Theory of Operating System Security

Ryan Scott

Grand Canyon University  
ITT-305 Information Security I

Prof. Scott Knapper

# Abstract

In this paper, we will discuss a few security vulnerabilities found in operating systems, which make them a prime target for cybercriminals. Important operating system terms will be defined, along with their advancements over time. There are many operating system features that can significantly impact security, and the importance of these features cannot be overstated! Lastly, we'll discuss rootkits, how they're combated, and why they're undetectable.

# 

# Operating Systems Terminology

## Enforced resource sharing

Users should be able to share resources as appropriate. It is essential to maintain integrity and consistency when sharing. It is possible to maintain the integrity and consistency of a resource using a lookup table combined with integrity controls such as a monitor or transaction processor, (Pfleeger et al., 2015).

## Guaranteed fair service

There is a general expectation that all users will receive comparable CPU usage and other services whenever they use the system. In this way, no user will be starved of services indefinitely (Pfleeger et al., 2015). Fairness is ensured by hardware clocks and scheduling disciplines, while data security is ensured by hardware facilities and data tables, (Pfleeger et al., 2015).

## Memory protection

A program run by a user must run in a secure and protected part of memory that cannot be accessed by unauthorized third parties. Besides preventing outsiders from accessing the program space, memory protection may also limit the access of a user to certain program areas, (Pfleeger et al., 2015). The memory space of a user may be subject to differential security, such as read, write, and execute. Memory protection is generally achieved by hardware mechanisms, including *paging* and *segmenting*, (Pfleeger et al., 2015).

# Advancements in Memory Management

## Fence

Single-user operating systems introduced the "fence" as the simplest form of memory protection to prevent faulty user programs from destroying critical aspects of the operating system (such as the kernel). In essence, a fence prevents access to certain areas by restricting users to one side of a boundary. As a result, it is generally guaranteed that operations issued before the fence will be executed before operations issued after it, (Pfleeger et al., 2015).

## Boundary Registers

In most cases, we cannot predict where programs will be loaded if there are more than one user. The relocation register solves this problem by providing a starting address as an anchor, (Pfleeger et al., 2015). An offset from this address is used for all other addresses in the program. In contrast to the fence register, which acts as a lower address limit, the bounds register acts as an upper address limit. As a result, the addresses of the programs of a given program are neatly sandwiched within the space between the base register and the bound register, (Pfleeger et al., 2015).

## Tagged memory

Occasionally, we may want to protect specific data values, but not all of them. This can be accomplished with tagged architecture, in which each word in machine memory has a single bit added to it to identify who has access to it. By setting access bits only with privileged instructions (from the operating system), security can be ensured, (Pfleeger et al., 2015). Every time a program attempts to access this location, these bits are tested.

## 

## Segmentation

As the name implies, segmentation refers to the separation of a program into pieces. The pieces are logically linked, displaying a relationship between their code or data. Segmentation was developed as a practical solution to mimic an unlimited number of base bound registers, (Pfleeger et al., 2015). Essentially, the segmentation of a program enables it to be divided into various parts with different rights of access. This also allows hardware to regulate access to different memory sections in a variety of ways, (Pfleeger et al., 2015).

## Paging

A worthwhile substitute to segmentation is paging. A paging system divides the program into equilivent chunks called pages, and the memory is divided into identical units called page frames, (Pfleeger et al., 2015). Every page in the paging approach is of the same fixed size, making fragmentation a problem of the past. Using paging, you are able to combine the security advantages of segmentation with a more efficient memory management technique, (Pfleeger et al., 2015).

# Securing Operating Systems

## Layered Privilege Processes

Security operating systems often have a layered structure which is reminiscent of the layers of an onion, with the most sensitive operations located at the center. As a result, the trustworthiness of a process can be determined by its proximity to the center; typically, the more trusted processes are those that are located closest, (Pfleeger et al., 2015). The use of layers is another method of achieving *encapsulation*, and is generally regarded as a wise approach when it comes to operating system design, (Pfleeger et al., 2015). As each layer provides a certain level of functionality to the layers further out, damage control and limited access are provided by the more central layers. In addition, the layers farther out provide certain levels of functionality to the more central layers, (Pfleeger et al., 2015).

## Virtualization

Virtualization plays a critical role in operating system security by creating the illusion of a single set of resources while simultaneously using a variety of different resources at the same time, (Pfleeger et al., 2015). This is achieved by presenting the system as containing only the resources that the user is authorized to access. The benefits of virtualization go beyond just security, but that is beyond the scope of this paper, (Pfleeger et al., 2015).

## Sandbox

Sandboxes offer a secure environment where programs can run without harming anything else on the machine. This is accomplished by segregating running programs, typically in an effort to avoid the spread of system failures as well as software vulnerabilities, (Pfleeger et al., 2015). When a process is run in a sandbox, the impact on external resources is limited and controlled.

# Fundamentals of Rootkits

**Rootkits are malicious programs embedded beneath an operating system.** These malicious pieces of code reside between the operating system and the hardware. With the rootkit in this position, it has the ability to gain root access, (Pfleeger et al., 2015). Admins (or root) are the most powerful users in UNIX systems. They have access to sensitive resources such as memory and are able to perform powerful actions such as creating users and killing processes. With root, a malicious actor could easily bypass, disable, or alter the functionality of the operating system, (Pfleeger et al., 2015). The threat of rootkits to the security of computer systems cannot be understated!

***Rootkits remain invisible by effectively integrating with the OS kernel.*** Due to the fact that rootkits are integrated into the operating system, they can carry out any function that your operating system can perform completely undetected, (Pfleeger et al., 2015). Since rootkits run with these privileges, they are automatically loaded upon OS startup. While the operating system is responsible for audit logging, a rootkit might be able to fail to log its own activities. As a result, the rootkit can remain undetected and perform actions without restriction, (Pfleeger et al., 2015).

***Anti-rootkit tools require the elusive rootkit to first be identified.*** This can be done by taking a page out of the rootkit’s own playbook: interception of file directory enumeration functions, (Pfleeger et al., 2015). A low-level rootkit revealer can help compare file sizes and detect discrepancies, which can lead you straight to the rootkit. By identifying a rootkit, the penetration can be patched. This is a very risky and abandoned security strategy, as it usually opens up major security vulnerabilities, (Pfleeger et al., 2015). Today, rootkits and malware are constantly evolving, and antivirus software keeps a list of both old and new ones to scan for. In spite of its simplicity, ease, and painlessness, it cannot be denied that the process is eerily similar to that of a rootkit itself, (Pfleeger et al., 2015).

# References

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). Operating Systems. Security in Computing (5th ed., pp. 280–340).