Information Security: Principles and Practice, Second Edition
by Mark Stamp

Chapter 13

Operating Systems and Security

UNIX is basically a simple operating system,
but you have to be a genius to understand the simplicity.
— Dennis Ritchie

And it is a mark of prudence never to trust wholly
in those things which have once deceived us.
— Rene Descartes

13.1 Introduction

In this chapter, we'll look at some of the security issues related to operating systems (OSs). OSs are large and complex pieces of software. Recall that in Chapter 12 we argued that there are almost certain to be security flaws in any large and complex computer program. But here we are concerned with the security protection provided by the OS, not with the very real threat of bad OS software. That is, we are concerned with the role of the OS as the security enforcer. This is a large topic that ties into many other aspects of security and we'll just barely scratch the surface.

First, we'll describe the primary security-related functions of any modern operating system. Then we'll discuss the notion of a trusted OS, and we'll conclude with a look at Microsoft's fairly recent effort to develop a trusted operating system, which goes by the catchy name of the Next Generation Secure Computing Base, or better yet, NGSCB.

13.2 OS Security Functions

An OS must deal with potential security issues whether they arise accidentally or as part of a malicious attack. Modern OSs are designed for multi-user environments and multi-tasking operations. Consequently, an OS must, at a minimum, deal with *separation*, *memory protection*, and *access control*. We briefly discuss each of these three topics below.

13.2.1 Separation

Arguably the most fundamental security issue for a modern OS is that of separation. That is, the OS must keep users and processes separate from each other.

There are several ways that separation can be enforced [235], including the following:

- *Physical separation* — Users are restricted to separate devices. This provides a strong form of separation, but it is often impractical.
- *Temporal separation* — Processes are separated in time. This eliminates many problems that arise due to concurrency and simplifies the job of the OS. However, there is a loss of efficiency.
- *Logical separation* — For example, each process might be allocated its own "sandbox." A process is free to do almost anything within its sandbox, but it can do almost nothing outside of its sandbox.
- *Cryptographic separation* — Crypto can be used to make information unintelligible to an outsider.

Of course, various combinations of these methods can be used.

13.2.2 Memory Protection

Another fundamental issue an OS must deal with is memory protection. This includes protection for the memory that the OS itself uses as well as the memory of user processes. A fence, or *fence address*, is one option for memory protection. A fence is a particular address that users and their processes cannot cross—only the OS can operate on one side of the fence, and users are restricted to the other side.

A fence could be static, in which case there is a fixed fence address. However, this places a strict limit on the size of the OS, which is a major drawback (or benefit, depending on your perspective). Alternatively, a dynamic fence can be used, which can be implemented using a fence register to specify the current fence address.

In addition to a fence, *base* and *bounds* registers can be used. These registers contain the lower and upper address limits of a particular user (or process) space. The base and bounds register approach implicitly assumes that the user (or process) space is contiguous in memory.

The OS must determine what protection to apply to a specific memory location. In some cases it might be sufficient to apply the same protection to all of a user's (or process's) memory. At the other extreme, *tagging* specifies the protection for each individual address. While this is as fine-grained protection as possible, it introduces significant overhead. The overhead can be reduced by tagging sections of the address space instead of each individual address. In any case, another drawback to tagging is compatibility, since tagging schemes are not in common use.

The most common methods of memory protection are *segmentation* and *paging*. While these are not as flexible as tagging, they're much more efficient. We briefly discuss each of these next.

Segmentation, as illustrated in Figure 13.1, divides the memory into logical units, such as individual procedures or the data in one array. Then appropriate access control can be enforced on each segments. A benefit of segmentation is that any segment can be placed in any memory location—provided the location is large enough to hold it. Of course, the OS must keep track of the locations of all segments, which is accomplished using `<segment,offset>` pairs, where the cleverly named `segment` specifies the segment, and the `offset` is the starting address of the specified `segment`.

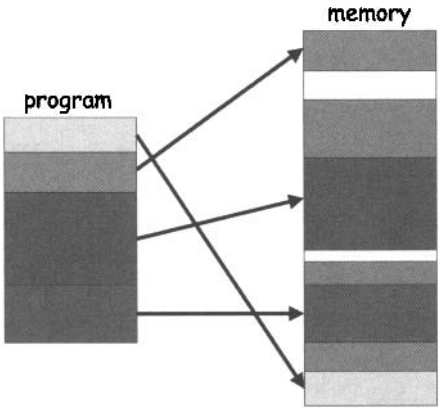

Figure 13.1: Segmentation

Other benefits of segmentation include the fact that segments can be moved to different locations in memory and they can also be moved in and out of memory. With segmentation, all address references must go through the OS, so the OS can, in this respect, achieve complete mediation. Depending

on the access control applied to particular segments, users can share access to some segments or users can be restricted to specific segments.

One serious drawback to segmentation is that the segments are of variable sizes. As a result, before the OS tries to reference any element of a given `segment` it must know the size of the segment so that it can be sure that the requested address is within the `segment`. But some segments—such as those that include dynamic memory allocation—can grow during execution. Consequently, the OS must keep track of dynamic segment sizes. And due the variability of segment sizes, memory fragmentation is a potential problem. Finally, when memory is compacted to make better use of the available space, the segmentation tables change. In short, segmentation is complex and places a significant burden on the OS.

Paging is like segmentation, except that all segments are of a fixed size, as illustrated in Figure 13.2. With paging, access to a particular page uses a pair of the form `<page,offset>`. The advantages of paging over segmentation include no fragmentation, improved efficiency, and the fact that there are no variable sizes to worry about. The disadvantages are that there is, in general, no logical unity to pages, which makes it more difficult to determine the proper access control to apply to a given page.

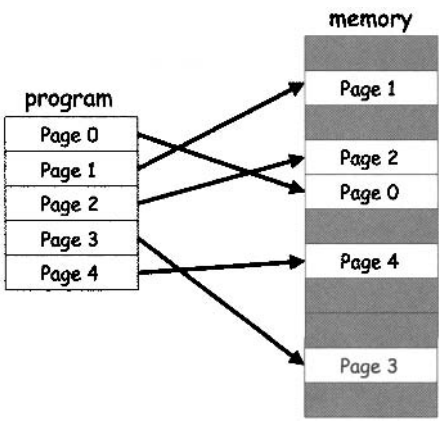

Figure 13.2: Paging

13.2.3 Access Control

OSs are the ultimate enforcers of access control. This is one reason why the OS is such an attractive target for attack—a successful attack on the OS can effectively nullify any protection built in at a higher level. We discussed access control in Chapter 8 and we'll briefly return to the subject in the next section when we discuss the concept of a trusted OS.

13.3 Trusted Operating System

There's none deceived but he that trusts.
— Benjamin Franklin

A system is *trusted* if we rely on it for security. In other words, if a trusted system fails to provide the expected security, then the security of the system is broken.

In this context, there is a distinction between trust and security. Trust implies reliance, that is, trust is binary choice—either we trust or we don't. Security, on the other hand, is a judgment of the effectiveness of a particular mechanisms. Security should be judged relative to a clearly specified policy or statement.

Note that security depends on trust. For example, a trusted component that fails to provide the expected level of security will break the overall security of the system. Ideally, we only trust secure systems, and all trust relationships are explicit.

Since a trusted system is one that we rely on for security, an untrusted system must be one that we don't rely on for security. As a consequence, if all untrusted systems are compromised, the security of the system is unaffected. A curious implication of this simple observation is that only a trusted system can break security. Hold this thought, since we'll have more to say about it in the next section.

What should a trusted OS do? Since any OS must deal with separation, memory protection, and access control, at a minimum, a trusted OS must do these things securely. Any list of generic good security principles would likely include the following: least privilege (e.g., the low watermark principle), simplicity, open design (e.g., Kerckhoffs' Principle), complete mediation, whitelisting (as opposed to blacklisting), separation, and ease of use. We might expect a trusted OS to securely deal with many of these issues. However, most commercial OSs are feature-rich, which tends to lead to complexity and poor security. Modern commercial OSs are not to be trusted.

13.3.1 MAC, DAC, and More

As mentioned above and illustrated in Figure 13.3, any OS must provide some degree of separation, memory protection, and access control. On the other hand, since we rely on a trusted OS for our security, it will almost certainly need to go beyond the minimal security operations. Specific security measures that we would like to see from a trusted OS likely include mandatory access control, discretionary access control, object reuse protection, complete mediation, trusted path, and logs. A trusted OS is illustrated in Figure 13.4.

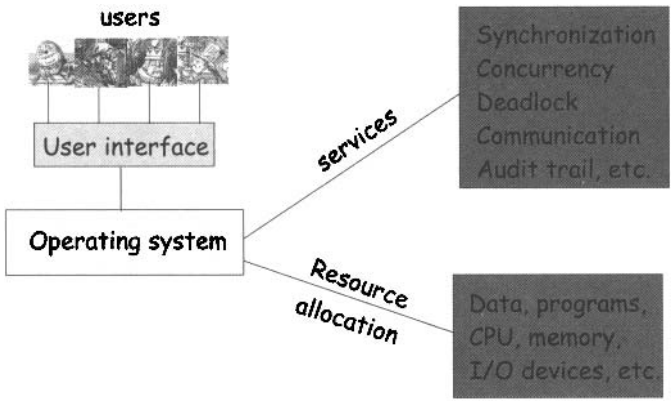

Figure 13.3: Operating System Overview

Mandatory access control, or MAC, is access that is not controlled by the owner of an object. For example, Alice does not decide who holds a TOP SECRET clearance, so she can't completely control the access to a document classified at this level. In contrast, *discretionary access control*, or DAC, is the type of access control that is determined by the owner of an object. For example, in UNIX file protection, the owner of a file controls read, write, and execute privileges.

If both DAC and MAC apply to an object, MAC is "stronger." For example, suppose Alice owns a document marked TOP SECRET. Then Alice can set the DAC since she owns the document. However, regardless of the DAC settings, if Bob only has a SECRET clearance, he can't access the document because he doesn't meet the MAC requirements. On the other hand, if the DAC is stricter than the MAC, then the DAC would determine the access.

A trusted OS must also prevent information from leaking from one user to another. Any OS will use some form of memory protection and access control, but we require strong protection from a trusted OS. For example, when the OS allocates space for a file, that same space may have previously been used by a different user's process. If the OS takes no additional precautions, the bits that remain from the previous process could be accessible and thereby leak information. A trusted OS must take steps to prevent this from occurring.

A related problem is *magnetic remanence*, where faint images of previously stored data can sometimes be read, even after the space has been overwritten by new data. To minimize the chance of this occurring, the DoD sets guidelines that require memory to be overwritten repeatedly with different bit patterns before it's considered safe to allow another process access to that space [132].

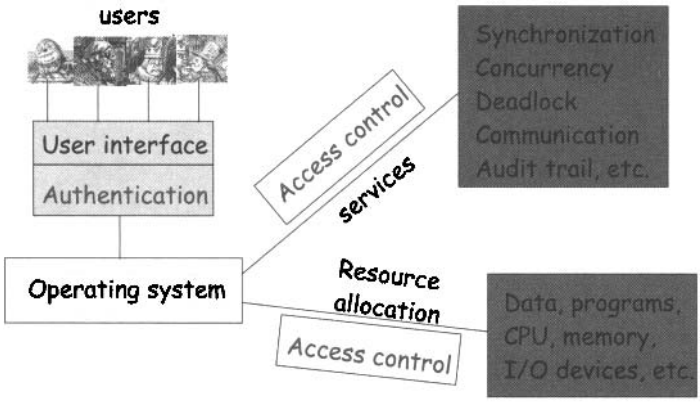

Figure 13.4: Trusted Operating System Overview

13.3.2 Trusted Path

When you enter your password at the login prompt, what happens to that password? We know what is supposed to happen to the password (hashed with a salt, etc.), but what actually happens depends on the software that is running on your system. How can you be sure that software is not doing something evil, such as writing your password to a file that will later be emailed to Trudy? This is the *trusted path* problem, and as Ross Anderson puts it in [14]:

> I don't know how to be confident even of a digital signature I make on my own PC, and I've worked in security for over fifteen years. Checking all of the software in the critical path between the display and the signature software is way beyond my patience.

Ideally, a trusted OS would provide strong assurance of a trusted path. If so, one benefit is that we could have confidence in a digital signature on a PC.

The OS is also responsible for logging security-related events. This sort of information is necessary to detect attacks and for postmortem analysis. Logging is not as simple as it might seem. In particular, it is not always obvious precisely what to log. If we log too much, then we might overwhelm any human who must examine the data, and we could even overwhelm automated systems that try to find the relevant needle in this haystack of data. For example, should we log incorrect passwords? If so, then "almost" passwords would appear in the log file, and log files would themselves be security critical. If not, it may be harder to detect when a password-guessing attack is in progress.

13.3.3 Trusted Computing Base

The *kernel* is the lowest-level part of the OS. The kernel is responsible for synchronization, inter-process communication, message passing, interrupt handling, and so on. A *security kernel* is the part of the kernel that deals with security.

Why have a dedicated security kernel? Since all accesses must go through the kernel, it's the ideal place for access control. It's also good practice to have security-critical functions in one location. By locating all such functions in one place, security functions are easier to test and modify.

One of the primary motivations for an attack on the OS is that the attacker can get below higher-level security functions and thereby bypass these security features. By putting as many security functions as possible at the OSs lowest layer, it may be more difficult for an attacker to get below these functions.

The *reference monitor* is the part of the security kernel that deals with access control. The reference monitor mediates all access between subjects and objects, as illustrated in Figure 13.5. Ideally, this crucial part of the security kernel would be tamper resistant, and it should also be analyzable, small, and simple, since an error at this level could be devastating to the security of the entire system.

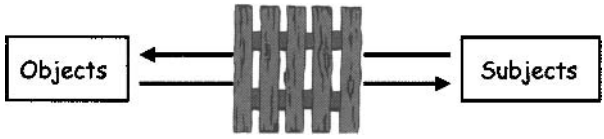

Figure 13.5: Reference Monitor

The *trusted computing base*, or TCB, is everything in the OS that we rely on to enforce security. Our definition of trust implies that, if everything outside TCB were subverted, our trusted OS would still be secure.

Security-critical operations will likely occur in many places within the OS. Ideally, we would design the security kernel first and then build the OS around it. Unfortunately, reality is usually just the opposite, as security tends to be an afterthought instead of a primary design goal. However, there are examples of OSs that have been designed from scratch, with security as a main objective. One such OS is SCOMP, which was developed by Honeywell. SCOMP has less than 10,000 lines of code in its security kernel, and it strives for simplicity and analyzability [116]. Contrast this to, say, Windows XP, which has some 40,000,000 lines of code and numerous dubious (from a security point of view) features.

Ideally the TCB should gather all security functions into an identifiable layer. For example, the TCB illustrated in Figure 13.6 is a poor design, since

security-critical features are spread throughout the OS. Here, any change in a security feature may have unintended consequences in other OS functionality, and the individual security operations are difficult to analyze, particularly with respect to their interactions.

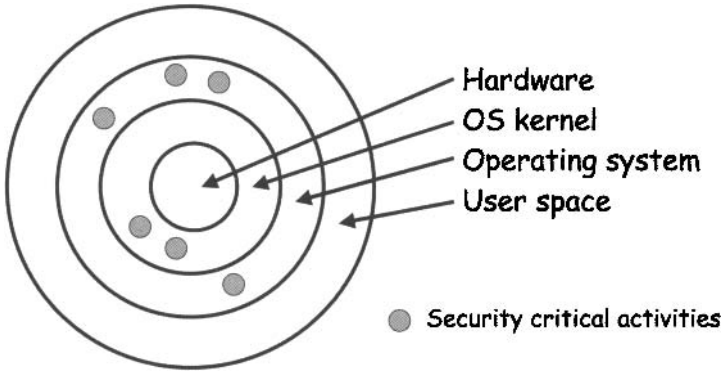

Figure 13.6: Poor TCB Design

The TCB illustrated in Figure 13.7 is preferable, since all security functions are collected in a well-defined security kernel [235]. In this design, the security impact of any change in one security function can be analyzed by studying its effect on the security kernel. Also, an attacker who subverts OS operations at a higher level will not have defeated the TCB operations.

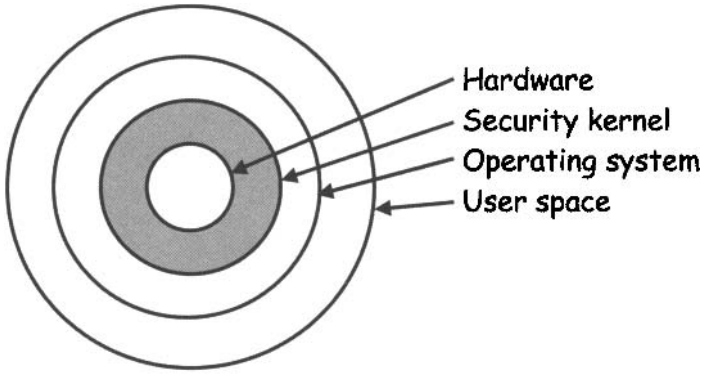

Figure 13.7: Good TCB Design

In summary, the TCB consists of everything in the OS that we rely on for security. If everything outside the TCB is subverted, we're still secure, but if anything in the TCB is subverted, then the security is likely broken.

In the next section we'll examine NGSCB, which is an ambitious effort by Microsoft to develop a trusted OS for the PC platform. DRM was the original motivation for NGSCB, but it has wide security implications [107].

13.4 Next Generation Secure Computing Base

Microsoft's Next Generation Secure Computing Base, or NGSCB (which is, strangely, pronounced "en scub"), was originally slated to be part of the "Longhorn" OS (i.e., Windows Vista). But it appears that most of the features of NGSCB won't appear until a later release, if ever.[1] Regardless, the concept is intriguing and it might yet find widespread application.

NGSCB is designed to work with special hardware, which is to be developed by the Trusted Computing Group, or TCG, led by Intel [306]. NGSCB is the part of Windows that will interface with the TCG hardware. TCG was formerly known as the Trusted Computing Platform Alliance, or TCPA, and NGSCB was formerly known as Palladium. It's been theorized that the name changes are due to bad publicity surrounding the initial discussion of TCPA/Palladium [190].

The original motivation for TCPA/Palladium was digital rights management. Due to the negative reaction this received, TCG/NGSCB now downplays the DRM connection, although it clearly remains a motivating factor. Today, TCG/NGSCB is promoted as a general security-enhancing technology, with DRM being just one of many potential applications. But, as we'll see below, not everyone is convinced that this is a good idea. Depending on who you ask, TCG/NGSCB—which is often shortened to TC—stands for "trusted computing" [219] or "treacherous computing" [13].

The underlying goal of TCG/NGSCB is to provide some of the strengths of a closed system on the open PC platform [102, 220]. Closed systems, such as game consoles and smartcards, are very good at protecting secrets, primarily due to their tamper-resistant features. As a result, closed systems are good at forcing people to pay money for the use of copyrighted information, such as game software. The drawback to closed systems is their limited flexibility. In contrast, open systems such as PCs offer incredible flexibility, but, as we have seen, they do a poor job of protecting secrets. This is primarily because open systems have no real means to defend their own software. Ron Rivest has aptly described NGSCB as a "virtual set-top box inside your PC" [74].

The TCG is supposed to provide tamper-resistant hardware that might someday be standard on PCs. Conceptually, this can be viewed as a smartcard embedded within the PC hardware. This tamper-resistant hardware

[1] Only one application of this technology appears to have been implemented so far. The "secure startup" feature in Vista and Windows 7 is said to use some features of NGSCB [204].

provides a secure place to store cryptographic keys or other secrets. These secrets can be secured, even from a user with full administrator privileges. To date, nothing comparable exists for PCs.

It is important to realize that the TCG tamper-resistant hardware is in addition to all of the usual PC hardware, not in place of it. To take advantage of this special hardware, the PC will have two OSs—its usual OS and a special trusted OS to deal with the TCG hardware. NGSCB is Microsoft's version of this trusted OS.

According to Microsoft, the design goals of NGSCB are twofold. First, it is to provide high assurance, that is, users can have a high degree of confidence that NGSCB will behave correctly, even when it's under attack. The second goal is to provide authenticated operation. To protect the secrets stored in the tamper-resistant hardware, it's critical that only trusted software can access the TCG hardware. By carefully validating (i.e., authenticating) all software, NGSCB can provide a high degree of trust. Protection against hardware tampering is not a design goal of NGSCB, since that is the domain of the TCG.

Specific details concerning NGSCB are sketchy, and, based on the available information, Microsoft has not yet resolved all of the fine points. As a result, the following information is somewhat speculative. The details might become clearer in the future.

The high-level architecture of NGSCB is illustrated in Figure 13.8. The "left-hand side," or LHS, is where the usual, untrusted, Windows OS lives, while the "right-hand side," or RHS, is where the trusted OS resides. The Nexus is the trusted computing base, or TCB, of the NGSCB. So-called Nexus Computing Agents, or NCAs, are the only software components that are allowed to communicate between the (trusted) Nexus and (untrusted) LHS [27]. The NCAs are a critical component of NGSCB—as critical as the Nexus.

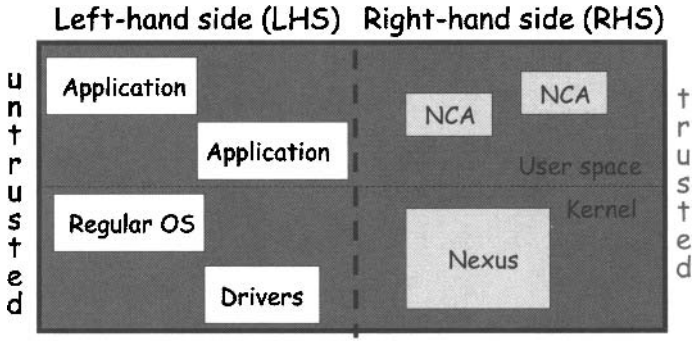

Figure 13.8: NGSCB Overview

13.4.1 NGSCB Feature Groups

NGSCB includes the following four major "feature groups."

- *Strong process isolation* — Prevents processes from interfering with each other.
- *Sealed storage* — The tamper-resistant hardware where secrets (that is, keys) can be securely stored.
- *Secure path* — Provides a protected path to and from the mouse, keyboard, and monitor.
- *Attestation* — A clever feature allows for "things" to be securely authenticated.

Attestation allows the TCB to be securely extended via NCAs. All four feature groups are primarily aimed at protecting against malicious code. Next, we'll describe each of these feature groups in a little more detail.

13.4.1.1 Process Isolation

Process isolation is enforced by "curtained memory," which appears to be little more than a buzzword. In any case, the trusted OS (the Nexus) must be protected from the untrusted OS as well as from the BIOS, device drivers, and other low-level operations that could be used to attack it. Curtained memory is the name for the memory protection scheme that provides such protection.

Process isolation also applies to the NCAs. The NCAs must be isolated from any software that they don't trust. These trust relationships are determined by users—to an extent. That is, a user can disable a trusted NCAs, but a user cannot make an untrusted NCA trusted. If the latter were possible, then the security of the trusted OS could be easily broken.

13.4.1.2 Sealed Storage

Sealed storage contains a secret, which is most likely a key (or keys). If software X wants to access the secret, as an integrity check, a hash of X is computed. The confidentiality of the secret is protected since it can only be accessed by trusted software while the integrity of the secret is assured since it resides in the sealed storage.

13.4.1.3 Secure Path

The details of the secure path feature are also vague. It's claimed that for input, the path from the keyboard to the Nexus and the path from the mouse

to the Nexus are both "secure"—but exactly how this is implemented is not entirely clear. Apparently, digital signatures are used so that the Nexus can verify the integrity of the data [302]. For output, there is a similar secure path from the Nexus to the screen, although here the signature verification would seem to be more exposed.

13.4.1.4 Attestation

The most innovative feature of NGSCB is attestation, which provides for the secure authentication of "things," such as devices, services, and, most importantly, software. This is separate from user authentication. Attestation is accomplished using public key cryptography, and it relies on a certified key pair, where the private key—which is not user accessible—lives in the sealed storage.

The TCB can be extended via attestation of NCAs. A new NCA is trusted provided that it passes the attestation check, which enables new applications to be added to an NGSCB system. This is a major feature, and we'll have more to say about it below.

One issue with attestation is that, since it uses public key cryptography, certificates must be exchanged. Since public keys reveal users' identities, anonymity is lost in this approach. To protect anonymity, NGSCB provides support for a trusted third party, or TTP. The TTP verifies the signature and vouches for it. Anonymity can be preserved in this way—although the TTP will know the signer's identity.

It is also claimed that NGSCB provides support for zero knowledge proofs. As we discussed in Chapter 9, zero knowledge proofs allow us to verify that a user knows a secret without revealing any information about the secret. According to Microsoft, when using zero knowledge proofs in NGSCB, "anonymity is preserved unconditionally" [27].

13.4.2 NGSCB Compelling Applications

What good is TCG/NGSCB? There are several compelling applications, but here we'll mention only two. First, suppose that Alice types a document on her computer. She can then move the document to the RHS (the trusted space), read the document carefully, then digitally sign the document before moving it back to the (untrusted) LHS. In this way, Alice can be confident of what she actually signed, which, as indicated by Ross Anderson's quote on page 497, is almost impossible on a non-NGSCB computer today.

A second application where NGSCB is useful is DRM. One fundamental problem that is solved by NGSCB is that of protecting a secret or key. In Chapter 12 we saw that it's impossible to securely protect a key in software.

By using tamper-resistant hardware (sealed storage) and other NGSCB features, protecting a key is much more plausible.

The NGSCB secure path also prevents certain DRM attacks. For example, with DRM-protected digital documents, an attacker could use a screen capture to scrape protected data from the screen. This would be much more difficult with the NGSCB secure path in place.

NBSCB also allows for the positive identification of users. Although this can be done without a trusted OS, there is a much higher degree of assurance with NGSCB, since the user's ID (in the form of a private key) is embedded in the secure storage.

13.4.3 Criticisms of NGSCB

Microsoft isn't evil, they just make really crappy operating systems.
— Linus Torvalds

According to Microsoft, everything you know and love about Windows will still work in the LHS of an NGSCB system. Microsoft also insists that the user is in charge, since the user determines all of the following:

- Which Nexus (if any) will run on the system
- Which NCAs are allowed to run on the system
- Which NCAs are allowed to identify the system

In addition, there is no way for an external process to enable a Nexus or NCA. This is to allay the fear that Microsoft would be in charge of an NGSCB computer. In addition, the Nexus code is open source. Finally, the Nexus does not block, delete, or censor any data—although NCAs do. For example, if a particular NCA is part of a DRM system, then it must "censor" any data for which user Alice has not paid. But each NCA on Alice's system must be authorized by Alice, so she could choose not to authorize the particular NCA that deals with DRM. Of course, she won't have access to DRM-protected content if she does not authorize the required NCA.

Microsoft goes to great lengths to argue that NGSCB is harmless. The most likely reason for this is that many people seem to be convinced that NGSCB is anything but harmless.

There are many NGSCB critics, but here we'll only consider two. The first is Ross Anderson, whose criticisms can be found at [13]. Anderson is one of the harshest TCG/NGSCB critics and perhaps the most influential. We'll then discuss the criticisms of Clark Thomborson, whose criticisms are less well known but raise some interesting fundamental issues [302].

Anderson's primary beef seems to be that when NGSCB is used, a digital object can be controlled by its creator, not by the user of the machine where it currently resides. For example, suppose Alice writes a book, *Bob in Wonderland*. With NGSCB, she can specify the NCA that must be used to access the digital form of this book. Of course, Bob can refuse to accept the NCA, but in that case his access is denied. And if Bob allows the NCA on his system, he may have restrictions placed on his actions (such as, he cannot use a screen capture, he cannot email the book, etc.).

It's worth noting that such restrictions are exactly what is needed in certain applications such as multilevel security (MLS). But Anderson's argument is that such restrictions are inappropriate as part of a general-purpose tool, such as a PC. Anderson gives the following simple example: suppose Microsoft Word encrypts all documents with a key that is only made available to Microsoft products. Then it would be even more difficult to stop using Microsoft products than it is today.

Anderson also claims that files from a compromised machine could be blacklisted (for example, to prevent music piracy). To illustrate this point, he gives an example similar to the following. Suppose that every student at San Jose State University (SJSU) uses a single pirated copy of Microsoft Word. If Microsoft blacklists this copy and thereby prevents it from working on all NGSCB machines, then SJSU students will simply avoid using NGSCB. But if Microsoft instead makes all NGSCB machines refuse to open documents created with this copy of Word, then SJSU users can't share documents with any NGSCB user. This could be a way to coerce SJSU students into using legitimate copies of Word.

Anderson makes some rather strange statements in [13], including the following:

> The Soviet Union tried to register and control all typewriters. NGSCB attempts to register and control all computers.

And there is an even more "interesting" statement:

> In 2010 President Clinton may have two red buttons on her desk—one that sends missiles to China and another that turns off all of the PCs in China...

Fortunately, this Orwellian prediction was way off the mark (in every respect). In any case, it's not clear to your usually paranoid author exactly how NGSCB would enable either scenario. Nevertheless, these are the kinds of concerns that an influential critic has raised.

Clark Thomborson has raised some issues that strike at the heart of the NGSCB concept [302]. In his view, NGSCB should be seen as a security guard. By passive observation, a real-world security guard can learn a great

deal about the workings of the facility he or she is guarding.[2] The NGSCB security guard is similar to a human security guard, in the sense that it can learn something about a user's sensitive information by passive observation.

So, how can Alice be sure that NGSCB is not spying on her? Microsoft would probably argue that this can't happen since the Nexus software is public, the NCAs can be debugged (as required for application development), and, besides, NGSCB is strictly an "opt in" technology. But there may be a loophole here. The release versions of NCAs can't be debugged and the debug and release versions will necessarily have different hash values. Consequently, the release version of an NCA could conceivably do something that the debug version does not do—such as spy on Alice.

The bottom line with regard to TCG/NGCSB is that it's an attempt to embed a trusted OS within an open platform. Without something similar, there is a legitimate concern that the PC may lose out, particularly in entertainment-related areas, where copyright holders might insist on the security of closed-system solutions.

NGSCB critics worry that users will lose control over their PCs—or be spied on by their PC. But it could reasonably be argued that users must choose to opt in, and, if a user does not opt in, nothing has been lost. So, what's the big deal?

However, NGSCB is a trusted system, and as we noted above, only a trusted system can break your security. When put in this light, NGSCB deserves careful scrutiny.

13.5 Summary

In this chapter, we considered operating system security and, more specifically, the role of a trusted OS. We then discussed Microsoft's NGSCB, which is an attempt to build a trusted OS for the PC platform. NGSCB has implications for many security-related fields, including digital rights management, a topic we covered in some detail in Chapter 12. NGSCB has its critics and we discussed some of their criticisms. We also considered possible counterarguments to the criticisms.

13.6 Problems

1. Expand and define each of the following acronyms: TCG, TCB, PITA, MAC, DAC, NGSCB.

[2]Recently, a former security guard at a major apartment complex took your author's class. This student confirmed that as a security guard, he learned a lot about the residents of the apartment complex, simply by passive observation. Your puritanical author would like to share some of these observations, but he cannot since this book is rated "G."

2. This problem deals with the definition of a trusted system.
   a. What does it mean to say that a system is "trusted"?
   b. Do you agree with the statement, "Only a trusted system can break your security"? Why or why not?
3. In this chapter we discussed segmentation and paging.
   a. What are the significant differences between segmentation and paging?
   b. Give one significant security advantage of segmentation over paging.
   c. What is the primary advantage of paging over segmentation?
4. Explain how paging and segmentation could be combined in one system.
5. This problem deals with mandatory access control (MAC) and discretionary access control (DAC).
   a. Define the terms mandatory access control and discretionary access control.
   b. What are the significant differences between MAC and DAC?
   c. Give two specific examples where mandatory access control is used and give two examples where discretionary access control is used.
6. Why would Trudy almost certainly prefer to subvert the OS rather than successfully attack one particular application?
7. In this chapter we briefly compared blacklisting and whitelisting.
   a. What is blacklisting?
   b. What is whitelisting?
   c. As a general security principle, which is preferable, whitelisting or blacklisting? Why?
   d. Which is likely to be more convenient for users, blacklisting or whitelisting? Why?
8. Recall that a trusted computing base (TCB) consists of everything in the OS that we rely on to enforce security. Which parts of NBSCB comprise its TCB?
9. In this chapter, a few compelling applications of NGSCB are mentioned, including "what you see is what you sign," digital rights management (DRM), and multilevel security (MLS). Discuss one additional compelling application of a trusted OS such as NGSCB.

10. Explain how NGSCB helps to solve some of the fundamental problems in digital rights management (DRM).

11. Explain how NGSCB helps to solve some of the fundamental problems in multilevel security (MLS).

12. A trusted OS, such as NGSCB, would make multilevel security (MLS) much more feasible. Given that this is the case, the military and government are likely to be interested in NGSCB. Why might businesses also be interested in NGSCB?

13. Some people believe that businesses will find NGSCB useful and that NGSCB will become commonplace in PCs as a result. If this is the case, then most PCs will eventually have a trusted operating system, but not because consumers find it particularly useful. Do you think this is likely to occur? Why or why not?

14. It is sometimes argued that digital rights management (DRM) is, in some sense, the modern incarnation of multilevel security (MLS).

    a. List some significant similarities between DRM and MLS.

    b. List some significant differences between DRM and MLS.

15. Suppose that you happen to have a secure multilevel security (MLS) system. Could this system be used to enforce digital rights management (DRM)?

16. Suppose that you have a secure digital rights management (DRM) system. Could this system be used to enforce multilevel security (MLS)?

17. This problem deals with NGSCB.

    a. What is attestation and what is its purpose?

    b. What are NCAs and what two purposes do they serve?

18. In the text, we mentioned two critics of NGSCB, namely, Ross Anderson and Clark Thomborson.

    a. Summarize Ross Anderson's criticisms of NGSCB.

    b. Summarize Clark Thomborson's criticisms of NGSCB.

    c. Which of these two critics do you find more compelling and why?

19. In Chapter 12, we discussed software reverse engineer. It's also possible to reverse engineer most hardware. Since this is the case, would DRM be any more secure on an NGSCB system than on a non-NGSCB system?

20. Give two real-world examples of closed systems. How well does each protect its software?

21. Give two real-world examples of open systems. How well does each protect its software?

22. Is each of the following an open system or a closed system? For each system, give an example of a real-world attack that has occurred.

    a. PC

    b. Cell phone

    c. iPod

    d. Xbox

    e. Kindle (an e-book reader)

23. Find an influential critic of NGSCB (other than the critics mentioned in the text) and summarize his or her arguments against NGSCB.

24. Find a supporter of NGSCB and summarize his or her arguments in favor of NGSCB.

25. Read the discussion of "treacherous computing" at [13] and summarize the author's main points.

26. Public key crypto is used in NGSCB for attestation. One concern with this approach is that anonymity might be lost. Recall that in Kerberos, Alice's anonymity is protected (e.g., when Alice sends her TGT to the KDC, she doesn't need to identify herself). Since anonymity is a concern, would it make sense for NGSCB to use an approach similar to Kerberos?

27. Why is the NGSCB sealed storage integrity check implemented using hashing instead of public key signing?

28. Why is NGSCB attestation implemented using digital signatures instead of hashing?

29. In NGSCB, how do each of the following help to protect against malicious software?

    a. Process isolation

    b. Sealed storage

    c. Secure path

    d. Attestation

30. Give two reasons why NGSCB attestation is necessary.

31. In NGSCB, each of the four "feature groups" is, apparently, necessary but not sufficient to ensure security. Discuss a specific attack that is difficult or impossible on an NGSCB system, but is easy when the specified feature group is missing.

    a. Process isolation

    b. Sealed storage

    c. Secure path

    d. Attestation

32. Explain Rivest's comment that TCG/NGSCB is like "a virtual set-top box inside your PC."

33. Suppose that students take in-class tests on their own laptop computers. When they finish answering the questions, they email their results to the instructor using a wireless Internet connection. Assume that the wireless access point is accessible during the test.

    a. Discuss ways that students might attempt to cheat.

    b. How could NGSCB be used to make cheating more difficult?

    c. How might students attempt to cheat on an NGSCB system?

34. Google's Native Client (NaCl) is a technology designed to allow untrusted code to run securely in a Web browser [332]. The primary advantage is speed, but there are many security issues, some of which are reminiscent of issues faced by NGSCB.

    a. Outline the NaCl security architecture.

    b. NaCl uses a "trampoline" to transfer control from untrusted code to trusted code. Explain how this works.

    c. Compare and contrast the security approach used in NaCl with each of the following: Xax, CFI, Active X.