User vs Kernel Mode: How Operating Systems Protect Web Applications

Samantha Geoghegan
Department of Computer Science and Physics
Rider University
Lawernecville, New Jersey

geoghegans@rider.edu

Akacia Itasha Howell
Department of Computer Science and Physics
Rider University
Lawerencville, New Jersey
howellak@rider.edu

***Abstract*— Modern web applications operate within operating systems that enforce strict separation between user mode and kernel mode to protect system resources. This dual mode architecture is fundamental to maintaining stability and security. It is important that you have protection against unauthorized access to memory, files, and hardware devices. As web applications handle sensitive data and complex operations, understanding how operating systems implement these boundaries becomes essential for cybersecurity.**

This project examines how the separation between user mode and kernel mode protects web applications from accessing restricted system resources. It further investigates the security risks that arise when applications attempt to pass certain restrictions, including protected documents and unauthorized memory access. With a conceptual analysis of operating system architecture, system calls, and real-world examples such as browser sandboxing and malware exploitation, it explores how operating systems control these risks through access control, permission enforcement and restricted system calls. The findings aim to clarify the role of OS-level protections in limiting damage even when web applications are compromised

Keywords—Operating Systems, user mode, kernel mode, web application security, system calls, privilege escalation, sandboxing

I. Introduction

Modern operating systems have a strict separation between user mode and kernel mode in order to maintain system stability and to protect resources. In user mode, applications operate with restricted privileges, while kernel mode allows full access to hardware and system memory. [1] This dual-mode execution model ensures that user-level programs cannot directly manipulate sensitive system components or interact with other processes.

II. System Architecture And Process Flow

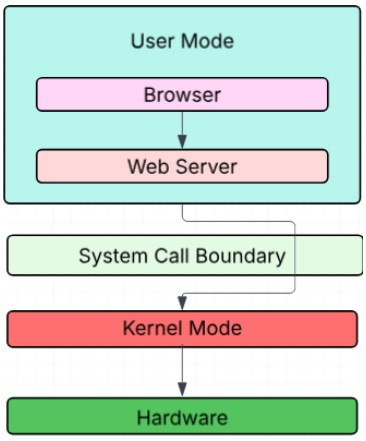
A. User Mode and System Calls

Web Applications and Server execute as user mode processes within a dual mode operating system architecture. In user mode, access to resources is restricted. The kernel usually runs in privileged mode while applications run in user mode. [2] When a web application requires access to system resources, such as reading files or allocating memory, it must issue a system call. System calls are implemented by using a software interruption mechanism of the CPUs that can raise a privilege level in a safe and restricted way. [3] During a system call, the operating system validates the request, making sure it does not give restricted information to unprivileged parties, and enforces access control. Afterwards, the operation is executed on behalf of the application before returning to user mode.

B. Flow Explanation and Architectural Limitations

The interaction follows a structured flow:

Figure 1: System Call Boundary Flow



The web application requests a system call, and it goes past the System Call Boundary or Interface. As shown in Figure 1, it accesses the kernel and the hardware following while it sees if the information requested is not restricted, then sends it back to the web application and back into user mode. This model ensures that web applications cannot directly manipulate system memory, change kernel data structures, or execute restricted instructions. By enforcing the boundary, the operating system isolates application-level processes and limits the potential damage caused by malicious code. The separation between user and kernel mode serves as a fundamental protection mechanism for modern web applications.

III. Security Implications

This research project implements layered operating system security principles through sandboxing, virtualization, and monitoring tools such as Virtual Machines, Wireshark, and Process Monitor (Procmon). This project demonstrates how malware-like behavior interacts with operating system resources such as the file system and registry, showing that user-mode processes rely on kernel-mediated system calls to modify protected components. As the Chromium security architecture explains, “Compromising a renderer process does not grant access to the underlying system” [6], illustrating how sandboxing limits damage from untrusted code. Similarly, User-Mode Linux reinforces that restricting

privileged execution confines failures to user space rather than the host kernel [4], and snapshot restoration further confirmed hypervisor-level containment by reverting the system to a clean state.

IV. Methodology

In this project, we first start by creating a web application to be able to demonstrate what system calls look like in the kernel and how they truly work. When creating the server, we used JavaScript or js with an addon called Node.js. It allows ofr us to make a web server to allow for a live demonstration of how system calls are executed. Within the code, we create a simple HTTP server that reads a simple sentence when that file is retrieved from the system. The way it is done is with the fs.readFile() command where there is a secret file name looked for in the systema and if unrestricted, allowed to be given to the web server or with the right credentials, given to the web server. Once online and shown to be working we run this command:

strace –e trace=open,read,write,sendto node server.js

When this is run, it shows all the system calls that are made in accordance to the website. We see all the read() calls as well as the write() calls when its file is written and sent back to the web server to then display the files contents. To also demonstrate what would happen when a file is requested but is protected, another js code was written to simulate that in real time. While in the successful file retrieve, we had a file that was 'secret.txt', in this denied demonstration, we put a new file called 'secret_admin.txt' within the root of the computer by running:

sudo touch /root/secret_admin.txt

sudo chmod 600 /root/secret_admin.txt

Which then puts the file into the root. The command 'chmod 600' makes it so that only the root can read and access the

file. So, when requested like when done for the normal server but instead we run this for the denied example:

strace –e trace=open,read,write,sendto – node server-denied.js

It shows the attempts to read the file and is then terminated with an "Access Denied" on the web server as the file is protected and not allowed to be read. This happens because the file is in the /root directory, which is restricted to only administrators. In Linux, normal users do not have permission to read or modify files inside the /root, which protects critical system data from unauthorized access. This security model helps protect your computer from malicious programs or websites that might try to access sensitive system information without permission.

The next section of this project was conducted within a controlled virtual machine environment to safely simulate malware-like behavior while preventing impact on the host system. A Windows virtual machine was configured and isolated using virtualization and sandboxing principles. Simulated persistence techniques were executed to interact with operating system resources such as the file system and registry, demonstrating how user-mode processes rely on kernel-mediated system calls to modify protected components. Process Monitor (Procmon) was used to monitor file system and registry activity in real time, allowing observation of system calls and permission enforcement. Wireshark was used to capture and analyze network traffic to determine whether the simulated behavior attempted external communication. After executing the persistence behavior, the system was powered off, and a previously created snapshot was restored. This restoration reverted all changes to a clean state, confirming hypervisor-level containment and demonstrating that virtualization effectively isolates and limits the impact of potentially malicious activity.

References

[1] Sharma, Lalitsen. "Analysis of Delivery of Web Contents-Mode and User-Mode Web Servers." Internation Journal of Computer Applications, Foundation of Computer Science, 2011. https://www.academia.edu/49077478/Analysis_of_Delivery_of_Web_Contents_for_Kernel_mode_and_User_mode_Web_Servers

[2] Behnam, Moris. Implementation of Overrun and Skipping in VxWorks. 2010. https://www.academia.edu/2738325/Implementation_of_overrun_and_skipping_in_VxWorks#page=6

[3] Maeda, Toshiyuki, and Akinori Yonezawa. "Kernel Mode Linux: Toward an operating system protected by a type theory." Annual Asian Computing Science Conference. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[4] Dike, J. (2000). *A secure operating system for Linux*. In Proceedings of the 4th Annual Linux Showcase & Conference (ALS 2000). USENIX Association. https://www.usenix.org/legacy/publications/library/proceedings/als00/2000papers/papers/full_papers/dike/dike.pdf

[5] S. Sultan, I. Ahmad and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," in IEEE Access, vol. 7, pp. 52976-52996, 2019, doi: 10.1109/ACCESS.2019.2911732.

[6] Yinzhi Cao, Zhichun Li, Vaibhav Rastogi, Yan Chen, and Xitao Wen. 2012. Virtual browser: a virtualized browser to sandbox third-party JavaScripts with enhanced security. In Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS '12). Association for Computing Machinery, New York, NY, USA, 8–9. https://doi.org/10.1145/2414456.2414460