**The Pennsylvania State University**
**The Graduate School**
**Department of Computer Science and Engineering**

# DYNAMIC MANDATORY ACCESS CONTROL FOR MULTIPLE STAKEHOLDERS

A Thesis in
Computer Science and Engineering
by
Vikhyath Rao



Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2009

The thesis of Vikhyath Rao was reviewed and approved* by the following:

Trent Jaeger
Associate Professor of Computer Science and Engineering
Thesis Advisor, Co-Director: Systems and Internet Infrastructure Security (SIIS)

Patrick D. McDaniel
Associate Professor of Computer Science and Engineering
Co-Director: Systems and Internet Infrastructure Security (SIIS)

Raj Acharya
Professor of Computer Science and Engineering
Department Head, Director: Advanced Laboratory for Information Systems and Analysis (ALISA)

*Signatures are on file in the Graduate School.

# Abstract

*In this thesis, we present a mandatory access control system that uses input from multiple stakeholders to compose policies based on runtime information. In the emerging open cell phone system environment, many devices run software whose access permissions depends on multiple stakeholders, such as the device owner, the service provider, the application owner, etc. However, current access control administration remains as either mandatory, requiring a single system administrator to know every possible permission, or discretionary, allowing possibly compromised processes to administer permissions. A key problem is that the system should limit arbitrary programs while allowing reasonable functionality. However,* conflicting permissions *and permission dependencies may lead to an attack, such as allowing voice-over-IP calls. In our approach, we use a "soft" sandboxing mechanism to first contain such processes, request the stakeholder to authorize operations outside the sandbox that are not prohibited by policy, and maintain a runtime* execution role *for the process to identify its access state to the stakeholders. Our framework was implemented by modifying the SELinux module and using a remote proxy policy server. We incur a 0.288 $\mu$s performance overhead only when stakeholders need to be consulted, and new permissions are cached.*

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to thank my advisor, Dr Trent Jaeger, for providing me with all the advice, guidance and encouragement that I could possibly require during the course of my research work and thesis writing period. I am extremely grateful for his patience and support, without which I could not have been successful.

I am grateful to the Penn State Systems and Internet Infrastructure Security (SIIS) laborotory where I did my research, and in particular to Dr Patrick McDaniel for his invaluable guidance and support during the course of my Master's degree.

I am thankful for all the support I received from the Computer Science department while pursuing my degree.

Finally, I would like to thank my friends and family, in particular my parents Premalatha Rao and Ramraj Rao for their support and encouragement.

# Chapter 1

# Introduction

## 1.1 Area

Security in distributed systems has been an active research problem for the past decade. Objectives of security, specifically access control are difficult to implement due to the de-centralized, dynamic nature of distributed components. One of the largest distributed systems, where we encounter the above problems is the telecommunications network. We concentrate on the specific example of the telecommunications network as our distributed system of choice throughout the thesis, though the concepts and solutions introduced could be applied to other distributed networks.

In just the United States, IEMR forecasts a customer base of 362 million wireless subscribers in 2013 [1]. Apart from the sheer magnitude of users, the growing cellphone network is fast becoming a part of our critical infrastructure as our primary communication device. From basic services like ordinary voice calls and SMS, to time critical emergency services like 911, and business specific services like Mobile phone banking, e-commerce, location based services, users have become highly

dependent on this technology for their communication needs. Even data services like email, web browsing, and instant messaging have seen increase in usage. Further, many subscribers give up their land lines resulting in a single point of failure for their communication needs. We can argue that subscribers are more dependent on the cellular network than ever before. In this scenario any breakage in coverage would be acute, possibly even resulting in life-threatening situations.

Coupled with this growing importance to the telecommunications network, we also notice a systemic change in the operation of the network itself. Recently, we have witnessed a shift from the traditional closed network infrastructure to a more open network model similar to the Internet. One of the primary reasons for this is the evolution of cell phones toward ubiquitous general-purpose computing systems which resulted in the emergence of a wide range of applications. The introduction of these downloadable applications to a formerly closed system results in additional challenges to security as the dynamics of the system change significantly. In the case of personal computers, the primary responsibility of the end-device is left to the user and the network itself belongs to the service provider. In contrast, the telecommunications system is traditionally closely coupled to the network operator. But in the newer, more open telecommunications system, multiple stakeholders wish to retain some control over various aspects of security. Thus, when an unknown application is downloaded, it is imperative that the application fall under the administration of different stakeholders, say the end-user, phone manufacturer, network provider, applications provider etc. Further, since the untrusted application may leverage sensitive permissions, the stakeholders need a way to make runtime administrative decisions to control such software, while providing the desired user experience.

Failure to exercise good access control on these untrusted applications could

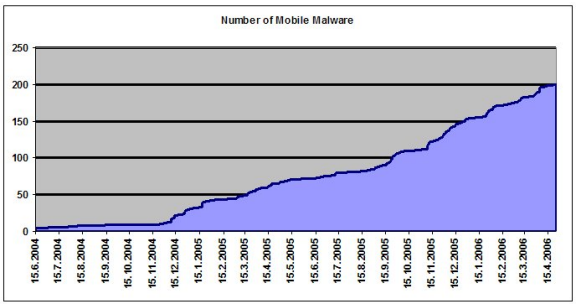


**Figure 1.1.** Problem Overview

result in the spread of viruses and worms, not unlike what was witnessed in the nascent PC industry. Already we see many instances of worms and viruses for popular phone operating systems like Symbian. Since the first Symbian virus Cabir that replicated via Bluetooth (July 2004), the growth in number of viruses discovered has been almost exponential as show in figure 1.1 from Fsecure Security Research [2].

As mobile phone OSes undergo standardization and clear flavors like Apple iPhone, Blackberry RIM, Google Android, Symbian, and Windows Mobile emerge, we can expect to see more coordinated and dedicated attacks on the network in general, it's subscribers, and their personal information.

## 1.2 Problem

Bringing access control security to a distributed network, especially one as dynamic as the telecommunications network comprises of many difficulties. Examples of some of the challenges associated are: an exhaustive monolithic security policy not always being present on the client; scalability as the number of clients increases; transient permissions due to a highly dynamic environment,; and inability to provide stateful access control instead of a static policy. Further, the

presence of multiple stakeholders and absence of a single undisputed authority in distributed environments adds a new dimension to an already difficult problem. For example, in telecommunications system, policy decisions may be made by the end-user, phone manufacturer, telecommunications provider, or even the application provider and consulting these different stakeholders exacerbates scalability issues.

In general, there are primarily two approaches for administering access control policies, discretionary access control (DAC) and mandatory access control (MAC). DAC permits administration by the owners of system objects, often users. The central problem with DAC administration is that users or processes on behalf of users, accidentally or intentionally, may override important system permissions, thereby compromising security. In MAC, administration is restricted to trusted subjects, such as system administrators, which can ensure that a specific goal is enforced on the system.

While MAC administration appears to provide the control necessary to ensure system security, it is too restrictive for the dynamic environment of these ubiquitous devices. For MAC administration, it is imperative that system administrators define accurate security policies for all possible application deployments and executions, but that is difficult for the following reasons. First, each deployment may have a different set of stakeholders with different security requirements that must be composed. Even the set of stakeholders may not be known until runtime. Second, some applications may want to use some sensitive permissions. While it is easy to deny permissions that lead directly to attack, in many cases it is a combination of mutually conflicting permissions (e.g., access to voice input and the WiFi network that would enable VoIP, circumventing telephony charges) that may lead to attack, rather than a single permission. Third, some applications may be

previously unknown to the stakeholders, so it is necessary to derive permissions on the fly. However, strict sandboxing may be too restrictive, so the mechanism needs to be able to authorize limited permissions, again preventing attack vectors. A traditional MAC policy is inadequate in this case, because it does not capture the requirements of all stakeholders nor express the dynamic requirements for determining policy.

Existing approaches do not balance the security with the dynamic requirements of such systems accounting for governance by multiple stakeholders. In phone systems, Google Android uses static manifests to load the policy for new applications at install time [3], and Symbian uses certificate-based permissions [4]. These approaches assume that the policy is sufficiently static to be pushed to the client at install time and importantly, that the program is already known to a particular stakeholder responsible for the system security, and finally, only static policy needs to be enforced by a Client Reference Monitor (CRM). Traditional MAC systems presume either that all programs are known ahead of time (e.g., SELinux [5]) or that the mapping between users and labels is determined at login (e.g., MLS [6] and RBAC [7]). None of these systems support the download of programs that may need sensitive permissions whose use may preclude other permissions (e.g., Chinese Wall [8]) nor do these approaches support the input of multiple stakeholders.

## 1.3 Approach

Our approach implements dynamic administrative decisions in a distributed environment consisting of multiple stakeholders. For each application process, we provide a base policy that only provides access to those operations granted for all runs. For previously unknown applications, this policy provides a sandbox. When

permission is requested that is not authorized by the base policy, but not strictly prohibited, a policy server is consulted to determine whether this operation should be authorized based on the input of the application's stakeholders. The policy server implements a hierarchical mechanism to determine the application's stakeholders and their administrative decisions regarding this request. We say that the mechanism is hierarchical because the stakeholder's decisions may be cached at the policy server on the device, a proxy policy server (e.g., in the telecommunications network), or on the stakeholders themselves. In best case, the stakeholders are already known and the administrative policy is already cached on the device, so it is a matter of updating the MAC policy policy. A variety of policies are permitted for combining stakeholder decisions. At present, we only consider a Chinese Wall-style selection of one set of permissions from a conflict set, although others are possible. We support both transient updates (e.g., limited use or temporary permissions) and persistent updates (e.g., policy modules in SELinux).

## 1.4 Contributions

In Chapter 3, we briefly describe a novel Denial of Service (DoS) attack that could cause significant service degradation and overwhelm the core telecommunication network. This attack serves to highlight the vulnerability of the cellular network as it transitions toward a more open system. Our major contribution in the rest of the thesis is an administrative policy mechanism for MAC in dynamic environments with support for multiple stakeholders. This mechanism provides the functionality described above, as well as supporting revocation, since we believe in any dynamic environment, policy may also be removed. A key insight in implementing such a mechanism is that it will not be possible to push the state of each devices MAC

policy around the network (e.g., to the proxy or stakeholders), so we identify that subjects correspond to dynamic "roles" that, like typical RBAC, imply that the subject possesses a set of permissions, but unlike RBAC, is used by policy servers and stakeholder to make administrative decisions. We have implemented our administrative mechanism and find that the performance overhead for each access request is minimal at 0.288 $\mu$ s. Of course, there may be additional overhead to talk to the proxy server and stakeholders. We expect that such delays will be similar to registering the device on the network, and much administrative policy may be captured in that task. Our framework can be implemented in any platform, but we chose Linux because it is open source, and has potential for increased market share. Linux market share is already substantial at 8.1% [9] and by estimates, poised for growth to atleast 23% by 2013 [10, 11] due to new platforms like Google Android, and LiMo initiative.

## 1.5 Thesis Outline

The rest of this thesis is structured as follows. Other existing approaches with regards to the enveloping skeleton of Usage CONtrol (UCON) is presented in related work in Chapter 2. Chapter 3 discusses the novel DoS attack that is a significant threat to the cellular network. In Chapter 4 we define the problem and challenges our framework is trying to solve. This is followed by an explanation of our solution in Chapter 5. Implementation details of our framework are discussed in Chapter 6. We also present some additional features of our framework, and deal with the performance and overhead involved in our solution. We conclude the thesis with Chapter 7 which also summarizes our future work.

# Chapter 2

# Related Work

In this chapter we review the body of literary work that is related to the thesis. This includes other approaches to access control and dealing with multiple stakeholders, providing Usage CONtrol for applications, and policy composition techniques for distributed systems.

## 2.1 Multiple Stakeholders and Access Control

Certificate based techniques on the topic of multiple stakeholders and distributed policy have been explored in Akenti, however the authors deal more with access to distributed resources, which is different from our goal of a tightly coupled local CRM enforcing policy provided by multiple stakeholders [12]. Tresys has implemented an SELinux policy server that allows remote administration of SELinux policies, however, it is limited to single administrator, pre-computed policy modules that allow remote installation [13]. Static manifests techniques like kirin are not suitable for scenarios where the manifest itself is highly dynamic [3]. Certificate based techniques have been already used in practical systems like Symbian

[4], but have weaknesses similar to static manifests. Furthermore, external certification leads to increased costs for third party developers, as they must get their applications certified for a fee, in case of any modification.

Tools like audit2allow, have previously used the set of all denied messages obtained from an installed application to identify the permissions it needed [13]. However this was performed off-line using log files to generate new policy files which allow the application to operate. This approach is not dynamic since we would need to obtain the appropriate log files, and run the tool each time to generate the new policy. Further, the newly generated policy needs to be compiled and inserted into the kernel. Finally, we would still need to identify the permissions that should be allowed or denied, since we cannot perform a blanket guarantee on the entire log file and allow all requests.

## 2.2 Policy Composition

Policy composition is the consolidation of access control restrictions from a combination of different policy specifications, each possibly under the control of independent authorities (stakeholders). Inherent complications arise when the specifics of some component policies may not be known apriori. Turning individual specifications into a coherent policy to be fed into the access control system requires a non-trivial combination and translation process.

Separation of policy from mechanism in distributed systems was discussed by Sloman et al based on two ESPRIT funded projects, SysMan and IDSM [14]. The paper also contained concepts of authorization, obligations and constraints. These principles are expanded in the UCON framework in the section 2.3 below. Later work by Sloman et al also focused on conflicts in policy based distributed systems

[15]. However, these works dealt with policy management at a conceptual level rather than enforcement and associated framework.

In this thesis, we leverage some of the seminal research in policy composition performed by Bonatti et al [16]. Their work deals with a proposed algebra of security policies together with its formal semantics, and illustrates how to formulate complex policies. We detail the algebraic constructs and utilize them for our policy composition. In our framework incompletely specified and piecemeal policies from the stakeholders are consolidated into a monolithic policy that is presented to the end device. Related work in this domain includes implementing policy composition in the XACML policy language [17].

## 2.3 UCON

Usage control (UCON) which is a generalization of traditional access controls, trust management, and digital rights management is a systematic approach for next generation access controls [18]. Traditional access control is limited to a closed system with a server-side reference monitor where all the users are known. Instead, UCON can consider previously unknown entities, establishing their access rights via the server-side reference monitor (SRM), while usage control (access control and resource management) is performed using a client-side reference monitor (CRM).

The UCON model includes obligations and conditions as well as authorizations as part of usage decision process to provide a richer and finer decision capability. Authorizations use subject attributes and object attributes to create a decision process based on permitting or denying access. Obligations are requirements that have to be fulfilled for usage allowance. Conditions are environmental or system requirements that are independent from individual subjects and objects. These

are decision predicates that need to be evaluated and can be used to constrain any access request.

Our work develops a multi-stakeholder form of UCON, although our approach enforces access only prior to use, limited by existing reference monitor implementations. We use SELinux as a standard CRM, but we build a layered SRM framework to reason about access decisions from multiple stakeholders and cache those decisions for efficient update and revocation. Although we provide some resource control, such as limiting the number of execution attempts fr a permission, our main focus in this thesis is restricted to access control: determining the rights of an application based on the input of multiple stakeholders. Nonetheless, we believe that our framework is flexible enough to incorporate more usage control, and increase the amount of state involved in policy decisions in the future.

## 2.4 TRBAC

Access control constrained by time of access has been studied extensively in Temporal Role Based Access Control (TRBAC) [19]. Here access control is defined by roles, and access to roles is limited to certain time periods. Through periodic role enabling and disabling, access control is associated with role triggers that constrain access to limited time periods. However, TRBAC requires support for RBAC in the framework, and this is not always feasible if the implementations are limited to Type Enforcement (TE), for example, in SELinux. Our implementation does not require RBAC support, and additionally supports spatial constraints, with flexibility to add new constraints as desired.

## 2.5 Other Approaches to Security Policy

### 2.5.1 PinUP

An interesting approach to implementing security policy is the PinUP framework introduced by Enck et al [20]. In their paper they use the concept of associating files to the applications that use them. Once adhered to their respective applications, this framework uses protection mechanisms that identify access policy as positive application-aware work flows. However as the authors noted, Pinup is intended to augment existing security servics within the OS and is specifically suited to protecting user files rather than system files. It is designed to operate above MAC systems such as SELinux and AppArmor. These characteristics make Pinup largely complementary to our framework rather than being a replacement system.

### 2.5.2 Google Android

The Android security model has many features that make it antithetical to the Symbian certificate model discussed previously [21]. These include ability for third party developers to self-sign their applications, a marketplace where anyone can publish, and easy enabling of non-market apps. The most important difference is that application permissions are defined explicitly by the developer and approved by the user at install. This is in complete contrast to the Symbian model where the permissions are requested by the developers and then granted by the Symbian Signed organization after being manually vetted. In comparison to these approaches, our framework allows a flexible selection of stakeholders. This means the user, the apps developer or even a formal organization like Symbian Signed can be represented as stakeholders in our system. Hence we can think of our frame-

work as the superclass of these individual instantiations. Further, we derive the permissions the application requires at runtime automatically rather than relying on the application developer to identify them. Other important characteristics of the Android security model is that each application is sandboxed with its own process, and that all permissions are verified and stored at install time. However, this does not allow flexibility for dynamic changes in permissions, and support for stateful constraints like our security framework.

### 2.5.3 DIFC systems (Flume and HiStar)

Decentralized Information Flow Control (DIFC) definition is an approach to security that allows application writers to control how data flows between the pieces of an application and the outside world. DIFC systems enable the management of application privileges at a finer granularity. These are implemented by attaching DIFC semantics to each file as metadata. Once associated, the system can permit, prevent information flow. Two variants of implementing DIFC are Flume [22] and HiStar [23]. Both provide per-process and per-file labeling to permit only safe flows in the system. Flume uses separate integrity and secrecy labels. HiStar uses a scheme that integrates integrity and secrecy into a single label. However, DIFC systems are more coarse grained, and one need to fragment an application into smaller processes in order to be effective. This often requires writing wrappers around existing applications which results in challenges in adoption to a dynamic environment. Such a environment is possible when the permissions of the applications may themselves change in the event of revocations. Further these systems were not intended to be used with multiple stakeholders and are localized to the client device. Finally, we can also modify our framework to transmit and main-

tain DIFC metadata and enforce DIFC policy in our reference monitor and use it for multiple stakeholders and allow dynamic changes in policy showing that these works are complementary and are focus on solving different problems.

# Chapter 3

# Telecommunication DoS

Telecommunications networks have become more susceptible to core infrastructure attacks than ever before [24, 25, 26, 27]. However directed attacks targeting critical components are a newer phenomenon. In this chapter we discuss a novel DoS attack that targets the main registry database in a cellular network. There are two reasons for the attack: Bottleneck in the core network, and easily compromised end devices that allow the attack to take place. While fixing the core bottleneck problem is one of the telecommunications network, we focus on the handset devices. Hence we demonstrate a need for increased handset security. We conclude the chapter by quoting the end-to-end principle which agrees with our supposition that to prevent such attacks the end points of the network, namely the cell phones, need to be fortified.

## 3.1 Overview of Cellular Systems

In this section we provide a brief overview of cellular phone system, and the responsibilities of each component, specifically in the core network. Figure 3.1 is a

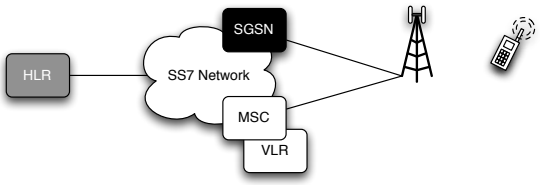


**Figure 3.1.** Cellular Systems Overview

simplified representation of a cellular network.

The Home Location Register (HLR) is a fundamental database that is considered the heart of a cellular network. The HLR performs a variety of services ranging from user authentication, call-forwarding, and billing. It is also essential for basic phone calls in the following way. The network assigns users to specific HLRs based on their phone numbers. When a subscriber attempts to call a user, the caller's switch queries the appropriate HLR with a request for the targeted user's current location. Only with this current location can the subscriber reach the targeted user in the network. Thus the primary application of the HLR is to serve as a central repository of user profile data. Accordingly, it is crucial in providing service in a network with mobile endpoints. The next component is the Mobile Switching Center (MSC). In brief, the goal of the MSC is to act as a telephony switch and deliver circuit-switched traffic in a GSM network. Apart from this basic goal, the MSC performs a magnitude of functions. They are responsible for performing handoffs between base stations, assist in billing operations and can function as gateways to both wired and neighboring cellular systems. With the assistance of a Visiting Location Register (VLR), MSCs can identify and store information about currently associated subscribers. However, information requests for other subscribers still require interaction with the HLR. For data connections, these operations are performed by the Serving GPRS Support Node (SGSN). Mo-

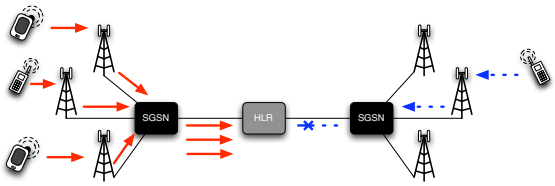


**Figure 3.2.** Attack Overview

bile phones and other cellular-enabled devices connect wirelessly to the network through the Base Station Subsystem (BSS). These units provide the logic for operations such as wireless resource management, encryption and frequency hopping.

## 3.2 Attack Overview

The basis of the attack is twofold. First, we must compromise the end devices with which we attempt to overwhelm the network. This is not inconceivable as phones transition to newer operating systems, and have not been through a vetting process similar to the personal computers on the Internet. Mobile phone security is still very much in its infancy. With the compromised phones, We attempt a targeted Denial of Service (DoS) attack in order to prevent legitimate users of a cellular network from sending or receiving calls or text messages. Instead of attempting an undirected DoS attack, our attack involves selectively targeting the HLR. This is because most of the functionality of the networks relies on the availability and proper functioning of HLRs. Legitimate users relying on the same HLR will be unable to receive service as their requests will be dropped. Figure 3.2 shows how such an attack would occur.

**Table 3.1.** Transaction Mix

| Transaction | Type | Default Mix |
|---|---|---|
| `get_subscriber_data` | r | 35% |
| `get_new_destination` | r | 10% |
| `get_access_data` | r | 35% |
| `update_subscriber_data` | w | 2% |
| `update_location` | w | 14% |
| `insert_call_forwarding` | rw | 2% |
| `delete_call_forwarding` | rw | 2% |

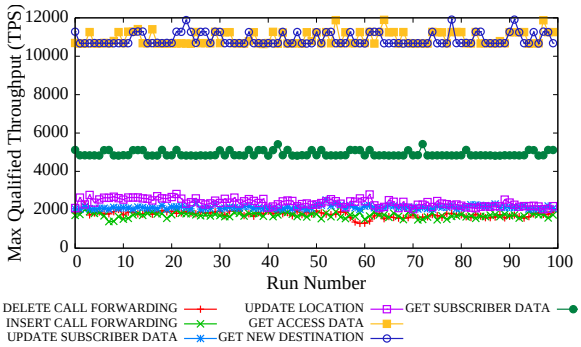


**Figure 3.3.** HLR throughput for each transaction type with 500K subscribers

### 3.2.1 HLR Performance

While the attack seems straightforward, there are a number of details that need to be focused on. Firstly, not all network messages from the compromised phones reach the HLR. Most network messages are screened or dealt with only at the basestation or MSC. Further, among the messages that reach the HLR, we need to identify those utilize maximum resources require minimum execution time. We measured network timings by programming a phone to send different types of messages to the HLR. The various messages and their probabilities in a normal scenario are shown in table 3.1. Our attack involves increasing the rate of those network messages which utilize maximum resources so as to overwhelm the HLR.

The HLR is a transactional database and the commands can be broadly classified as 80% read and 20% write. This allows the throughput to remain fairly high

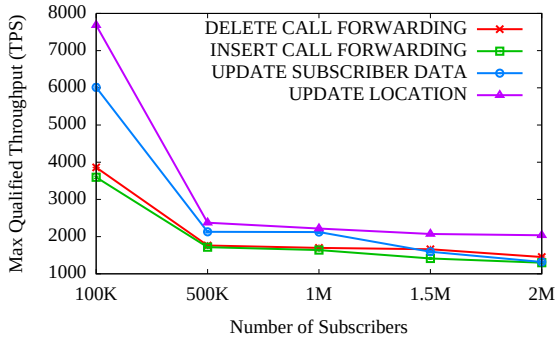


**Figure 3.4.** Comparison of candidate transaction types based on the resulting throughput in a MySQL database

as read commands utilize less resources. Hence in our attack, this immediately eliminates the read commands from the above list since we want to concentrate on those commands that utilize maximum resources. Figure 3.3 below shows the throughput for the various commands running in default mix. From this figure we can isolate the four database write commands as possible threats. Namely, update location, update subscriber data, insert call forwarding, and delete call forwarding.

Correspondingly Figure 3.4 shows how the transactions per second for these meta-commands drop, as the number of subscribers in the network increases. This typifies the bottleneck that can be exploited as the number of subscribers increase in the network, or in the case of an attack, where increased network messages are injected into the network.

## 3.3 Attack Characterization

We identify the time required for each meta-command by runnings tests on the live network. We note that at no point did we try to overwhelm the network, and only tested commands from a single phone to measure round trip time (RTT) required. Among the commands, the insert/delete call forwarding meta-commands

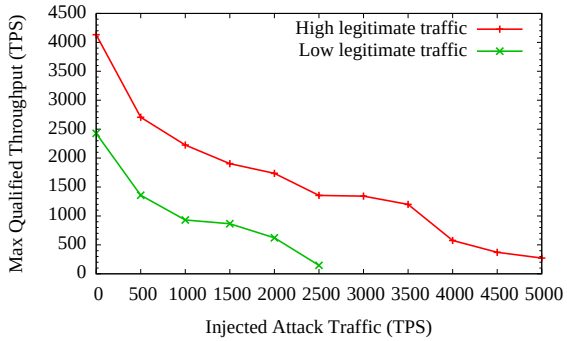


**Figure 3.5.** Attack on HLR running MySQL using default mix of commands.

took on average 3 seconds to complete. The total turn around time for the GRPS attach/detach (update location meta-command) is 5 seconds(3 seconds response time, and 2 second interval). This gives a total output rate of 0.2. However we found that only 1 in 5 messages reach the HLR while the remaining are returned from a cache located at the MSC. This gives a total output rate of 0.04 commands per second. While the resources utilized in execution of this meta-command is significant the slow turnaround time in reaching the HLR limits its usefulness. Finally the update subscriber data (Call Waiting update) meta-command has 2.5 seconds RTT but utilizes less resources than the insert Call Forwarding meta-command (2.7 seconds rtt )and hence is not considered as strong for the targeted attack. More information regarding each meta-command and the timings measured are present in the following work [28]. We do not mention the details here since it is not of direct interest in this thesis.

Thus we identify insert call forwarding as the preferred meta-command for our injection attack to overwhelm the HLR. Figure 3.5 shows the impact of attack traffic on an HLR running MySQL with 1 million users. The lower intensity traffic stream, which achieves a throughput of 2427 default mix TPS, is reduced to a throughput of only 146 default mix TPS at an attack rate of 2500 insert

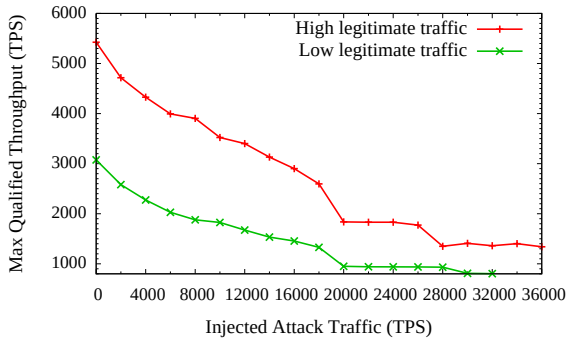


**Figure 3.6.** Attack on HLR running SolidDB using Default Mix of commands.

call forwarding TPS. The high intensity traffic stream, whose throughput is 4132 default mix TPS under normal conditions, is dropped to 273 default mix TPS under a constant attack of 5000 insert call forwarding TPS. For both cases, the throughput of legitimate traffic is reduced by more than 93% by the presence of attack traffic.

The impact of attacks on a more capable HLR supporting one million users is shown in Figure 3.6. While the system running SolidDB is certainly capable of maintaining service during small attacks, the performance of these systems can also be extensively degraded. In the high traffic case, throughput is reduced from 5424 to 1340 default mix TPS at an attack rate of 30,000 insert call forwarding TPS. The lower traffic scenario is similar impacted and experiences a reduction in throughput from 3075 to 803 default mix TPS at an attack rate of 30,000 insert call forwarding TPS. These attacks represent a reduction in throughput of approximately 75% in both cases.

Further extending the calculation, we can identify the number of devices an adversary requires to successfully launch an attack against an HLR. Given the 4.7 second wait between the successful transmission of AT commands in our experiments, an attack on the HLR running the MySQL database under normal

conditions would require approximately 11,750 infected mobile phones. Achieving a similar result against a high traffic period would require 23,500 infected phones.

## 3.4 Conclusion

Thus installation of third party applications and the increased connectivity of data networks has increased the core networks susceptibility. In order to protect the network, we can focus on two approaches. The first deals with the core network itself as the security is reinforced using filters, data replication, and shedding. However, each method is not an end to security in itself. Data replication and load sharing in components like HLR results in increase in traffic at other HLRs. Further, an architecture where the entire traffic is handled on a few highly capable HLRs with high capacity links between HLRs presents vulnerabilities and bottlenecks of its own. Filters and shedding are more effective in attack prevention, but developing mechanisms intelligent enough to respond to a more dynamic attack remains challenging, especially if adversaries deliberately attempt to target nodes other than the HLR. Moreover, because of the large overhead associated with the first hop of communications in such networks, filtering in the core may occur too late to prevent users from experiencing significant congestion. Preventing such devices from ever transmitting malicious traffic is arguably more critical in this environment than the traditional Internet setting. Thus the rest of the thesis focuses on presenting a security framework for mobile phones in order to ensure the end points of the network itself is more secure and leads to increased security against such attacks. This is in agreement with the end-to-end principle which states: *Whenever possible, communications protocol operations should be defined to occur at the end-points of a communications system.* Of course such a framework can be

enhanced by strong security in the core of network as well, but in thesis we focus on security for the device itself.

# Chapter 4

# Problem Definition

Apart from just providing viable host security, there are other requirements in today's telecommunications networks. In this chapter, we clearly define the problem we are trying to solve by using an example scenario involving the cellular network. We then highlight the key requirements for a solution, and finally conclude by outlining the expected operation of an example application in the proposed framework. While we use the telecommunications network as an example to illustrate our framework, we stress that it can be extended to other distributed scenarios easily.

## 4.1 Problem

For our example scenario, we consider the cellular network system whose core network is owned by the network operator, and provides services to millions of distributed clients. Early telecommunications systems did not allow users to install any software obtained without the service providers permission. Even basic features like ring tones could only be downloaded Over-The-Air (OTA). This is

unlike personal computers, where the lines are demarcated more in favor of increased user privileges to the detriment of security. In recent years, however, the telecommunications network has opened up, as consumers demand more services, and advanced cellphones with support for installation of third party applications were introduced. Following an Internet-like model where "anything goes" is not feasible in a closed network like the telecommunication network. Many researchers already warn us about the effects of end-users installing third party applications [29, 30, 31].

Further, the more restrictive the service providers are, the more alienated the users will become, leading to a situation where the users rebel against the very same providers who are trying to protect them and their systems. For example, the initial iPhones were introduced which locked users into a certain service provider. This quickly resulted in mass "Jail-breaking" of these phones [32, 33]. "Jail-breaking" is a term for removing locks that the service providers implement. The users who installed these hacks, many from questionable sources, further exposed themselves and their systems to more threats. This creates a security model where the users are adversaries. This scenario makes using Discretionary Access Control (DAC), the preferred security model for Linux desktops a difficult fit. DAC relies on file owners i.e. users, to propagate permission transitions and putting the responsibility of security solely in the user's hands is both too much responsibility, and too much of a security hazard. MAC is a good solution as it can implement verifiable security, but the demand for increased services and flexibility from consumers, coupled with the dynamic cellular network system of today, means a static MAC policy is not flexible enough.

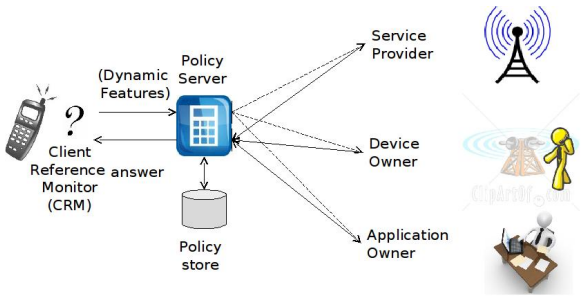


**Figure 4.1.** Solution Requirements

### 4.1.1 Problem Statement

Thus we define our problem statement as follows: *Enabling dynamic policy administration in MAC systems by multiple stakeholders in a distributed environment.*

## 4.2 Framework Requirements

We summarize the framework features we desire as follows

- Support for multiple stakeholders and dynamic policy composition
- Automatic identification of manifest permissions and local state for remote policy decisions.
- Separate handling of high risk, transient permissions from low risk, stable permissions for improved efficiency
- Extensive support for revocation including temporal and spatial constraints, and pay-per-usage functionality.

We use Figure 4.2 and establish the steps required for any effective framework in the following subsections.

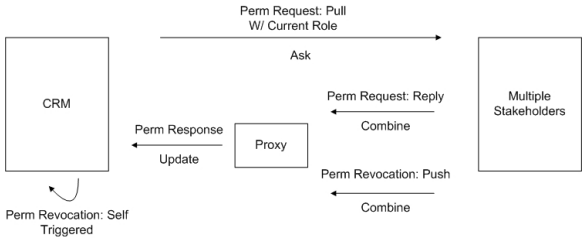


**Figure 4.2.** Problem Overview

### 4.2.1 Identify when to ask

This corresponds to the instances when we submit the permission request message from the Client Reference Monitor (CRM) to the stakeholders. Instances include verifying usage constraints for every resource access, and obtaining conflict sets for the same to prevent allocation of mutually exclusive resources. We also need to obtain additional stateful constraints based on temporal and spatial availability for local enforcement. Trivially, we would submit every access request since we essentially advocate decoupling of policy enforcement and policy to remote server. However such an implementation is impractical as the large number of requests would lead to scalability issues. In our solution we create a practical and scalable framework by using our CRM to enforce policy and identify permission requests dynamically.

### 4.2.2 Identify what to ask

Here we identify the contents of the permission request to the stakeholders. In order to reduce the overhead involved and create an efficient solution we need to minimize the information transmitted . For our framework, the following information needs to be transmitted by the CRM; the access request comprising of subject, object, task being performed, unique application identifier, and state information. State information refers to the set of permissions already existing

on the phone. Maintaining a separate state unique to every set of permissions is unfeasible. Hence, in our framework we define sets of permissions as roles. Possible roles may include, Wi-Fi enabled, GPRS-enabled, handset microphone and speaker-enabled, and GPS-enabled, to name a few. Using these roles, we obtain conflict sets from the stakeholders.

### 4.2.3 Response from Stakeholder

Responses are either replies to permission requests submitted, or permission revocations pushed independently to the device. The stakeholders provide administrative policy that may be cached at the proxy server (in the network) and policy server (on the device). The administrative policy associates applications and roles with rules for administering the MAC policy. The mechanism does not limit the types of rules, but in our initial approach, stakeholders may state no interest or conflict sets among roles (and the permissions associated with those roles). The policy server will find the role associated with a permission request, determine if this role is in conflict with one of the application's current roles, and, if not, return the new role and its specified permissions (including positive and negative permissions). Additional temporal/spatial state information associated with the policy is also returned to the device where it is locally enforced.

### 4.2.4 Response from Proxy/Policy Server

Downloading entire policies for transient requests or updating permission requests incrementally for intransient permissions are both inefficient. We categorize our policy update responses into two possible forms. The first is a transient update that is request-specific. The permissions are added, but they may expire or otherwise

be revoked. This approach is better suited to high risk, downloaded applications with individual permissions that may need to be revoked. The second type updates permissions at the scale of the application or the application-run, where a batch of permissions analogous to a policy module are updated. It is not intended that such permissions be revoked individually or perhaps at all.

### 4.2.5 Update client

Finally, after obtaining the permission response, and verifying current temporal/spatial state of the device against the policy requirements, we either make a transient change or a module change to policy. The roles of the application and its state (temporal/spatial) may be updated locally. Also, by transferring constraints for number of resource accesses, we can perform self triggered local revocations to limit number of executions for an application.

## 4.3 Example Scenario

In the following sections we analyze how an Internet based telephony application similar to Skype, installed on a handset in the telecommunications system, operates in our framework. With the advent of Wi-Fi on handsets, such applications can be used to make free phone calls over the Internet bypassing the telecommunications infrastructure and costs associated with it [34].

The first restriction for such applications comes from the service provider. Network operators do not want to allow end users to make calls through the Internet as this creates a loss of revenue for them. On the other hand, the phone manufacturer would have no problem allowing installation of such applications if its the telecommunications systems, as it increases the phone's appeal to consumers and

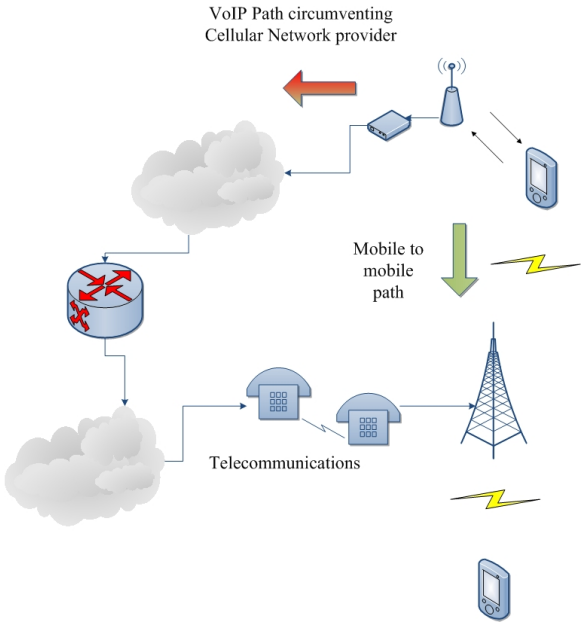


**Figure 4.3.** Telephony Vs VoIP

may result in increased sales. Finally, the application provider might want to limit the user to only 20 executions of the application since it is a trial version and the user has not paid the requisite fee. Here, the phone manufacturer, network operator, applications provider, and the end user are all stakeholders in the system, and have specific access control restrictions that need to be enforced. Even if the service provider stakeholder decides through a change in policy that Internet telephony is allowed, the applications provider would still want to enforce pay-per-use applications or trialware, where application access is limited. In this case, the permission granted to the telephony application must be revoked after the requisite usage. One of the main challenges of policy revocations in distributed environments is the overhead associated with frequent policy downloads. This problem is compounded in an environment where users may install new applications regularly. Hence scalability must be carefully accounted for in any framework solution. Finally, one of our framework requirements is that depending on if the telephony application is

recognized by a stakeholder, or is an unknown third party application, we either obtain an application specific policy, or a request specific response respectively and this affects ease of revocation which ultimately impacts scalability.

# Chapter 5

# Approach

In this chapter we describe our solution to the problem described. Our approach is a security framework that enforces MAC, but with some important changes that allow it to function in a more transient environment. We choose MAC because phones are inherently single user and DAC relies on file owners assigning and transitioning file permissions. Thus in a traditional closed system, like older generation phones, an implementation of MAC with explicit policy defining every subject-object interaction would be the preferred solution. However, in the current generation of phones, that allow third party installation, and transient permissions, we require a more dynamic approach to security. MAC is very rigid in that it requires a static policy that defines every subject-object interaction, and any change in policy requires a completely new policy to be sent and uploaded on the host device. This creates problems with performance and scalability.

We use functional mechanisms like incremental updates for transient permissions, while maintaining the default policy download option to improve scalability. We also support multiple stakeholders and overcome the limitation of a single administrator in conventional MAC. Other features we support include an efficient

revocation implementation. Thus we provide the strong security of a completely specified MAC system, overlayed with framework modifications that allow operation in the transient, dynamic phone systems of today. We first present an architectural overview of our framework, followed by the details of our main functional mechanisms.

## 5.1 Architecture Overview

In the subsection below, we present a brief architectural overview of our framework by highlighting the important structural components. This is followed by a logical interpretation of decision hierarchy in the next subsection.

### 5.1.1 Framework Components

The main components of our framework as shown in Figure 5.1 are the MAC sandbox, local policy server, and Proxy Server. The MAC Sandbox is the Client Reference Monitor (CRM) that is responsible for policy enforcement on the client, and for transmitting permission requests with the information required for various stakeholders to make a decision. By using a CRM to identify the permission requests dynamically, we avoid the problem of requiring an external entity to create a list of static permissions that is required at install time.

The local policy server contains access decisions that are stored locally on the host platform. These stored decisions are used by the CRM to enforce policy locally. The decisions comprise of both the default initial-state policy that the phone is provided with, and cached decisions returned from the stakeholder on the local platform. It additionally enforces temporal and spatial constraints.

The proxy server requests administrative policy from the various stakeholders

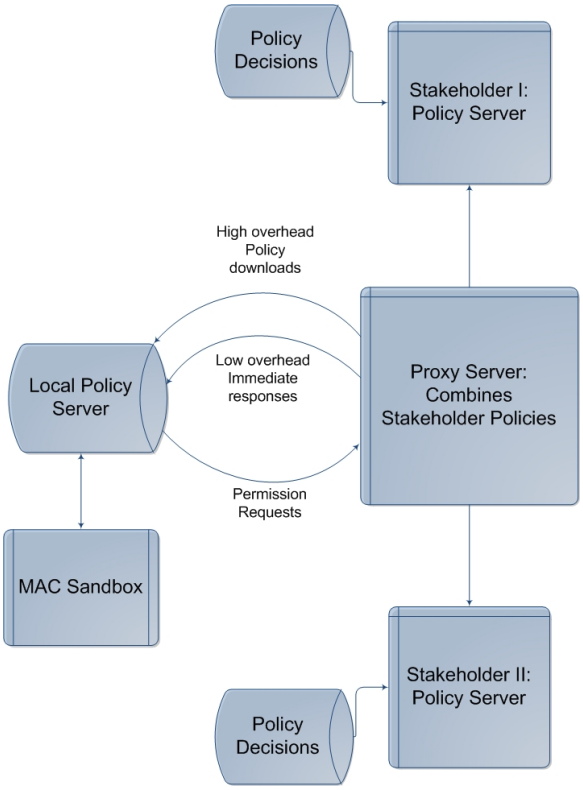


**Figure 5.1.** Framework Overview

and consolidates them. It serves as a staging point for responding to administrative requests and delivering administrative decisions to the local policy server. We detail the functions of these components and discuss its operations in Section 5.2.

### 5.1.2 Logical Hierarchy

Viewed differently, our framework components provides an interesting hierarchical structure as depicted in Figure 5.2. Top to bottom, the responsibilities gradually change from administering the policy to enforcing the decision. The top tier consists of the various stakeholders and their individual policies. These are consolidated and staged at the proxy policy server which acts as a proxy and an abstraction to the lower tiers, simplifying their interaction to a single entity.

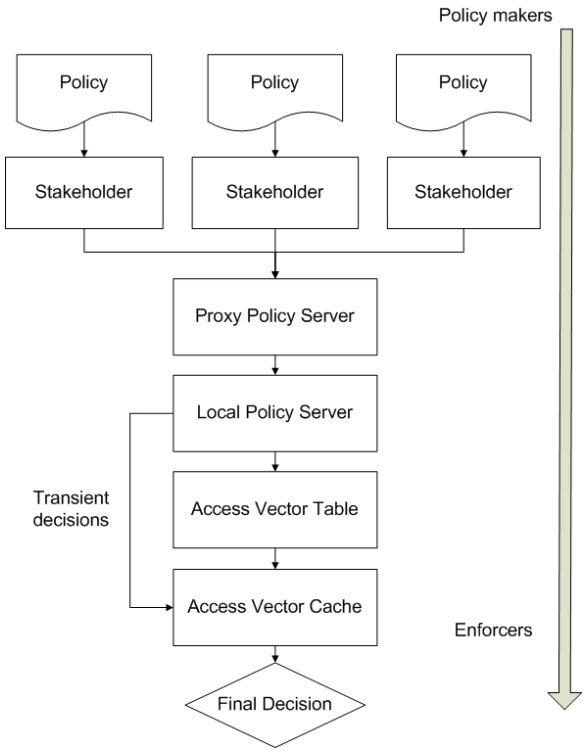


**Figure 5.2.** Hierarchical flow diagram from the policy creators to enforcers

Then, we have the local policy server that processes updates to the reference monitor's policy storage with permanent policy changes (in the *access vector table*) and transient policy changes (in the *access vector cache*). We describe the value of transient permissions in the next section, as these enable faster updates and permit incremental revocation (e.g., for resource management).

## 5.2 Functional Mechanisms

In this section, we highlight the functional mechanisms that we use to develop our framework.

### 5.2.1 Subspace Management

To deal with the large number of permission requests that occur if all requests are submitted to the stakeholders, we define the conditions under which we ask for new permissions. For this, we use prior work in *access control spaces* [35] as guidance. The entire request space can be divided into *prohibited*, *permissible*, *specified* and *unknown* subspaces. The base policy on the client defined by the manufacturer corresponds to prohibited and permissible subspaces. Core applications that have already been granted permissions to run, including the default phone dialing, SMS, and phone book applications come under permissible subspace. Critical resources like SIM card secrets are permanently denied access are part of the prohibited subspace. Unknown subspace is the remaining set of permissions that have not been defined locally. When an access request for a permission residing in the unknown subspace is encountered, the decision is sent to the stakeholders, and the response becomes part of the phone policy, and this transitions the request from unknown to the specified subspace. The permission is added to the *specified* rather than the permissible or prohibited subspaces because such assignments are transient. In general, a stakeholder may have a choice of whether to grant or revoke the permission unconditionally, but at present, we only consider temporary assignments (if authorized), and may extend the prohibited subspace. In order to support changes to the prohibited and permissible subspaces, we would need to upload a new security policy that overrides the existing default policy on the phone, and sets a new initial state.

By ensuring only the *unknown* subspace decisions are referred to the stakeholders, we can reduce unnecessary communication overhead, and ensure scalability of our framework. A revocation of permission results in the access request associated

with the revocation returning to the unknown subspace.

### 5.2.2 MAC Sandbox and Dynamic Requests

In a traditional MAC system, formulation of an accurate and precise policy is performed by the system administrator. Also we would need to consult all stakeholders and enforce a conflict free consolidated policy. Further since the configuration in a conventional MAC system is performed one-time, every rule set needs to be accurately pre-configured at the local client itself. In a distributed system we need to identify the permissions that an application requires and enforce a dynamic policy that is not local to the client. This set of permissions is called a manifest. After identification of the manifest and other state information associated with it, we obtain the requisite permissions from the stakeholders and finally mediate the access requests using the CRM.

In order to identify the manifest, we use a strict MAC policy tailored to the unspecified subspace. We term this as a "soft" sandboxing solution or a *MAC Sandbox* since unspecified application requests get denied initially as if running in a sandbox. Thus when newly installed applications like our example Internet telephony application attempt to use cell phone functionality in the unspecified subspace, the MAC policy will isolate these requests. Conversely, specified subspace requests like core permissions and restrictions, are directly defined in the local MAC policy and do not require further processing. Since the manifest is identified dynamically, our framework eliminates the need for external certifiers.

To enable the stakeholders to make stateful decisions, permission requests are accompanied with a representation of the runtime state of the access control policy. Rather than sending the entire state of the access control policy to the proxy and

stakeholders, we propose for the local policy server to maintain *execution roles* to represent the runtime policy state. Each execution role represents a permission set from a Chinese Wall-style [8] conflict of interest policy for the system. Any stakeholder may define such a conflict of interest policy, and the local policy server maintains which permission set is currently being used. The local policy server or proxy policy server can then prevent conflicting assignments, depending on who holds the conflict of interest policy. Thus, a permission request consists of a permission assignment request (source label, target label, and requested operation) and the execution role. We transmit this access request tuple to the proxy policy server before the denial is final. We term this run-time permission request generation as a *dynamic manifest* (e.g., like a Google manifest which provides permissions for an application). We note that this is only a manifest in principle, since these requests are generated on resource access and they are not consolidated.

An important advantage of our dynamic manifest solution is that an application's unused privileged functionality does not affect its operation. This is unlike an install time certification system like Symbian where static policy directly prevents installation regardless of usage. Another advantage of our framework is that it ensures applications run with least privilege. This is because permission requests are only sent for accesses that are denied, and the application is only granted the minimum permissions it requests and nothing more.

### 5.2.3 Proxy/Policy Server

Request tuples need to be resolved according the multiple stakeholders. We may use a staging point called the proxy server to support the same. Various policy servers corresponding to the different stakeholders are connected to the proxy

server. If bypassed, the client needs to query every stakeholder's policy server individually and cache responses in the local policy server. The remote proxy server design avoids this by handing policy consolidation externally and thus minimizes the complexity and delay overhead for the client. We can arbitrarily scale the number of stakeholders supported as required with this design. The trade-off is the overhead of an additional component in the framework. We note that although the proxy server component is optional, the rest of this thesis assumes it is part of the framework. Implementations without the proxy server are straightforward as its functionality is just pushed to the local policy server.

#### 5.2.3.1 Policy Composition

Policy rules are generally defined as explicit "allow" with default deny to support no interest. Explicit "deny" rules also exist, which are overarching restrictions on request tuples. This is termed as strict-policy. In contrast, a targeted-policy scheme is where all undefined requests are allowed and the policy only consists of "deny" rules. However this is less secure as we then have to define every malicious scenario.

In the case of multiple stakeholders, we need a method to combine each individual policy corresponding to a stakeholder into a holistic policy that end hosts can refer to. This mechanism is called Policy composition and it involves the actual combination step and a method to resolve conflicts. Policy composition is handled by using the following semantically different constructs. We leverage research already performed in this area by Bonatti et al [16].

- Conjuction(&) - All individual policies must explicitly have an allow ruleset. Implemented by "oring" access vectors to obtain the intersection of policies.

- Addition(+) - At least one of the policies must have an allow ruleset. Implemented by "anding" access vectors to obtain the union of individual policies.
- Subtraction(-) - Can be considered as addition of explicit deny rulesets of individual policies, or direct removal of rulesets.

Using these basic constructs, we can implement complex policy composition like overriding. For example, in the originally composed policy $P_1$, we can override decisions made by stakeholder A having policy $P_2$, with those approved by stakeholder B having policy $P_3$ as follows.

$$Override : P_1(P_2, P_3) = (P_1 - P_2) + (P_2 \& P_3) \tag{5.1}$$

Here we remove the portion P2, and add the conjunction of policies P2 and P3. Thus only those rulesets which have been authorized by P3 are enabled in P1. Similarly other complex policy compositions can be represented by using these basic constructs.

Consensus is when one or more policies have an allow ruleset and none exists with an explicit deny ruleset or conversely when one or more policies have a deny ruleset and none exists with an allow ruleset. While consensus enables us to provide a basic implementation that is elegant and simple, it does not take into account conflicts that occur by treating all stakeholders equal. For example, a secondary stakeholder say, the manufacturer, should not override a primary stakeholder, say the telecommunications provider, in the context of a particular access decision. Conflicts are detected by checking for consensus during policy composition. In case of a conflict, we have the following techniques of dealing with it.

- Priority - Each individual policy is assigned an increasing priority, and con-

flicts are resolved by accepting a higher priority rulesets over a lower one.

- Majority - Number of allow decisions are tallied against denied decisions, and the larger sum is selected as the final decision. We can also use a threshold to decide the decision boundary instead of simple majority.

- Weighted Majority - Number of allow decisions are weighted against their assigned priorities and then tallied against similarly weighted and tallied denied decisions. Finally, the larger measure is selected as the final decision.

Thus in the above example, the priority approach to resolving the conflict is to assign the primary stakeholder, the telecommunications provider a higher priority compared to the secondary stakeholder, the manufacturer. Now when there is a ruleset conflict, the higher priority ruleset belonging to the telecommunications provider is chosen as the final decision response. The majority approach, is to tally the number of allow, deny decisions, ignoring the "no interest" responses of all the stakeholders. The larger sum is then chosen as the final decision response. In case of a tie, we need to use an additional constraint like priority to break it. The final approach is to use a weighted majority, that combines advantages from the priority and majority techniques. Further identification and mapping of policy compositional techniques for different stakeholders is left as future work.

#### 5.2.3.2 Maintenance of State

We now look at how state information, i.e. role transmitted with the dynamic manifest helps us make permission decisions at the proxy server. When the Internet telephony application first runs on the MAC enabled phone, it requests permission for access to the handset speaker and microphone. The service provider stakeholder grants this permission and it is inserted into the local cache since it

is not a violation of policy. The phone changes state into the "handset speaker and microphone-enabled" role. Following this, the application then attempts to access Wi-Fi to stream the voice, another request is made to the stakeholder, and this time, based on the stateful information that handset speaker and microphone is already permitted, the stakeholder can identify that an Internet telephony application may be running and either deny the new request, or revoke the previously granted permissions and permit the Wi-Fi request. This would also change the state from "handset speaker and microphone enabled" role to the "Wi-Fi enabled"role. If "handset speaker and microphone" was not already enabled, the proxy server would have permitted access to "Wi-Fi" as the end-user cannot use Internet Telephony unless both are active. Since these conflict sets remain static for a given policy, an optimization is to transfer the conflict sets to the local policy server, and use revocations when there are updates to cached conflict sets.

We note that other complexities in state might be involved, for example, temporal restrictions on application access or spatial restrictions, based on a roaming/home network subscriber, may be required. In order to deal with dynamic state like temporal or spatial information, we use the local policy server to trigger updates in enforced policy according to the current state. For example, say we need to restrict access to Internet Telephony to a non-roaming user, during peak hours (9AM:6PM). The local policy server maintains additional temporal and spatial state for Internet Telephony conflict set and pushes a permit access response if the current time is in the peak hours interval, and if the user is in the home network. When the peak hours time interval has elapsed, or the user is roaming, the local policy server triggers a revocation and the system is reset. An external revocation from the proxy policy server/stakeholders removes the local policy server's conflict set and associated state information in case of a policy change.

We also note the framework is flexible enough to deal with increased contextual information in determining access requirements. In case of increased context, the local policy server can be modified with additional boolean checks to operate under the new conditions in addition to temporal and spatial constraints. For example, say we want to include a new binary conditional variable "x". Then we would check this variable x in addition to the regular time/space constraints before pushing the permit access response into the local security policy. Only the local policy server needs to be modified to allow this functionality.

### 5.2.4 Incremental Policy Addition

The decision payload returned from the stakeholders or proxy server, is transferred to the local policy server. Here depending on the category of response, i.e. permission or application specific, local policy updates are performed.

Permission specific responses are termed as incremental policy addition. We explain its operation through our example scenario. The Wi-Fi access request from our Internet telephony application is in the unspecified subspace and correspondingly the local policy server transmits a request tuple to the proxy server, which replies with an instant request-specific response. The local policy server receives this response, identifies it to be request specific, and inserts into the local client cache. Since the cache is always checked before consulting the local policy, we exploit this layer of indirection to enforce system policy. We minimize the performance overhead as once the cache is populated, subsequent permission responses are returned directly through cache hits.

The main advantage of this online approach is twofold. First, downloading complete policy modules involves significant overhead for every permission reply

received. Modifying a monolithic policy involves downloading the new binary policy to the phone, which is in megabytes, and inserting it into the kernel. Even in modular form, binary policy modules for meaningful applications like Firefox or Thunderbird are about half a megabyte in size and need to be inserted into the kernel. Unknown high risk applications conceivably result in frequent permission requests and revocations exacerbating the problem. Second, and more importantly, it greatly simplifies revocation, which is critical in dynamic environments. Since these transient decisions are only inserted into the cache, revocation becomes as easy as invalidating the cache either incrementally or completely. Thus in this mechanism, we exploit cache poisoning as a fundamental feature to provide low overhead policy enforcement and easy revocation.

### 5.2.5 Batch Policy Module Insertion

The application specific solution is utilized if the application making requests is recognized by the stakeholder and it has a module available that encompasses resource requirements at the proxy server. We assume our Internet telephony application is a store application bought from a stakeholder. Known applications are recognized by Application IDentifiers (AppID) that are provided by stakeholders to developers as a premium service for money. In this case, the application module is provided directly in response to the access request, and the local policy server inserts the module into the client MAC policy. This mode of batch operation where a complete permissions manifest is downloaded one-time to the client, is more akin to traditional policy servers that provide on-demand policy modules. Revocations are a little challenging, as policy changes will require a new module download to override existing policy. However as this option is used only for recognized applica-

tions, the frequency of revocations should be low and thus easily handled. In this mechanism, we exchange the advantage of easy revocation for a more persistent solution that alleviates per-access response delay.

### 5.2.6 Revocation

The telecommunication system was initially built to be a closed network, but with the advent of smart phones, we see a constant evolution of permissions in an dynamic environment, where millions of end users routinely install and uninstall new applications. In order for the service provider to effectively control the end points of this network, revocation of permissions is a high priority. Our framework has been carefully designed with revocation in mind, so it supports both mass revocation and functionality-specific revocation.

We first look at revocation of the request specific, incremental decisions. The permission replies from the policy server are inserted into the cache instead of modifying the phones local policy directly. Now resetting the phone to its original state and revoking all permissions can be performed by a simple cache invalidation. Apart from mass revocation, we can also invalidate cache entries line by line by simply removing them individually from cache. Invalidation can be triggered by requests from the policy server as and when required.

Revocation of the application specific, policy insertions need to be handled a little differently. In this case, in order to invalidate permissions the old policy module will require removal and a new module is reinstalled at the client. However as policy insertions are performed only for stable recognized applications, there is less need for revocation in this case.

Another interesting feature of our framework is the ability to support pay-per-

use applications. We associate a counter with each cache entry that is initialized with the desired number of permitted accesses. Then decrement this counter every time a service is accessed by an application. When the counter reaches zero, we simply invalidate the cache entry denying further access to the service. Thus, a user may be allowed to access Internet telephony services for say, up to 50 times a month or send 100 SMS's a month.

## 5.3 Discussion

In this section we discuss some additional features of the framework that add to its usability.

### 5.3.1 Auditing

Having a tightly coupled security framework between the handset owner and the service provider allows both parties to maintain control over content and services. For example, permissions requested that are not granted by the handset local policy have to be granted by the service provider. This means that all access requests can be logged and maintained at the proxy server per user. This kind of auditing allows the service provider to have greater visibility over the services accessed by third party applications. For example, a third party SMS application installed by the user will require permission to access SMS services. When the user requests this permission from the proxy server, the service provider can flag the user as one with greater risk of misusing SMS resources. Thus if the user's handset appears to perform an anomalous operations like sending hundreds of SMS's shortly after installation of a third party application, the service provider can identify the software at runtime. This is in contrast to "Symbian Signed" which performs

extensive source code analysis on every signed application to provide the same feature. This also allows the service provider to maintain higher vigilance on users who install high risk content on their handsets. For example, an elderly couple might never install the latest third party games from an online site. Hence they will not have many requests for permission and will not require careful monitoring. Whereas, a user that installs many third party applications with access to critical services can be flagged as high risk, and monitored closely for any anomalous behavior. Intensive fine grained security flagging is often very useful in identifying security holes in a system quickly but there is a corresponding trade off in terms of privacy. However, not all loss of privacy infringes upon the user rights and many may be willing to balance their privacy requirements for security. However a thorough study of privacy issues and user behavior is out of scope for this paper.

### 5.3.2 DySEL as IDS

Since we already have the complete policy module for premium and core applications, any denied requests for the same can be assumed to be a deviation from its normal profiled behavior. For example, through a code injection attack like a buffer overflow, a highly privileged phone banking application attempts to access the Bluetooth device to offload user information. In such a scenario owing to the least privilege permissions constraints, the phone banking application should not have any access to the Bluetooth device, and any such resource request can be identified as a deviation from normal profiled behavior and classified as an attack. While we note that the MAC framework will prevent such an attack, we can additionally flag these deviations to provide basic Intrusion Detection System (IDS) capabilities.

### 5.3.3 AppID

The AppID assigned for each premium application that is recognized by the proxy policy server is unique to the proxy (i.e., service provider). This AppID is transmitted as part of the access request tuple and enables easy identification of the premium application and its corresponding tailored policy. Unknown applications are assigned an AppID that is locally unique on installation, but may have duplicates globally. However, during analysis and identification of rogue applications this locally unique AppID can be combined with a unique identifier for the phone like IMSI (International Mobile Subscriber Identity) or IMEI (International Mobile Equipment Identity).

### 5.3.4 Offline operation

In case of disconnection from the stakeholders we note that the basic operations of the phone are uninterrupted as these core applications form the *permissible* subspace that have been pre-configured in the local policy. Existing third party applications that have been granted permissions in the *specified* subspace also operate without interruption. Revocations that either time out locally, or invalidate themselves after a preset number of executions are performed successfully. The main impact would be the default deny policy that is enforced on newly installed third party applications that have not been verified against stakeholder policy. However we believe this is an acceptable stance, similar to cell phone operators denying service to unauthenticated phones. Another aspect of network disconnectivity is the inability to enforce revocations pushed from the stakeholders. A possible technique to mitigate this impact is to coerce timeout or limit execution attempts in the specified subspace, so transient permissions do not become

permanent and users have to periodically connect to the stakeholders to ensure their policies are not outdated. This is similar to certificate systems where offline revocations are performed through certificate expiration.

# Chapter 6

# Implementation

In this chapter we discuss the implementation of our framework designed to solve the requirements outlined in section 4.2. We integrate each functional component of the solution into our experimental Linux testbed. Finally we analyze and graph the framework's performance in terms of overhead added to a conventional MAC system.

## 6.1 Experiment Testbed

In this section we explain the details of the experimental testbed setup for the framework. Since our framework is an extension of a MAC system, we use the existing SELinux MAC framework that is a standard part of the 2.6 Linux kernel as our base MAC system. We modify the SELinux LSM module in the Linux 2.6.27.2 kernel according to our framework requirements. Our experiment showed that it is relatively simple to introduce our functionality as a patch into the kernel with minimum overhead. This makes adoption of our framework simple for existing Linux based platforms. We envision that implementing a similar framework for

other platforms like Windows Mobile, Symbian, and RIM Blackberry is equally feasible.

Establishment of a secure channel for communication like IPSEC or TLS is an important requirement for our approach. This is because we need to guarantee end-to-end security to avoid attacks on the transmitted data between server and host. In the telecommunications network, it is not unrealistic to assume a secure channel between the user and the core network. In our experiment we assume the presence of a secure channel and bypass this requirement to simplify implementation.

The hardware used for the experiment was a Dell Optiplex GX620 machine with Intel Pentium Dual Core processor @ 3.20 GHz. An Intel Pentium Centrino processor @ 1.60 GHz was connected over Ethernet LAN and used as a server. In order to enable basic SELinux support in the host, the kernel needs to be recompiled after enabling *CONFIG-SECURITY-SELINUX* and *CONFIG-SECURITY-SELINUX-BOOTPARAM* options. SELinux has two modes of operation namely, *Enforcing* and *Permissive*. In enforcing mode, SELinux will prevent operations that are not permitted by the policy. By comparison, in permissive mode every violation is logged for audit purposes but the operations themselves are allowed to proceed. We patch the kernel with our framework changes and boot it in permissive mode for testing purposes. The host running the patched SELinux framework communicates with a simple TCP/IP server program that responds to requests over LAN.

## 6.2 SELinux MAC framework

In this section we explain how the SELinux framework enforces MAC. SELinux uses a reference monitor to mediate all access on a system. It is predominantly a

type enforcement system (TE). All entities are labeled with contexts, and when a subject tries to access an object, both the source contexts (scontext) and target contexts (tcontext) are checked for access control in the security policy. SELinux was modified to operate as part of the Linux Security Module (LSM) framework. While the LSM uses TSIDs (Target Security Identifier) and SSIDs (Source Security Identifier) to identify the source and target objects, these get translated to source and target contexts in the SELinux module.

Explicit "allow" permissions that define access control for each source-target context pairing are represented as *avc->allow* bitmasks. These are 32 bit identifiers where each set bit refers to an allow decision for a specific operation request. Access control requests are represented in the same format as bitmask permissions. Except, only the requested permission bit is set, all other bits are zeroed. To identify if access should be granted or denied, a bitwise "and" is performed between the requested and *avc->allow* bitmask. If the result is non zero, access is granted. The access control bitmask is obtained by performing a lookup in the Access Vector Cache (AVC), using the tuple consisting of scontext, tcontext, and *class* identifier. The class identifier corresponds to entity-types like file, socket, directory, etc. If the bitmask has not been cached yet, it is obtained from the Access Vector Table (AVT) and populated in the AVC for subsequent requests.

We have identified that transmitting the scontext, tcontext and class identifier is enough to uniquely identify a permission request. The source and target contexts stay consistent between different machines and this enables us to maintain coherency between the stakeholders and local system.

Apart from the *avc->allow* bit access vector described above, there are two other access bit vectors namely *avc->auditallow*, and *avc->deny*. The auditallow vector corresponds to one that grants permission and has its output logged in the

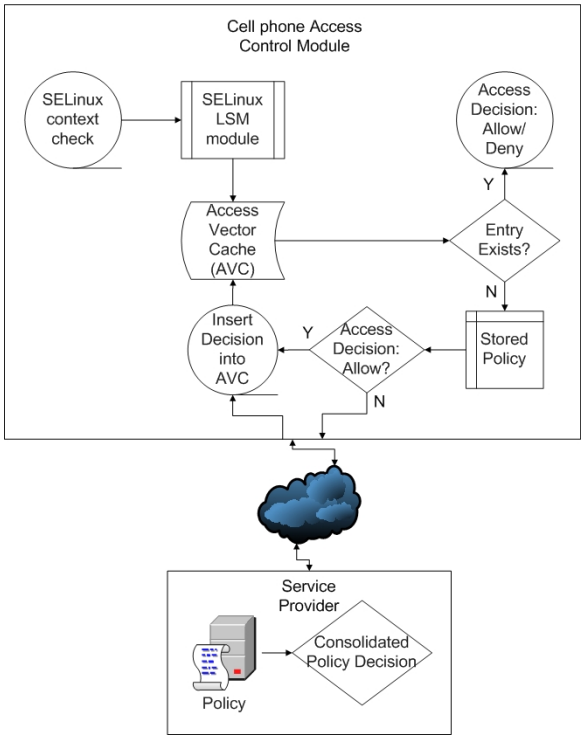


**Figure 6.1.** Implementation Framework for MAC Security Model

audit report. This is useful for troubleshooting and logging purposes. The deny access vector is similar to the allow vector, except it is initialized as ones and zero corresponds to allow. These access bit vectors used together allow us to obtain a fine grained policy.

## 6.3 Dynamic MAC Implementation

In this section we explain how we implement our dynamic MAC approach into the existing SELinux framework. Below are the implementation details of the main components in our framework.

### 6.3.1 MAC Sandbox

The MAC sandbox is the primary component in the implementation of the dynamic request concept. The implementation framework is outlined in figure 6.1. We define the handset user *sysadm_t* as our subject and a new object type *untrusted_t*, which corresponds to an unknown third party application. The phone's pre-configured local MAC policy is limited to only core applications and this becomes our specified subspace. The specified subspace is inserted into the AVC as part of normal SELinux operation when an access request causes a lookup from the AVT. However, when a user of type sysadm_t attempts to execute untrusted_app, it is not allowed, as this permission is not enabled in the local policy (unspecified subspace). This simulates running unknown applications in a sandbox, and this becomes our dynamic manifest. We install the local policy on the host machine to set up the MAC sandbox.

Once we obtain the unspecified access request, we insert a hook and instead of the default deny message being logged, an independent kernel thread is spawned to communicate with the proxy policy server. The security tuple consisting of the scontext, tcontext, target class, and current role, if one exists, is sent to the proxy server via a secure network socket.

### 6.3.2 Proxy Policy Server

Policy consolidation is performed at the proxy server using any of the techniques mentioned in section 5.2.3. Hence the proxy server maintains policy decisions composed from multiple stakeholders. This allows an immediate response to the submitted requests. When the MAC sandbox sends a permissions request to the proxy server, it checks the transmitted AppID to confirm if it is a known appli-

cation. If it is unknown, the incremental access decision vector is returned to the local policy server. Instead, if the request is for a known application with a policy module already existing at the proxy server, we just send the compiled binary module to the local policy server for insertion. The incremental decision and policy module insertion implementation is explained further in the following subsections. Similarly revocation requests, both cache invalidation and module removal requests can be initiated from the server.

#### 6.3.2.1 Incremental Access Decision

The proxy server on receipt of a request message checks its consolidated policy database and obtains the access decision. This decision is communicated back to the local policy server over the network. It is important to note here that the client does not busy wait for an access vector decision from the policy server. The kernel thread created earlier blocks while waiting for the access vector decision. On receipt of the packet from the proxy server, the local policy server performs a bitwise "or" between the local bitmask and the proxy server bitmask. This corresponds to a union of the local and remote decision vectors. This new bitmask is then "anded" with the requested permission vector to generate the allowed/denied decision. the local policy server inserts this access vector decision in the AVC. This functionality is verified by viewing the kernel log messages. Once a decision is inserted into the cache, the next time the same functionality is accessed, the access decision is returned as a cache hit. Thus the unspecified request, becomes part of the specified subspace and exists in the cache. It will once again be downgraded to unspecified only in the event of revocation.

Depending on the network traffic as well as the load at the policy server, the client might have to wait momentarily before it gets the correct access control

decision from the policy server. This one time transient wait at the client before it can run the application is an example where system security is given higher priority over a minimal performance increase.

#### 6.3.2.2 Policy Module Insertion

In order to support the policy insertion part of our framework, we wrote an SELinux policy module "trustedm.te" for our test application. The module creates a new object type called *trusted_t* that corresponds to a known phone application that is trusted and given access permission by the local policy server. The policy allows the user of type *sysadm_t* to read and execute a file of type *trusted_t*. This pre-compiled modular policy can be inserted into the kernel through the *semodule -i ¡module name¿* command dynamically without recompilation of the entire policy. When a user of type sysadm_t attempts to execute trusted_app, it is now granted permission to execute by phone's local SELinux policy. This is confirmed by the AVC logs. Revocation is achieved by using the *semodule -r ¡module name¿* command to remove the corresponding policy module.

### 6.3.3 Revocation

Revocation of granted permissions in case of an attack is well facilitated. This is done by allowing cache invalidation of a single-entry or the entire cache by sending an explicit invalidate request from the server. The ability to perform single entry cache invalidation after specific number of accesses is supported by invalidating cache entries that exceed a specific number of cache hits. The only modification to support this feature is insertion of a counter variable in the cache entry. On every cache hit, the counter variable is decremented and when it reaches zero, the entry

**Table 6.1.** Implementation delays for access request

| | |
|---|---|
| Data Payload Size | 28 bytes |
| Processor Speed | 3.2 GHz |
| Plain SELinux Kernel | 946 cycles = 0.295 $\mu$s |
| W/Local Policy Server | 1485 cycles = 0.464 $\mu$s |
| W/remote Policy Server | 1870 cycles = 0.584 $\mu$s |
| Single Request Overhead | 924 cycles = 0.288 $\mu$s |
| Network Round Trip Delay | 3 ms (approx) |

is deleted from the cache, and this revokes its permission. Alternatively, the entire cache can be invalidated and this resets the phone into the initial state with the default policy.

## 6.4 Performance Overhead

In this section, we study the performance overhead added by our framework implementation and compare it to a conventional MAC system. Introduction of our framework into an existing system is relatively easy as SELinux is now standardized in Linux kernels. In terms of performance, overhead is limited to initial resource access by newly installed applications and consists of round trip time required for communication between the client and the proxy server. The access decisions from the proxy server are cached. Hence, subsequent resource access decisions are returned as local cache hits with no additional overhead. The advantages we gain in exchange for this initial overhead include novel features like easy revocation and incremental policy updates, which improves performance and scalability in dynamic environments.

The unoptimized kernel patch currently consists of 256 lines of code. We measure our framework overhead by measuring the difference in response time for a cache miss in our framework and the standard SELinux implementation. We

note that the response time for cache hits are identical with no overhead, as no additional steps are executed in our framework. Our measurement values are summarized in Table 6.4. The unmodified kernel had a response time of 0.295$\mu$s for a cache miss. The response time for a cache miss in our framework with the proxy server was 0.584$\mu$s. Hence our overhead was measured to be 0.288$\mu$s. Additionally the overhead reduces to only 0.166 $\mu$s if just the local policy server is used. Of course, this measurement is independent of round trip time which varies depending on the network used and is probably a bigger bottleneck. However, we mitigate its influence by using caching and further, core applications and stakeholder recognized applications (specified subspace) will have their access decisions in the local policy itself.

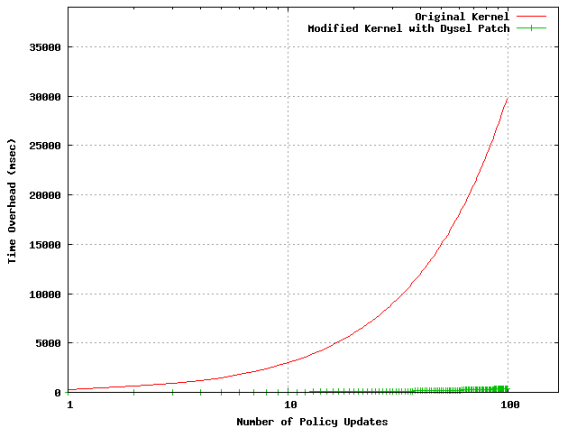


**Figure 6.2.** Time Overhead vs Number of Policy Updates

As shown in Table 6.4, network round trip time dominates the framework overhead. The alternative to our framework in distributed environments is to download a new binary policy module each time a permission request is sent to the policy server. An average application policy module like Firefox or Thunderbird is approximately half a megabyte in size. Each time we want to update the local policy, no matter how trivial the change, the entire binary module needs to be

transferred and installed into the handset. We use an example scenario to compute potential savings obtained in our framework, by eliminating the need for complete policy updates. Consider a 1.5 MB/s network connection transferring 0.5 MB for each policy update. In a dynamic network, we expect many policy revisions including revocations of permissions at regular intervals. Once the number of policy updates transferred to the device approaches 100, the total time overhead for transfer of complete policy updates reaches about 33 seconds. By comparison, our framework does not need to transfer entire 0.5 MB payload as it uses the incremental policy update model (28 byte payload consisting of only the access bit vectors) and hence, efficiently scales in the same scenario to just 300 ms. This is shown in figure 6.2.

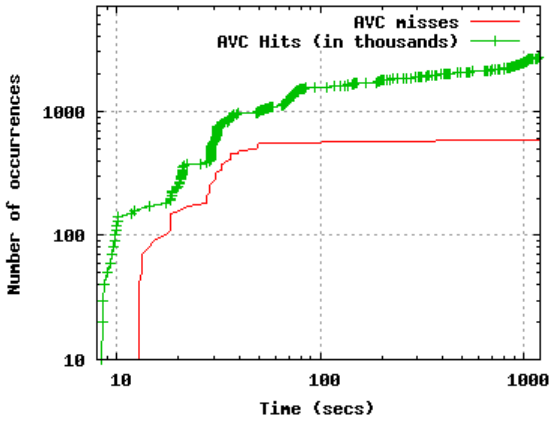


**Figure 6.3.** AVC statistics vs Time

We also present our motivation for using the access vector cache to store transient decisions, in Figure 6.3. From this graph, we see that after the system stabilizes, all security decisions are returned from the cache directly. The number of cache misses does not increase after 100s while the cache hits continue to increase steadily. Thus after the system converges to a stable state, communication between the local policy server and proxy server will be minimized and round trip

delay will not impact the system significantly over time. Only in the event of a revocation, the cache gets reset and the system will need to re-insert the new security decisions into the cache. However, the alternative of downloading entire policy modules instead of cache insertion, will result in much greater overhead. We leave further research regarding optimization of cache size, and dealing with cache replacement as future work.

# Chapter 7

# Conclusions and Future Work

We have presented a novel security framework using Mandatory Access Control (MAC) in a distributed environment. The main contribution of this paper was to provide a security framework with support for new features like multiple stakeholders, dynamic permission derivation, and extensive revocation support. Our framework was implemented by modifying the SELinux module and using a policy server database that can validate security permission requests in real time. The performance overhead introduced in the kernel was minimal at 0.288 $\mu$s. Our model provides an end-to-end solution using SELinux deployment and is beneficial to providing system security in a distributed environment.

In our future work, we plan to study the following: Impact of having policy servers outside the TCB and possibility of collusion among policy servers. A more in depth study of consolidation techniques of different stakeholders and compliance testing to ensure conflicts are resolved. Further research regarding optimization of cache size, and dealing with cache replacement.

# Bibliography


[1] Market Research (2009), "3Q09 United States Mobile Operator Forecast, 2009 - 2013," "http://www.marketresearch.com/product/display.asp?productid=2422847&g=1".

[2] F-Secure Weblog (2008), "New Century in Mobile Malware," http://www.f-secure.com/weblog/archives/00000864.html.

[3] W. Enck, M. Ongtang, and P. McDaniel (2008) *Automated Cellphone Application Certification in Android*, *Tech. rep.*, Pennsylvania State University.

[4] Symbian Limited (2006), "Symbian OS - the mobile operating system," http://www.symbian.com.

[5] National Security Agency, "Security Enhanced Linux," http://www.nsa.gov/selinux.

[6] Bell, D., L. L. Padula, M. Ben-Ari, and G. Benson (1988) "Secure Computer System Unified Exposition and Multics Interpretation," *Communications of the ACM.*

[7] Ferraiolo, D., J. Cugini, and D. Kuhn (1995) "Role-Based Access Control (RBAC): Features and Motivations," *Proceedings 11th Annual Computer Security.*

[8] Brewer, D. and M. Nash (1989) "The Chinese Wall security policy," *Security and Privacy, Proceedings.*

[9] Gartner Research (2008), "Smartphone Sales by OS," http://www.gartner.com/it/page.jsp?id=910112.

[10] Ars Technica (2008), "Smartphones market forecast," http://arstechnica.com/open-source/news/2008/06/23-of-smartphone-market-to-be-linux-powered-by-2013.ars.

[11] IT Facts.biz (2008), "Linux to lead smartphone OS market with 26.6% by 2010 - Diffusion Group," http://www.itfacts.biz/linux-to-lead-smartphone-os-market-with-266-by-2010/7040.

[12] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, A. Essiari (1999) "Certificate-based Access Control for Widely Distributed Resources," in *Proceedings of the 8th USENIX Security Symposium*, pp. 215–228.

[13] "Tresys Technology, SETools for SELinux," http://www.tresys.com/selinux/selinux_policy_tools.shtml.

[14] Sloman, M. (1994) "Policy Driven Management For Distributed Systems," *Journal of Network and Systems Management*, **2**, pp. 333–360.

[15] Lupu, E. and M. Sloman (1999) "Conflicts in Policy-based Distributed Systems Management," *IEEE Transactions on Software Engineering*, **25**, pp. 852–869.

[16] Bonatti, P., S. De Capitani di Vimercati, and P. Samarati (2002) "An algebra for composing access control policies," *ACM Trans. Inf. Syst. Secur.*

[17] Rao, P., D. Lin, E. Bertino, N. Li, and J. Lobo (2008) *An algebra for fine-grained integration of XACML policies*, *Tech. rep.*, Technical report.

[18] Jaehong Park, R. S. (2004) "The UCON usage control model," in *Proceedings of ACM Trans. Inf. Syst. Secur., vol 7*, pp. 128–174.

[19] Bertino, E., P. A. Bonatti, and E. Ferrari (2001) "TRBAC: A temporal role-based access control model," *ACM Trans. Inf. Syst. Secur.*, **4**(3), pp. 191–233.

[20] Enck, W., P. McDaniel, and T. Jaeger (2008) "PinUP: Pinning user files to known applications," in *Proceedings of the Annual Computer Security Applications Conference*.

[21] Google Android (2009), "Android - Official Website," http://www.android.com/.

[22] Krohn, M., A. Yip, M. Brodsky, N. Cliffer, M. Frans, K. Eddie, and K. R. Morris (2007) "Information flow control for standard OS abstractions," in *In SOSP*.

[23] Zeldovich, N., S. Boyd-wickizer, E. Kohler, and D. Mazires (2006) "Making Information Flow Explicit in HiStar," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, USENIX Association, pp. 263–278.

[24] Richtel, M. (2000) "Yahoo Attributes a Lengthy Service Failure to an Attack," *The New York Times*.

[25] Roberts, P. (2003) "Al-Jazeera Sites Hit With Denial-of-Service Attacks," *PCWorld Magazine*.

[26] Ilett, D. (2005), "Symantec website under DDoS attack," `http://software.silicon.com/malware/0,3800003100,39150478,00.htm`.

[27] Internet Corporation for Assigned Names and Numbers (ICANN) (2007), "Factsheet: Root server attack on 6 February 2007," `http://www.icann.org/announcements/factsheet-dns-attack-08mar07.pdf`.

[28] Traynor, P., M. Lin, M. Ongtang, V. Rao, T. Jaeger, T. L. Porta, and P. McDaniel (2009) "On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core," in *Proceedings of the ACM Conference on Computer and Communication Security (CCS)*.

[29] Guo, C., H. J. Wang, and W. Zhu (2004) "Smart Phone Attacks and Defenses," in *Proceedings of Third ACM Workshop on Hot Topics in Networks*.

[30] Traynor, P., V. Rao, T. Jaeger, P. McDaniel, and T. La Porta (2007) *From Mobile Phones to Responsible Devices*, *Tech. Rep. NAS-TR-0059-2006*, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA.

[31] Fleizach, C., M. Liljenstam, P. Johansson, G. M. Voelker, and A. Mhes (2007) "Can You Infect Me Now? Malware Propagation in Mobile Phone Networks," in *Proceedings of the ACM Workshop on Recurring Malcode*.

[32] Kelby, S. and T. White (2007) "The iphone book: how to do the things you want to do with your iphone," .

[33] Miller, C., J. Honoroff, and J. Mason (2007) "Security Evaluation of Apple's iPhone," .

[34] Benavides, L., F. Taliberti, and J. Domene (2005) "The Rise of Wireless VoIP," *Master in E-Business and ICT for Management–Politecnico di Torino, Italy*.

[35] Jaeger, T., X. Zhang, and A. Edwards (2003) "Policy management using access control spaces," *ACM Transactions on Information and System Security (TISSEC)*.