Editors: Elias Levy, aleph1@securityfocus.com
Iván Arce, ivan.arce@coresecurity.com

# The Kernel Craze

Iván Arce
*Core Security Technologies*

The kernel is a fundamental piece of the operating system that provides and mediates access to a computer system's resources. Naturally, such a critical component plays a key role in providing users a secure environment and should be subject to security practitioners' scrutiny.

Although kernel bugs have had serious security implications for the past three decades, researchers have not studied their importance or the mechanics for discovering, exploiting, and fixing them as much as they've focused on other information technology infrastructure components. Yet, over the past several years, an increasing number of kernel security vulnerabilities have demonstrated that the most important component of today's operating systems is not free of bugs—when these vulnerabilities are exploited, serious security incidents ensue.

In this installment of Attack Trends, I focus on kernel security issues, recent vulnerabilities, and the emergence of publicly available exploit code for them.

## Kernel overview

An operating system's principal function is to provide an execution environment in which users' programs run. This requires a basic framework for uniform program execution with a uniform and standardized way to use the hardware and access system resources in a coordinated and orderly manner. The kernel provides this basic service in all but the most simplistic operating systems.

To provide these fundamental capabilities to the operating system, several portions of software initialize and run at system boot time. Typically, each piece implements specific functions and exposes a well-defined programming interface to other portions of code that might use them. This outlines a view of the operating system as a layered set of components in which each layer fulfills specific requirements.

Ultimately, applications interact with the user—or other applications—and with hardware resources via programming interfaces made accessible by each layer of code.

Functional layers found in modern kernels include:

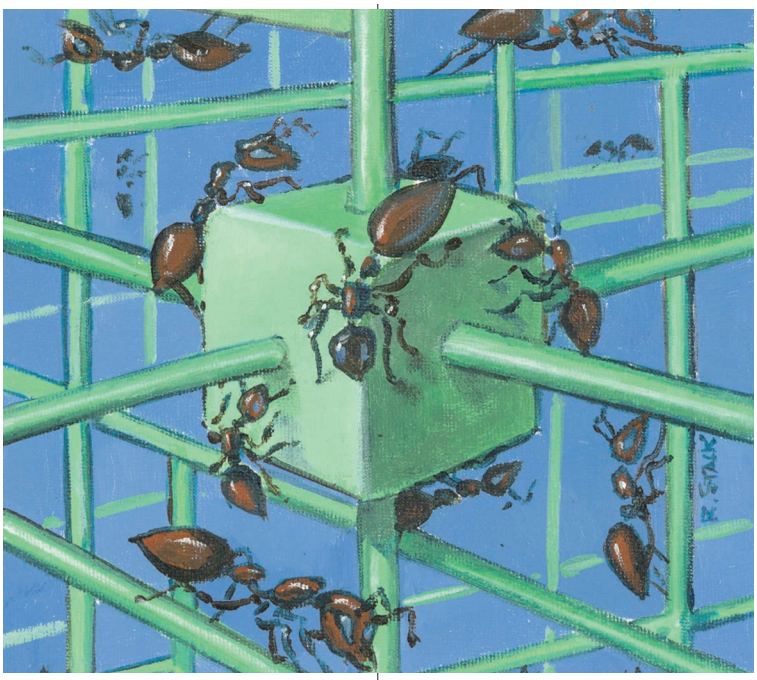
- device drivers and a hardware-abstraction layer,
- process- and thread-execution framework,
- process scheduling,
- memory management,
- file-system services,
- networking services, and
- security services

Design and implementation decisions made for developing today's kernels follow a small set of paradigms that produce different strains of operating systems based on monolithic, micro, and exo kernels. Although their differences are quite important from an engineering perspective, we won't delve into them here. For the purpose of this department, it suffices to say that even the minimal set of services that any kernel provides is complex enough to justify a closer look in the context of information security threats.

## Kernel and user modes of execution

Kernel software requires full and unrestricted access to all system components to operate and arbitrate concurrent requests from user applications to resources. To accomplish orderly program execution, a simple model provides two separate environments for code execution: Kernel code runs in a privileged execution environment with almost complete control of the computer system and without any security restrictions. This is often called *privileged* or *kernel mode*. In the other environment, called the *user mode*, the kernel code mediates system privileges, allowing a restricted execution environment for user applications. To acquire and use system resources or modify its own execution environment, a user-mode application communicates with the operating system's kernel via a well-defined channel, called the *system calls* interface.

Many system administrators and other IT practitioners generally regard the kernel as a *black-box* component in the operating system—a closed system whose internal workings do not need to be understood—and expect it to work in an efficient, reliable, and secure manner. Administrators generally resort to kernel configuration tweaking and optimization as a way to increase system perfor-



mance, solve efficiency and reliability problems, or address underlying hardware requirements, such as adding or modifying hardware devices.

To the security-conscious reader, my simplistic kernel overview might have already triggered concerns due to the kernel software's complex and critical nature. It is an accepted principle that simpler and smaller code is more easily scrutinized and less prone to software bugs.

Yet, the functional requirements for a modern kernel foster an increase in code complexity and size. We should not assume that any of the different layers of code running in kernel mode, including the system-calls interface, are free of programming errors.

## Impact of kernel bugs

The existence of a bug in the kernel is of utmost importance to security-conscious users due to its key role in the operating system. Kernel bugs can affect the entire operating system's stability and operations and, therefore, have a direct impact on all running applications.

Also, because code running in kernel space has direct and unrestricted access to all low-level system resources, a kernel bug provides a direct avenue of attack that bypasses all security controls in most general-purpose operating systems. From an information-security perspective, the kernel is the last line of defense; a bug discovered in it implies that, if the vulnerability is exploited, all security mechanisms are completely breached.

After a kernel is compromised, we can argue that we still rely on certain, very specific security mechanisms and features implemented at the hardware level, but for all practical purposes, an attacker will now have full reign over the computer system.

## Kernel bug exploitation

Because executing the kernel code correctly is directly related to the operating system's reliability and trustworthiness, exploiting kernel bugs is an extremely sensitive endeavor:

- Common development tools such as debuggers, compilers, and disassemblers must be used in a constrained manner to avoid system instability and crashes. This leads to a cumbersome and tedious development process with no room for programming errors in the exploit code: a failed exploit yields a crashed system.Components running in kernel space are tightly coupled and often have complex interactions; badly coded exploit code results in erratic and unexpected system behavior. This requires that the exploit developer have a thorough understanding of kernel operations, data structures, and internal algorithms.
- The exploit code's reliability is of outmost importance because the exploit is not only required to function properly, but to do so in a variety of system configurations and operational conditions. A trial-and-error approach is unsuitable because it makes the attack evident owing to continuous repeated system crashes and noticeably disrupted operations.
- Simple exploitation does not easily yield directly usable benefits in the kernel space. Because all operating system interactions with humans or external entities are carried via applications running in user mode, a kernel exploit must transfer its newly acquired privileges to a user-mode program.

These technical barriers will hardly deter a skilled and decided attacker.

Most exploit code found in the wild for kernel vulnerabilities does not go beyond triggering a denial-of-service attack against the system due to system resource exhaustion or system crashes. However, well-thought-out exploits, carefully developed using clever tricks to gain and maintain elevated privileges through kernel vulnerabilities on compromised systems, have emerged in the past few years.

Additionally, the kernel space is the best-suited portion of the operating system for an attacker to hide malicious programs that provide illegitimate access to system resources because it hides their activities from the rest of the operating system; this is referred to as *kernel-level rootkits*.

## A history of kernel vulnerabilities and exploits

An exhaustive account of the past 30 years of kernel vulnerabilities and exploits would probably consume a large portion of this issue of *IEEE Security & Privacy* and bore all but our most kernel-fanatic readers, so I will resort to pinpointing the more relevant references in this matter. The reader is left with the exercise of connecting the dots and adding more data points that lead to a more accurate extrapolation of trends in kernel vulnerability and exploitation.

We can track the first account of a kernel exploit to the US Air Force's security evaluation of the Multics operating system, published in 1974 ("Multics Security Evaluation: Vulnerability Analysis"; http://csrc.nist.gov/publications/history/karg74.pdf). In their report, authors Paul Karger and Roger Schell detailed a kernel vulnerability in the Multics operating system, followed by a step-by-step procedure on how to exploit the flaw and force the operating system and its underlying microprocessor to run unprivileged code in privileged mode.

Eugene Spafford illustrated the outcome of letting possibly malicious users mangle arbitrary portions of kernel memory in his account of an easy way to elevate privileges on buggy Unix systems. In it, the systems let user-mode programs execute a divide-by-zero operation—an erroneous operation that must be caught and dealt with carefully—and store the results in kernel memory (see http://securitydigest.org/core/archive/122).

The Last Stage of Delirium (LSD), a Polish research group, has studied the exploitation of kernel-level vulnerabilities in modern Unix operating systems that run on the Intel x86 family of microprocessors. Their report of their participation in a hacking challenge in 2001 provides a complete description of several kernel bugs and the exploit code they developed to successfully breach the security mechanisms of a security-hardened operating system (see "Kernel-Level Vulnerabilities: Behind the Scenes of the 5th Argus Hacking Challenge"; www.lsd-pl.net/documents/kernvuln-1.0.2.pdf).

Perhaps the most widely known and publicized kernel bug is the one leading to the infamous Ping of Death attacks that were rampant on the Internet in the mid 1990s. A bug in the kernel networking code of almost all operating systems then in use allowed attackers to remotely crash vulnerable computers—the sole requirement was to send a maliciously crafted IP datagram with an ICMP Echo Request ("ping") payload (see http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci822096,00.html).

In "More Bang for the Bug: An Account of 2003's Attack Trends" (*IEEE Security & Privacy*, vol. 2, no. 1, 2004, pp. 66–68), I gave an account of attackers' recent exploitation of a Linux kernel bug in their attempt to introduce backdoors in one of the most widely used Linux distributions.

A quick search on MITRE's Common Vulnerabilities and Exposures (CVE) directory (cve.mitre.org) results in 117 kernel-level vulnerabilities reported since 1999. As well, exploit code to elevate privileges on vulnerable systems with kernel-level bugs catalogued in Symantec's SecurityFocus vulnerability database (www.securityfocus.com/bid/keyword/) has been publicly available for at least 6 of the 12 kernel vulnerabilities reported since December 2003.

The proliferation of new types of hardware devices and the advances in microprocessor, storage, and networking technologies call for increasingly complex kernels. As I mentioned, security practitioners accept that code size and complexity grow to the detriment of security; the keep-it-simple principle applies to the kernel as well as—or even more than—to any other software program.

There is no temporary fix or workaround for kernel bugs. Usually, the only cure is to fix and replace the current kernel with one that is patched. Fixing kernel security bugs is both an unrewarding and critically sensitive activity that can often go wrong, with a possibly serious impact on network operations due to system outages, performance degradation, or erratic system behavior. However, remaining oblivious to recent advances in kernel vulnerability exploitation or underestimating would-be attackers' ingenuity and determination of is a perfect recipe for an information-security disaster.

Security problems in kernel code are difficult to spot, understand, and fix. We must address kernel security throughout the entire development process. Although operating-system vendors should place their most substantial efforts in the early stages of design and development, IT and security practitioners must prepare practical strategies to effectively address kernel security issues in operational environments. □

**Remaining oblivious to advances in kernel exploitation is a perfect recipe for an information-security disaster.**

***Iván Arce*** *is chief technology officer and cofounder of Core Security Technologies, an information security company based in Boston. Previously, he worked as vice president of research and development for a computer telephony integration company and as information security consultant and software developer for various government agencies and financial and telecommunications companies. Contact him at ivan.arce@coresecurity.com.*