SPF: Security Performance Flexibility Framework for Trusted Operating Systems

Nitish Pathak¹ and Neela Sharma²
¹BVICAM, GGSIPU, Delhi, India
²Department of Information Technology, Maharaja Agrasen Institute of Technology, Delhi, India
¹nitish_pathak2004@yahoo.com and ²sharmaneelam2@gmail.com

ABSTRACT
The rapid growth of networking, data sharing, and the Internet has made computer security an important part of computer research and development. A number of highly secure operating systems have been developed to handle the increasing need for security. These operating systems, typically called Trusted Operating Systems, offer a number of security mechanisms that can help protect information, make a system difficult to break into, and confine attacks far better than traditional operating systems. However, this security will come at a cost, since it can degrade the performance of an operating system. This performance loss is one of the reasons why Trusted Operating Systems have not become popular.

While Trusted Operating Systems offer an incredible amount of security, observations about computing workloads suggest that only some parts of the operating system security are actually necessary. Web servers are the best example. For many web servers, the majority of the information on the server is publicly readable and available on the Internet. Therefore, if a Trusted Operating System is used on a web server, any security used to secure the confidentiality of the server’s information is not necessary. Any security used to protect the confidentiality of web server data can be considered a waste of computational resources. The security needed in web servers is the security to protect the integrity of data, not the confidentiality of data. Other workloads such as multimedia or database workloads may also only need parts of the operating system security.

Based on this observation, this research paper proposes the Security Performance Flexibility (SPF) Framework for Trusted Operating Systems. SPF recognizes that not all computing workloads require all the security in Trusted Operating Systems. SPF allows system administrators to selectively disable parts of the security in Trusted Operating Systems. By disabling parts of the Trusted Operating System security, performance of the system can potentially be increased. The SPF framework allows system administrators to balance the security and performance needs in their particular computing environment.

KEYWORDS

1. LITERATURE SURVEY
In particular, security developed and placed in the application layer of a system is the most inadequate to provide high security. Unfortunately, this is where a large majority of security has been researched and developed. This software includes (but is not limited to) cryptography, authentication, firewalls, and application layer access control. Cryptography and authentication techniques are probably the most popular form of security implemented and researched today. While it is extremely difficult to break, it has been and always will be subject to brute force or man in the middle attacks [COULOURIS01]. Firewalls and application layer access control can also help towards preventing malicious attacks.

Paul Clark perhaps stated it best, stating that most of today’s systems are, “crunchy on the outside, yet soft and chewy on the inside” [CLARK00]. The quote points out that most of today’s computer security is placed in the application layer of an operating system and at network entry points. However, there is little to no security placed inside an operating system kernel. All of the security problems that exist at the application layer suggest that operating system security mechanisms might protect a system far better.

Due to recent rising security concerns in the commercial sector, many of these operating system security technologies are being looked at and implemented again. Vendors such as Argus-Systems Group, Hewlett-Packard, and Sun Microsystems are developing highly secure operating systems with these security technologies [ARGUS] [TRUSTEDSOLARIS] [VIRTUALVAULT]. Several open source operating systems are also being developed with high security in the kernel. The NSA has released a secure version of Linux called Security-Enhanced Linux (SELinux) [SELINUX]. Linux distributions such as Rule Set Based Access Control Linux (RSBAC) and Linux Intrusion Detection System (LIDS) are also available [LIDS] [RSBAC].

2. TRUSTED EVALUATION CRITERIA
These highly secure operating systems are typically called Trusted Operating Systems (TOS), Trusted Systems, or Secure Systems. However, there seems to be four key features in all Trusted Operating Systems. They are Discretionary Access
Control (DAC), Mandatory Access Control (MAC), Least Privilege, and auditing. The ratings system established criteria to rate the strength of security mechanisms in operating systems. The strength of the security is classified into one of four classes. The classes are indicated as class A, B, C, or D, where class A systems have the highest amount of security and class D systems have the least [ABRAMS95].

Class D systems have minimal or no security built into the operating system. It is the class used to describe any operating system that cannot meet at least a C rating. Although it may be surprising, MS-DOS and Windows 95/98/Me are considered class D operating systems [TANENBAUM01].

Class C systems provide Discretionary Security Protection or Discretionary Access Control (DAC). They also provide auditing features. Many modern day operating systems, such as UNIX, Linux, and Windows NT, are considered class C operating systems. In addition to DAC and auditing, Class B systems provide Mandatory Security Protection or Mandatory Access Control (MAC).

In very general terms, class B systems are Trusted Operating Systems. Argus System’s Pitbull, Hewlett-Packard’s Virtual Vault, and Sun Microsystems Trusted Solaris are all class B systems. Due to the strict criteria of the TCSEC, the NSA’s Security-Enhanced Linux is not considered a class B system. This is despite the fact that it offers most of the requirements to be considered a class B system.

Class A systems have auditing, DAC, MAC, and the ability to prove the system provides the security indicated. Generally speaking, no commercial vendors develop or distribute class A systems. They primarily exist in military environments.

3. DISCRETIONARY ACCESS CONTROL (DAC)

Discretionary Access Control (DAC) security mechanisms let users determine who is authorized to access objects and how they can be accessed [ABRAMS95]. For example, DAC mechanisms may allow the owner of a file to give all other users in the system the ability to read a file. However, the same owner may not allow others to write to the file.

The best example of DAC mechanisms in practice can be seen in UNIX and Linux operating systems. In these operating systems, every file contains a set of bits that indicate the read, write, and execute permissions for the file. Every user is allowed to modify the permissions for the files they own. They may grant read, write, or execute access for themselves, anyone in their group, or to all users in the system [TANENBAUM01]. It is important to note that DAC allows users to modify security permissions for themselves and not just for other users in the system. This feature of DAC can help prevent human error [ABRAMS95]. For example, DAC can stop a user from accidentally deleting or modifying a file they do not wish to. DAC is the most widely used security mechanism placed in operating systems today.

Unfortunately, DAC is not capable of providing high security in an operating system and is one of the major security weaknesses in today’s operating systems. DAC lacks the ability to protect both the confidentiality and integrity of information. Trojan horses are the best example of how DAC is ineffective. Malicious code can take advantage of a user’s DAC and modify the security permissions of all the files they own. Malicious code can make files publicly readable or writeable, thus destroying the confidentiality and integrity of the files. Similarly, DAC cannot always protect against malicious insiders or even mistakes by authorized users.

4. MANDATORY ACCESS CONTROL (MAC)

Mandatory access control (MAC) can be best described as security that is not at the discretion of users [ARGUS01A]. In other words, security permissions are not dependent on how a user sets permissions on objects; the security is under complete control of the operating system and the policy set by a system administrator. MAC can control the flow of information in a system much more tightly than DAC can. How MAC provides higher security than DAC is best explained through an explanation of their implementation and modeling.

Typically, MAC is implemented through the assignment of a security label (also known as a sensitivity label or integrity label) to every object in an operating system [JOSHI01]. The objects include processes, files, sockets, network packets, semaphores, message queues, and most other objects or structures in an operating system.

For example, Hewlett-Packard has developed a secure version of Linux with trusted security mechanisms built into the kernel. In their system, security labels are assigned to every object in the system. The labels partition the system into distinct compartments. Throughout the kernel, security labels are compared between processes and objects the process wishes to access. If the security labels are not equal, access is denied. These security checks ensure that processes and objects in one compartment cannot read, write, or interfere with any process or object in another compartment.

This type of policy is illustrated by Figure 1.0a. The circle represents all objects and resources in the system. As can be seen, all objects in the system have been partitioned into four different areas, labeled compartments A through D. Each compartment does not have to be equal in size nor do the resources in each partition have to be equally distributed.

By partitioning the system into compartments, higher security can be achieved than with just DAC. As an example, let us consider a Trojan horse that makes a user’s confidential data publicly readable. MAC can limit the malicious effect of the Trojan horse. Only users and processes inside the compartment will be able to read the data made public by the Trojan horse. Users and processes from different compartments will be unable to read the data. This shows that the potential consequences of security breaches can be greatly reduced with MAC.

As one last example of how labels can be used to implement different types of MAC, Figure 1.0c shows how security labels are modeled in Argus System’s Pitbull. Pitbull’s
The principle of least privilege states that users and processes should only have the privileges necessary to complete their tasks, nothing more and nothing less [ABRAMS95]. It can be very difficult to administer the minimum amount of privileges for each user and process. However, giving any user or process more authority than it needs can open the door for security breaches.

There are various ways least privilege can be implemented. MAC can be used to implement least privilege if security labels are set up properly. However, the most popular way to implement least privilege is through mechanisms similar to capabilities. The permissions in capabilities indicate what a process is allowed to access and what types of actions a process is allowed to take. Every process in the system holds a list of capabilities. These capabilities list the actions a process is allowed to perform during its lifetime. With the previously mentioned security mechanisms, least privilege can help make a system far more secure.

Let us consider a network daemon that runs in the background of a network server. Capabilities can indicate what specific actions the daemon is allowed to perform and not perform. For example, capabilities can indicate that the daemon is allowed to transfer network packets but is not allowed to access files. These permissions limit the ability of an attacker to exploit a bug or backdoor in the network daemon. If a bug or backdoor is discovered, the capabilities ensure that he attacker cannot use the security hole to access files.

Separating the power of a system administrator is the most powerful use of least privilege mechanisms. Most systems today have an all-powerful account, often called the root or super user account. System administrators handle a variety of administrative duties, such as setting up file systems, accounts, and applications, with this account. Usually, the root user is not subject to any security checks, thus making it easier to handle administrative duties.

However, having an all-powerful user in your system can be dangerous. While it makes system administration easier, an attacker will have free reign to do whatever they wish, if the root account (or any process running under the root account) can be compromised.

In order to limit the power of the root user, many trusted systems separate the power of the root user into a number of smaller powers, usually called privileges. The privileges can be collected into sets of privileges, each of which can be assigned to a sub-administrator. A set of sub-administrators, each with a subset of the original root user’s privileges, is collectively responsible for the system administration duties of the system. The root user account is eliminated. If an attacker or piece of malicious code gains power as one of these sub-administrators, the division of privileges will limit the amount of damage the attacker can perform. Along with DAC and MAC, least privilege helps make a system highly secure.

6. AUDITING
Auditing is an important part of Trusted Operating Systems security. Auditing is the recording or logging of all security relevant operations and transactions in a system. Recording information is the key to identifying the source of an attack or foreseeing future attacks. Most systems today are configured to provide some basic logging of information; however most are not equipped to handle the fine grained auditing necessary for highly secure systems. In particular, most systems cannot audit user accesses to files or system resources. This information is vital towards discovering security holes or foreseeing future security problems.

7. ARCHITECTURE
The basic architecture of Trusted Operating Systems is illustrated below in Figure 1.1. As a reminder to the reader, the architecture of a system is not the same as the implementation. A variety of implementations for trusted security mechanisms can be done.

In Trusted Operating Systems, there is a much larger amount of security placed into the operating system. Figure 1.1b illustrates this security with a much thicker layer of kernel
security checks. What is inside the kernel security check layer of a Trusted Operating System depends on the implementation. The kernel security layer may include DAC, MAC, Least Privilege, auditing, or any number of additional security features. The key point is that the kernel security checks are much larger than traditional operating systems. This large layer of security causes Trusted Operating Systems to suffer performance degradation. All system calls to the kernel must go through this layer of security checks before they can do any useful work.

8. PROBLEM DESCRIPTION

As the architecture in Figure 1.1b shows, the additional security checks in the kernel will cause Trusted Operating Systems to be slower than traditional operating systems. This is one reason why Trusted Operating Systems have not become popular in the commercial sector. As the architecture in Figure 1.1b shows, the additional security checks in the kernel will cause Trusted Operating Systems to be slower than traditional operating systems.

Trusted Operating System security may also affect multimedia and video streaming services. Typically, when video is played, each frame is read off of disk one by one and displayed on a screen. In the case of video streaming, the frame is sent out through a socket onto the network. Every single read from disk and every write out to a network socket is now slowed down by repeated security checks. This may have a significant effect on the quality and frame rate of video, especially if the system is heavily loaded. Much like the example with web servers, this video quality degradation may not be entirely necessary. For example, a Video On Demand server may care more about the quality of the video stream than the security of read accesses to the server.

As was stated in the previous section, there are several types of system workloads that repeatedly do security checks in a Trusted Operating System. For these workloads, the security checks may be undesired or completely unnecessary. The information in these workloads could be public, integrity of the system may be the primary concern, or quality of the workload is more desirable than the security of certain operations. This suggests that performance in Trusted Operating Systems can be increased if security can be disabled in some parts of the operating system. Performance can be increased if system administrators are given the ability balance their security and performance needs.

In order to provide better performance for specific system workloads, this thesis proposes the security performance flexibility (SPF) framework for Trusted Operating Systems. The SPF framework allows system administrators to disable certain security checks in a Trusted Operating System. This gives system administrators the ability to balance their security needs with their performance needs.

9. PROBLEM SOLUTION

As was stated in the previous section, there are several types of system workloads that repeatedly do security checks in a Trusted Operating System. For these workloads, the security checks may be undesired or completely unnecessary. The information in these workloads could be public, integrity of the system may be the primary concern, or quality of the workload is more desirable than the security of certain operations. This suggests that performance in Trusted Operating Systems can be increased if security can be disabled in some parts of the operating system. Performance can be increased if system administrators are given the ability balance their security and performance needs.

In order to provide better performance for specific system workloads, this thesis proposes the security performance flexibility (SPF) framework for Trusted Operating Systems. The SPF framework allows system administrators to disable certain security checks in a Trusted Operating System. This gives system administrators the ability to balance their security needs with their performance needs.

The architectural idea behind the SPF framework is illustrated in Figure 1.2. The SPF framework gives system administrators the option of disabling security checks for specific system calls. By skipping security checks in the kernel, performance for a system can be increased. For example, a system administrator can use SPF to turn off all read security checks in a web server. By turning off the read security checks of a web server, it is possible the web server’s throughput can be increased.

There are different levels at which the SPF framework can be implemented in a Trusted Operating System. Three different levels have been identified for evaluation in this paper:

1. System Wide Security Performance Flexibility (System-SPF).
10. SYSTEM WIDE SECURITY PERFORMANCE FLEXIBILITY (SYSTEM-SPF).

System-SPF provides the ability to disable all trusted security checks in particular operations of the system. For example, all read security checks can be disabled in the system (Figure 2.0). By disabling the read security checks in the entire system, performance of the system as a whole can improve. This may be particularly useful for dedicated web servers, because web servers generally have pure public information. Disabling read security checks for the entire web server may help increase web server throughput.

However, System-SPF may not offer enough fine-grained ability for system administrators to manage their system. For example, a system administrator may want some applications to skip read security checks, but not all applications.

Process-SPF provides the ability to disable security checks in specific applications or processes (Figure 3.1). For example, a system administrator may disable read security for a MPEG video player. Therefore, every time a video frame is read from disk, the security check of this read operation will be skipped. By skipping the read security checks, we may be able to improve the quality of the MPEG video being played. In this example, Process-SPF only disables the read security on the MPEG video player. Different security checks can be disabled in other applications or processes.

The Process-SPF design is independent of the relationship between different processes. In hardware, we now show a disk and a set of executable programs, labeled M through L. The executable programs on disk may or may not be the executable that created processes A, B, or C. The Process-SPF design is independent of this information.

The Process-SPF Configuration indicates what system call security checks, if any, a particular process is allowed to skip. As the Figure 3.1 indicates, the Process-SPF Configuration is not centrally located, like it is with System-SPF. The Process-SPF Configuration is stored in two different locations.

First, a Process-SPF Configuration is associated with each process running in the system. These are indicated in the diagram with Process-SPF Configurations A through C.

10.1 PROCESS-SPF ARCHITECTURE AND DESIGN
Second, a Process-SPF Configuration is associated with each executable file on disk. These are indicated in the diagram as Process-SPF Configurations M through L. As noted in the figure, each of the Process-SPF Configurations should be a specific Process-SPF Configuration for that executable or process.

10.2 OBJECT-SPF ARCHITECTURE AND DESIGN
Object-SPF provides system administrators with a different fine-grained technique to manage their system. Object-SPF provides system administrators the ability to disable security checks on individual objects in the system. Objects can refer to files, network connections, interprocess communication (IPC) objects, etc.

Let us consider the MPEG video player example from before. In order to improve MPEG video quality, we want to disable read security checks when a video is being played. However, some systems may have both publicly readable video and private videos that are not public. The system administrator may want to disable read security checks only on the public videos and not the private videos.

Object-SPF allows system administrators to designate individual files that should have security checks skipped. Similarly, Object-SPF allows system administrators to disable security checks on specific network connections, such as sockets, or IPC objects, such as message queues.

Since Object-SPF refers to individual objects in the operating system, Object-SPF can be discussed in terms of the different types of objects that can be implemented within an Object-SPF framework. We will only consider files, network connections, and IPC objects in this thesis. The design and implementation of SPF at each of these object levels will be referred to as:
1. File Security Performance Flexibility (File-SPF)
2. Network Security Performance Flexibility (Network-SPF)
3. Interprocess Communication Security Performance Flexibility (IPC-SPF)

The basic architecture of distributed trusted operating system (DTOS) and Flask is shown in Figure 4.0. Unlike the traditional approaches of adding security layers in the kernel, there are two additional subsystems in this architecture (Figure 4.0). The object manager's responsibility is to call the security server every time a user attempts to access an object. The security server checks the security policy configuration and informs the object manager if access is permitted or denied. Note that the security server is not part of the kernel. It is a separate module that can be called by the kernel. It can also be modified or replaced.

11. CONCLUSION
- The novel idea that not all security in a Trusted Operating System is necessary.
- The novel idea that skipping non-essential security checks in a Trusted Operating System may increase system performance.
- The novel SPF framework for Trusted Operating Systems that gives system administrators the ability to balance security and performance needs of a system.
- The novel recognition that the SPF framework can be implemented at different levels within a Trusted Operating System.
- The novel development of system administrative commands that allow system administrators to configure a Trusted Operating System dynamically as the system executes.

12. FUTURE SCOPE
The architecture for several QoS aware operating systems and Trusted Operating Systems are quite similar. For example, resource control lists, which are very similar to access control
lists. how QoS is put into an operating system by creating a separate file system that stores QoS information for each application in a system. This separate file system is similar to how SELinux creates a separate table to store security label information. There are indications that the architectural ideas from QoS and Trusted Operating Systems can perhaps be integrated into one system together. This leads to the possibility that several of these features can be combined and implemented into a single kernel.

REFERENCES