the installer, we also wish to minimize exposure to the *confused deputy problem* [9] as well. As a result, our software installer runs with permissions that permit any program to invoke it and permissions that permit it to access a package file anywhere in the phone's file system, but the installer's permissions are dropped based on the label of the package that it will install.

Figure 3 shows the modified installer process and outlines the installer's MAC policy. The installer must provide a filtering interface that protects it from compromise on invocation. Thus, the installer's integrity will not be compromised by either a malicious invocation by an untrusted process or an invocation that includes a malformed package. The installer immediately determines whether it is installing a trusted or untrusted package, and drops privileges to that label. This prevents the confused deputy problem by not allowing the installer to use its trusted privileges when installing untrusted software.

3.4 System Security

In this section, we show informally that the MAC policy described above will enable verification of the integrity of the trusted programs and the installer and these policies will provide the necessary permissions for trusted programs and the installer to function properly.

Biba Integrity.

For trusted programs, the MAC policy aims to ensure isolation from untrusted programs. Isolation from other trusted programs may be desirable for *least privilege* per-

[1]We find that other system services on the phone, such as the baseband processor daemon that enables phone calls, SMS, etc., can also be invoked by untrusted programs, so these will also achieve CW-Lite integrity at best. A similar approach would be used for securing them, but their examination is outside the scope of this paper.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.