

# Operating Systems Security in Sandboxing and Isolation

## Mechanisms

Tech Paper Report

CS 4310 Operating Systems

Prof. Gilbert Young

Devin Khun

## Abstract

Modern operating systems rely heavily on sandboxing and process isolation to limit application privileges and reduce the impact of attacks. This paper compares how Linux, Windows, and macOS implement these mechanisms through tools such as AppArmor, SELinux, seccomp-bpf, AppContainer, Defender Application Guard, the macOS Sandbox, and System Integrity Protection. The focus is on how each platform confines applications, enforces permissions, and protects against threats like malware, privilege escalation, and kernel-level exploits. The paper also evaluates the strengths and weaknesses of these models and discusses how recent vulnerabilities and emerging threats shape the direction of OS security design.

## Introduction

Operating systems have grown into some of the most complex pieces of software we rely on every day. As they've evolved, so have the threats that target them. Malware, privilege-escalation attacks, and kernel exploitation techniques keep getting more sophisticated, pushing OS designers to rethink how much freedom applications should have by default. One of the biggest shifts in this space has been the move toward stricter isolation. Instead of assuming applications will behave, modern systems try to contain them.

Sandboxing and process isolation have become core strategies for making that happen. These mechanisms limit what a program can access, even if it becomes compromised, and they create clear boundaries between user applications and critical system components. Every major operating system approaches this idea differently, balancing security, performance, and usability in its own way. By comparing each operating system's sandboxing frameworks, permission

models, and real-world behavior under attack, we can better understand the strengths and tradeoffs behind each design.

## Discussion

### Background

Modern operating systems handle everything from everyday applications to critical system processes, which makes them a prime target for attackers. As threats have grown more complex, OS security has shifted toward limiting how much any single process can do by default. Zarif points out in *Securing Operating Systems (OS): A Comprehensive Approach to Security with Best Practices and Techniques*, OS defenses now rely on layered controls because traditional permissions alone cannot stop modern malware, memory corruption attacks, or privilege-escalation techniques (Zarif 2024). Sandboxing and process isolation address this by creating controlled environments where applications run with tightly scoped privileges. Even if an attacker compromises a process, isolation makes it much harder for the malware to break out, reach sensitive files, or interact with the kernel. This shift toward confinement has become a central theme in OS design, influencing Linux distributions, Windows security models, and macOS's hardware-backed protections.

### Sandboxing in Linux

Linux uses a combination of mandatory access control frameworks and low-level kernel mechanisms for confinement. Linux has the most diverse and customizable set of sandboxing technologies, which makes it powerful but also more complex to configure. AppArmor uses human-readable profiles that define what a program is allowed to access. It's straightforward to deploy and works well for limiting filesystem access and network interactions. Many

distributions, including Ubuntu, use AppArmor by default because it balances usability and security while keeping rule creation manageable. SELinux applies a mandatory access control (MAC) model with a dense set of policies that control interactions between processes, files, sockets, and kernel resources. It's extremely granular and well-suited for enterprise environments, but it's also known for being difficult to configure. SELinux enforces tight boundaries that limit how far an exploited process can go. According to Niemi, real-world analysis shows that Linux sandboxes are strong when configured properly, but their flexibility also makes them prone to misconfigurations. Memory forensics research further highlights how attackers often rely on gaps in policy enforcement to move laterally through the system. This makes Linux's biggest strength its variety of tools, but the flexibility comes with a cost.

### **Sandboxing in Windows**

Windows uses a mix of integrity levels, restricted tokens, and container-style isolation to limit application behavior. AppContainer is the core sandbox for modern Windows applications. It gives each application a restricted token, a private namespace, and tightly scoped capabilities. A compromised AppContainer process can only access resources the developer explicitly allowed. This model is widely used for Microsoft Store apps and certain browser components. Windows Defender Application Guard (WDAG) isolates the browser in a lightweight virtualized environment. It's designed to contain high-risk tasks, such as opening untrusted websites. Since WDAG uses a Hyper-V micro-VM, even a full browser compromise ends up isolated from the host OS. In addition, Windows has the typical mandatory integrity control that assigns trust levels to processes so that a lower-integrity process cannot modify higher-level objects. Restricted tokens further cut down what a process can access, reducing the damage malware can cause. Windows isolation is strong when applications run inside AppContainer or WDAG, but

older software that relies on classic APIs does not always benefit from this. According to Ligh (2014) in *The Art of Memory Forensics*, Windows malware strains abuse legacy subsystems or memory injection techniques that bypass weakly isolated processes. Regardless, Windows still offers some of the strongest containment when modern features are fully used.

### **Sandboxing in macOS**

macOS relies heavily on system-wide policies, hardware support, and stricter defaults compared to Linux and Windows. The macOS Sandbox uses rule-based profiles enforced by the kernel. Apps from the App Store must include entitlements that define what they're allowed to access. The model is non-negotiable, which gives macOS a consistent security baseline on all devices. System Integrity Protection (SIP) protects core system files, kernel extensions, and memory regions from modification, even by root. This blocks entire classes of privilege-escalation attacks and forces malware to rely on less direct paths. Recently, Microsoft Threat Intelligence found that CVE-2024-44243 lets attackers bypass System Integrity Protection by abusing vulnerable third-party kernel extensions (Microsoft Threat Intelligence 2025). Their analysis showed that once the extension was loaded, it exposed unsafe interfaces that allowed malicious code to modify protected system areas that SIP is supposed to lock down. The case makes it clear that even strong protections like SIP can be undermined when the kernel trusts external components that do not enforce the same security guarantees. macOS requires hardened runtime flags for many apps and expects software to be notarized by Apple. These steps ensure that apps run with predictable behavior and reduce the chances of injected code running unchecked. Even though macOS has strong defaults, its sandbox is less customizable than Linux's. macOS breaches rely on kernel vulnerabilities or logic flaws rather than policy

misconfigurations. When those flaws appear, attackers can bypass protections quickly because the system leans heavily on a smaller set of core mechanisms.

### **Strengths, Limitations, and Evolving Threats**

Even though all three major operating systems approach OS security at a similar core level, each security model has their own strengths and limitations in the ever-evolving cybersecurity landscape. Linux offers the most flexibility but depends heavily on correct configuration. Windows builds strong sandboxing into modern applications, but legacy systems remain a weak point. macOS benefits from strict defaults that reduce human error, but a single kernel exploit can undermine multiple layers at once. According to Zarif, emerging threats like supply-chain attacks, kernel-mode rootkits, and memory-safe exploit techniques are pushing OS developers to explore hardware-based isolation, micro-VMs, and memory tagging. These developments reflect a trend toward even tighter confinement and reduced trust in user-space applications.

### **Real-World Attacks and Case Studies**

Real-world attacks show how sandboxing and isolation help limit damage, but also how determined attackers find ways around them. A recent example is the macOS SIP bypass CVE-2024-44243, where Microsoft Threat Intelligence found that vulnerable third-party kernel extensions allowed malicious code to perform restricted actions and effectively sidestep SIP's protections. Browser sandbox escapes also appear frequently, with attackers chaining a renderer exploit to gain initial code execution and then using a second bug to break out of seccomp, AppContainer, or the macOS Sandbox. These multistage chains highlight that sandboxes often fail when a single layer has a logic flaw or incomplete policy setting. Windows has seen similar issues with token manipulation and process-injection attacks, where malware steals or forges

access tokens, or injects into higher-integrity processes to bypass isolation boundaries. Even though Windows provides strong containment through integrity levels and AppContainer, older components and legacy processes remain common targets. Together, these cases show that isolation is effective, but real-world attacks often succeed by exploiting the layers beneath the surface or around the sandbox rather than directly confronting it.

## Conclusion

Sandboxing and process isolation sit at the core of modern OS security. While Linux, Windows, and macOS take various approaches, all three systems aim to limit what an exploited process can do and reduce the attack vector surface. By comparing their designs, strengths, and real-world failures, it becomes clear that no single model is perfect. Still, process isolation remains one of the strongest defenses available, and future OS development is moving toward even more restrictive and hardware-assisted forms of containment.

## References

- A. Niemi, “Survey of Real-World Process Sandboxing,” *2024 35th Conference of Open Innovations Association (FRUCT)*, Tampere, Finland, 2024, pp. 520-531, doi: 10.23919/FRUCT61870.2024.10516417. keywords: {Surveys;Technological innovation;Linux;Focusing;Mobile communication;Servers;Security}.
- Bustan, Moshe Siman Tov, and Nir Zadok. “The Aftermath of CVE-2025-4609: Critical Sandbox Escape Leaves 1.5M Developers Vulnerable.” *Ox*, 14 Aug. 2025 [www.ox.security/blog/the-aftermath-of-cve-2025-4609-critical-sandbox-escape-leaves-1-5m-developers-vulnerable/](http://www.ox.security/blog/the-aftermath-of-cve-2025-4609-critical-sandbox-escape-leaves-1-5m-developers-vulnerable/).

Intelligence, Microsoft Threat. “Analyzing CVE-2024-44243, a Macos System Integrity Protection Bypass through Kernel Extensions.” *Microsoft Security Blog*, 14 Jan. 2025, [www.microsoft.com/en-us/security/blog/2025/01/13/analyzing-cve-2024-44243-a-macos-system-integrity-protection-bypass-through-kernel-extensions/](https://www.microsoft.com/en-us/security/blog/2025/01/13/analyzing-cve-2024-44243-a-macos-system-integrity-protection-bypass-through-kernel-extensions/).

Ligh, Michael Hale, et al. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley, 2014.

Zarif, Bin A. “Securing Operating Systems (OS): A Comprehensive Approach to Security with Best Practices and Techniques.” *International Journal of Advanced Network, Monitoring, and Controls*, vol. 9, no. 1, 2024, pp. 100-111. *ProQuest*.