Security Vulnerabilities and Mitigations in Sandboxed and Closed Systems

Xolani Vilakazi[1]

University of Johannesburg, Johannesburg, South Africa
xolanivilakazi@uj.ac.za

Abstract. Sandboxing is a critical security mechanism used to isolate applications, preventing interference with other programs and sensitive data. This paper reviews various vulnerabilities associated with sandboxed and closed systems, focusing on virtual machines, JavaScript runtimes, and hardware platforms. It explores significant threats, including deserialization attacks, buffer overflows, and code injections in Java Virtual Machine (JVM) environments, as well as prototype pollution in JavaScript runtimes. Hardware-based vulnerabilities, such as jailbreaking and side-channel attacks, are also discussed. Finally, the paper examines countermeasures, including software-based defenses like Janus and seccomp, and hardware-based protections such as Trusted Execution Environments (TEEs). The study emphasizes that as hardware and software evolve, security mechanisms must adapt to mitigate emerging threats.

Keywords: Sandboxing · Security vulnerabilities · Closed systems · Virtual machines · Hardware-based vulnerabilities · Mitigations

1 Introduction

The increasing complexity of modern computing systems introduces significant security challenges, particularly in applications exposed to malicious threats. Sandboxing, an isolated environment where applications can run without affecting other programs or accessing sensitive data, has become a widely adopted defense mechanism in both software and hardware systems. However, as attackers continuously evolve their techniques, breaking out of sandboxes and compromising closed systems has become a prevalent issue [1]. This paper provides a comprehensive review of the vulnerabilities, exploits, and countermeasures in sandboxed and closed systems, with a particular focus on virtual machines (VMs), JavaScript runtimes, and hardware platforms.

2 Virtual Machine Vulnerabilities and Cyber Attacks

2.1 Java Virtual Machine(JVM) Vulnerabilities

The Java Virtual Machine (JVM) is a widely deployed runtime environment that enables the execution of Java applications in a sandboxed manner. However, this sandbox is not immune to vulnerabilities. One of the most common attacks on the JVM is the deserialization attack, which allows rogue objects to bypass security restrictions and execute arbitrary code [2]. Additionally, buffer overflows in the JVM can lead to system crashes, memory corruption, and even privilege escalation [1]. Further research by Chen et al. highlights the rising concern over "Java Secure Random Vulnerabilities," which allow attackers to break cryptographic protections in JVM environments [7].

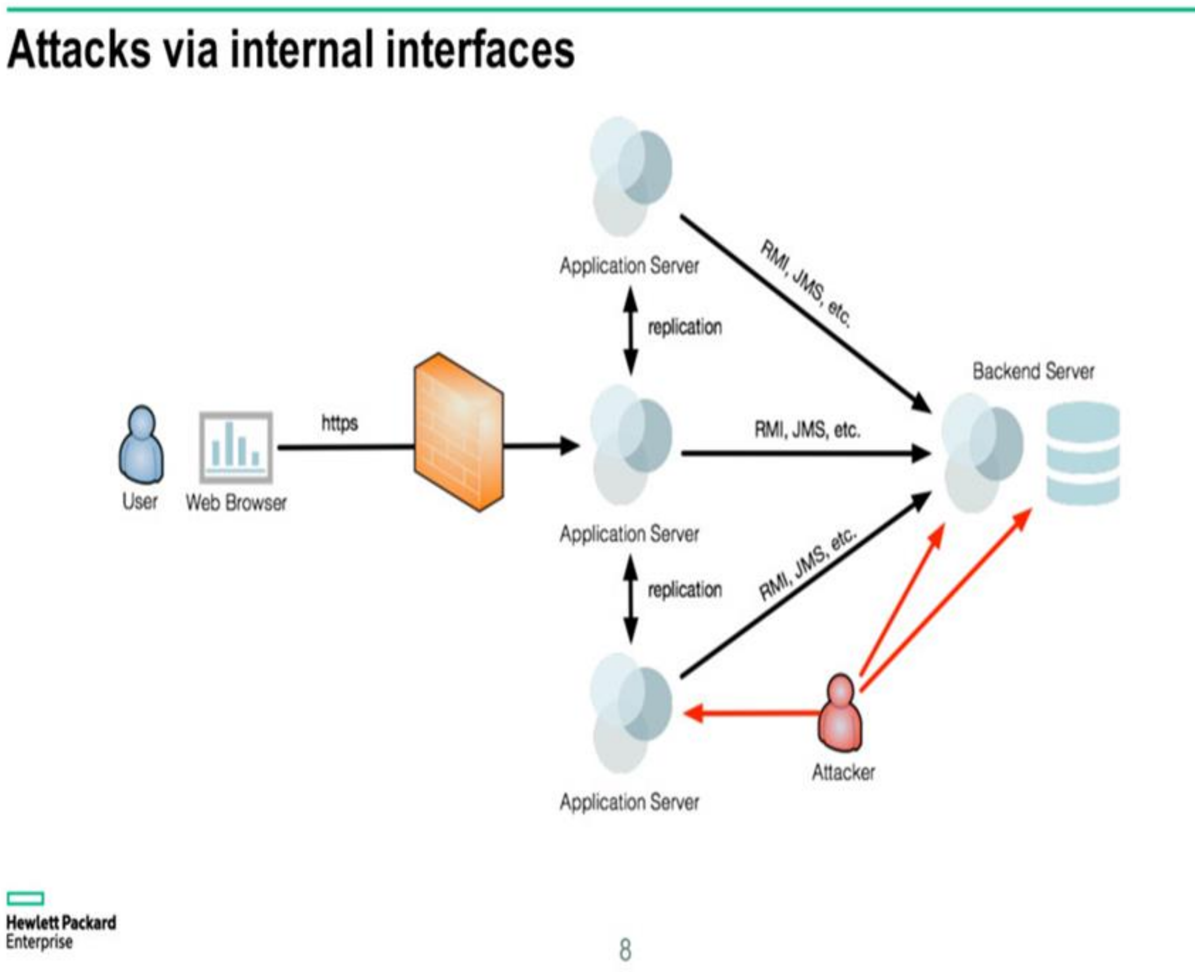
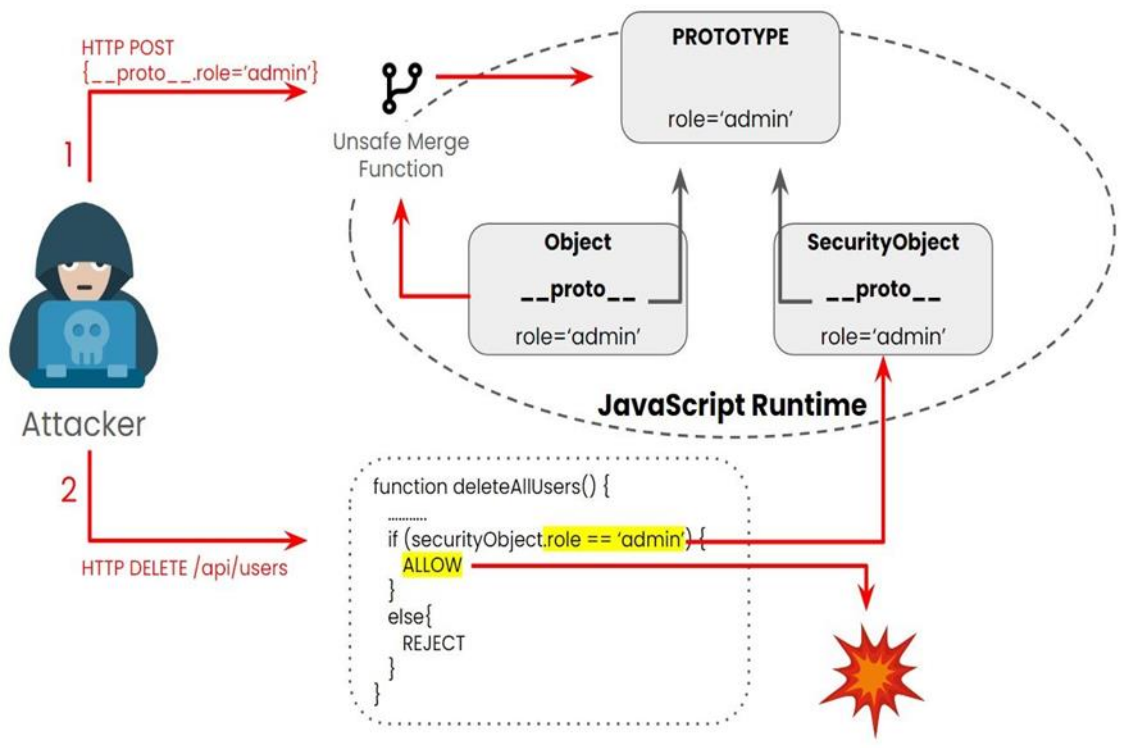


Fig.1. Diagram illustrating deserialization attack on JVM

2.2 Javascript Runtime Vulnerabilities

JavaScript runtimes, such as Node.js, are commonly used for server-side applications, but they are also prone to security vulnerabilities. Prototype pollution is a significant threat in JavaScript environments, where attackers manipulate object prototypes to introduce malicious behaviors [3]. Another critical attack vector is the sandbox escape, where untrusted code is able to break out of its isolated environment and interact with the underlying system [4]. Recent research shows that despite improvements in Node.js sandboxing techniques, new vulnerabilities are frequently discovered in third-party libraries used in runtime environments

[8].



```
function deleteAllUsers() {
  ..........
  if (securityObject.role == 'admin') {
    ALLOW
  }
  else{
    REJECT
  }
}
```

Fig.2. Prototype pollution in JavaScript runtime leading to sandbox escape

3 Hardware-Based Vulnerabilities and Cyber Threats

3.1 Jailbreaking and Unofficial firmware Attacks

Jailbreaking refers to bypassing the security restrictions of closed systems, such as smartphones and game consoles, to gain root access or install unofficial firmware. This type of attack targets vulnerabilities in the bootloader or firmware, allowing unauthorized software execution and system control [1]. Devices like iPhones have been frequent targets for jailbreakers, who exploit firmware loopholes to install unapproved apps or modify system behaviors. According to Peles et al., modern bootloader protections like Secure Boot have significantly reduced the number of successful jailbreaks, but vulnerabilities in early versions of bootloaders are still being exploited [9].

3.2 Side-Channel Attacks

Side-channel attacks are a form of hardware-based attack where attackers extract sensitive data by analyzing information such as power consumption, electromagnetic emissions, or timing variations. The Spectre and Meltdown vulnerabilities are prime examples, where attackers exploited timing differences in CPU architectures to break the isolation between processes, compromising data in sandboxed environments [5]. A recent survey by Kocher et al. identified new forms of side-channel attacks, including microarchitectural data sampling (MDS), which can potentially compromise cloud-based sandboxed environments [10].

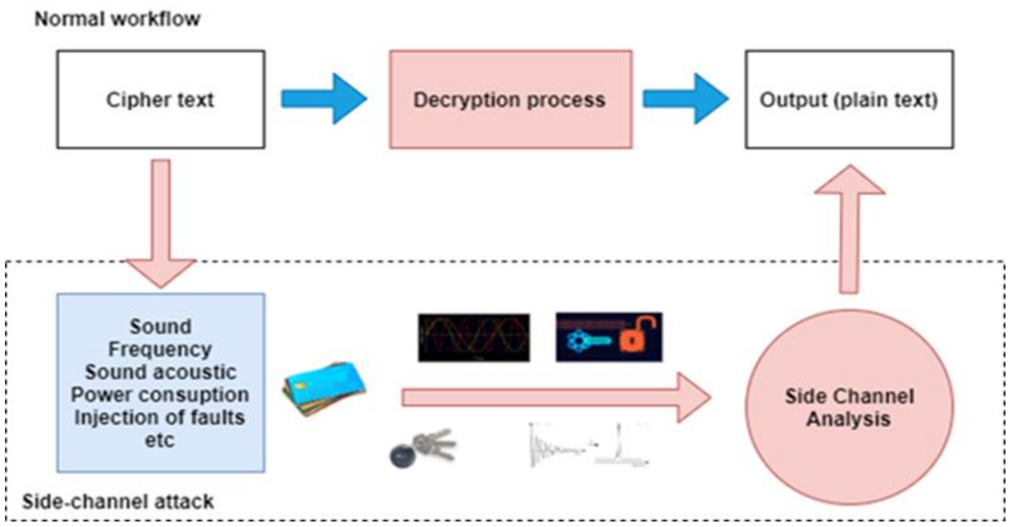


Fig.3. Side-channel attack process affecting sandboxed environments

4 Countermeasures and Mitigations

4.1 Software Based Mitigations

To defend against vulnerabilities in sandboxed environments, several softwarebased countermeasures have been developed. Janus, for instance, is a security tool that limits the interactions of sandboxed applications with system resources, thereby preventing attacks like privilege escalation [6]. Another widely adopted defense mechanism is seccomp, used primarily in Linux systems to restrict the system calls that an application can make. By reducing the attack surface, seccomp provides a strong layer of protection against various exploits, including remote code execution [2]. Research by Balzarotti et al. has demonstrated that extending seccomp with additional filters significantly improves its efficiency in high-security environments [11].

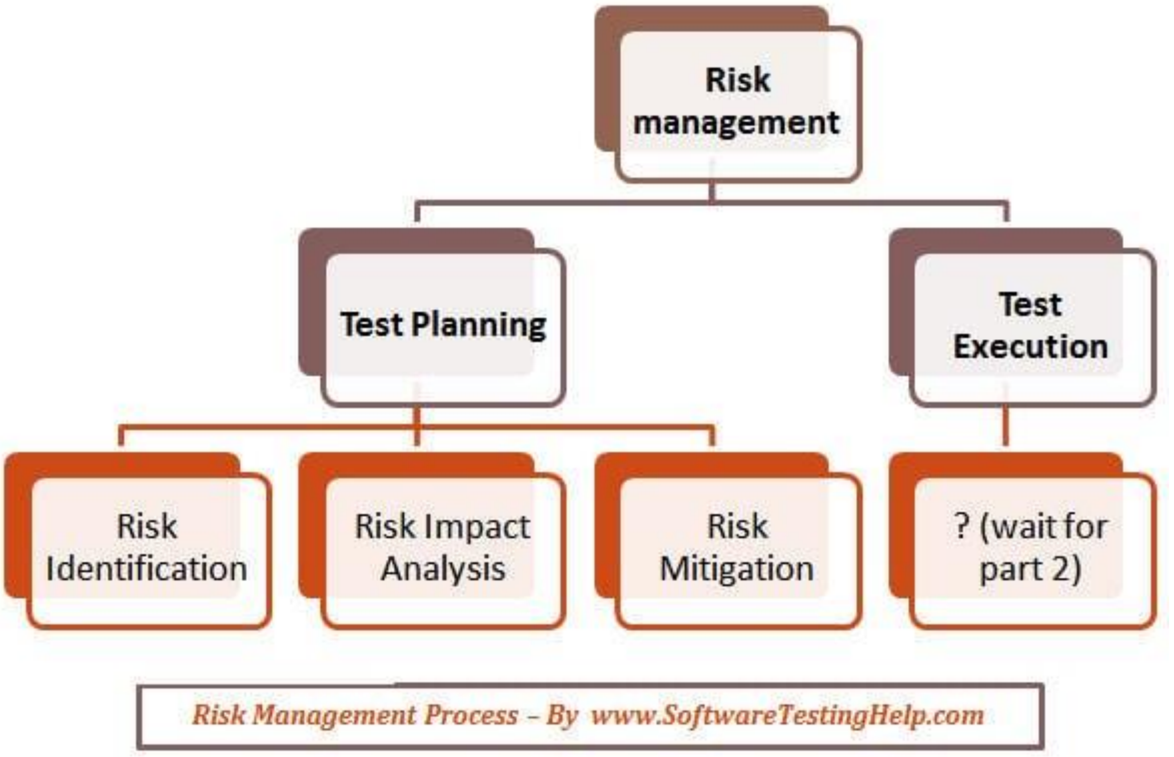


Fig.4. Risk Management process

4.2 Hardware-Based Defenses

On the hardware side, Trusted Execution Environments (TEEs) provide isolated environments for sensitive computations, ensuring that even if the main operating system is compromised, the data within the TEE remains secure [1]. Secure Boot is another essential mechanism, ensuring that only trusted and signed firmware is executed during the boot process, thereby protecting systems from unauthorized modifications. Furthermore, recent advancements in Intel's Software Guard Extensions (SGX) technology have enhanced the security of TEEs, as outlined by Costan and Devadas [12].

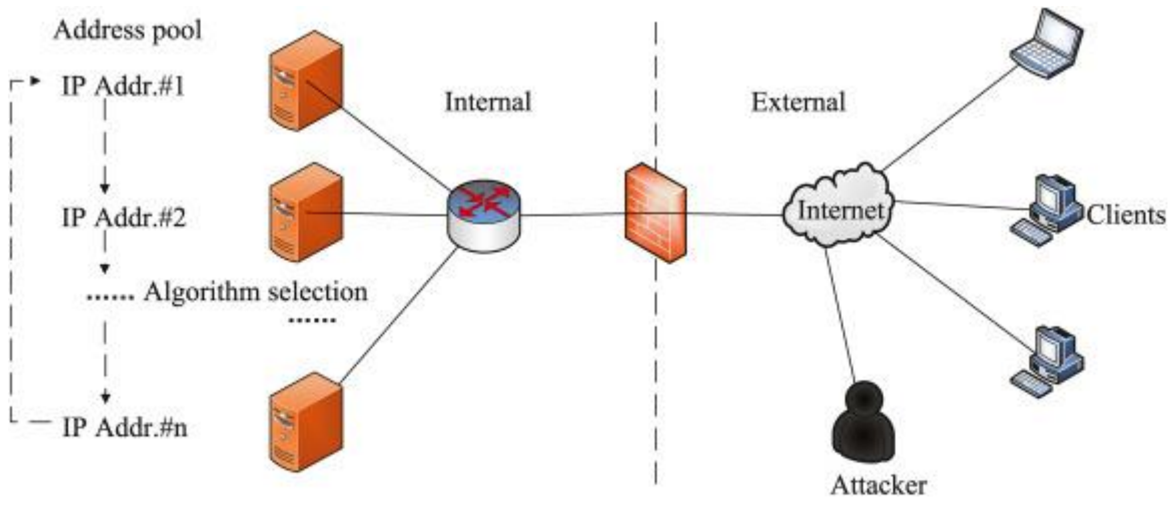


Fig.5. Address pool defence

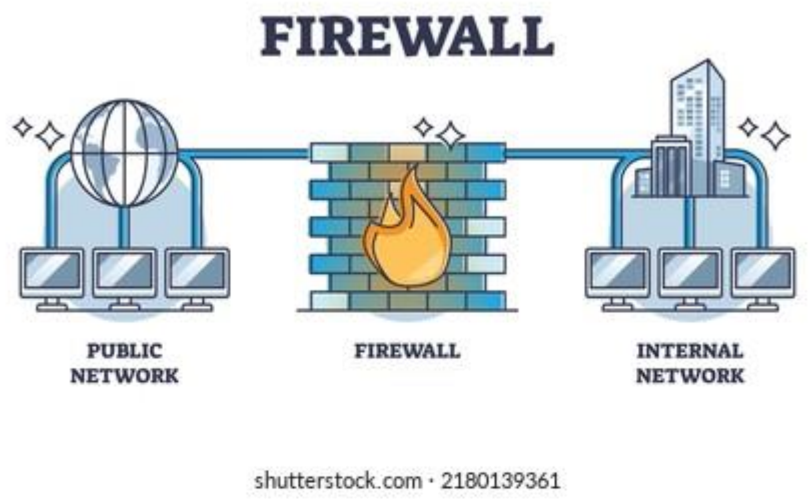


Fig.6. Hardware defense using firewall

Conclusion

Sandboxing and closed systems offer crucial layers of protection in modern computing environments. However, as hardware and software evolve, attackers find new ways to exploit vulnerabilities in these systems. This paper has reviewed key vulnerabilities, including deserialization attacks in JVM, prototype pollution in JavaScript runtimes, jailbreaking in hardware systems, and side-channel attacks. Countermeasures such as seccomp, TEEs, and Secure Boot provide essential defenses, but continuous innovation in security strategies is necessary to address emerging threats and ensure the integrity of sandboxed environments.

References

1. Prevelakis, V., Spinellis, D.: Sandboxing Applications. University of Pennsylvania(2003).
2. Peveler, M., Maicus, E., Cutler, B.: Comparing Jailed Sandboxes vs Containers.ACM SIGCSE Technical Symposium (2019).
3. Lingasubramanian, K., Kumar, R., Gunti, N., Morris, T.: Study of Hardware Trojans Based Security Vulnerabilities in Cyber Physical Systems. IEEE International Conference on Consumer Electronics (ICCE) (2018).
4. Yoshioka, K., Hosobuchi, Y., Orii, T., Matsumoto, T.: Vulnerability in Public Malware Sandbox Analysis Systems. IEEE/IPSJ International Symposium on Applications and the Internet (2010).
5. Reuben, J.S.: A Survey on Virtual Machine Security. Helsinki University of Technology (2007).
6. Janus: A Secure Environment for Untrusted Applications. USENIX Security Symposium.
7. Chen, Y., Ryan, M.: Java Secure Random Vulnerabilities: State-of-the-Art andLessons Learned. ACM Transactions on Information and System Security (2016).
8. Nash, J., Ryan, M., Mannan, M.: Security Vulnerabilities in JavaScript-Based Software: A Case Study on Node.js. ACM Transactions on Internet Technology (2019).
9. Peles, M., Chiche, A.: Securing Bootloader in Embedded Devices: A PracticalGuide. Springer, Lecture Notes in Computer Science (2018).
10. Kocher, P., Horn, J., Genkin, D., Gruss, D., et al.: Spectre and Meltdown Vulnerabilities: Implications and Next Steps. ACM Computing Surveys (2019).
11. Balzarotti, D., Kirdis, A., Bach, S.: Enhancing Seccomp for More Efficient SandboxSecurity. IEEE Security and Privacy (2020).
12. Costan, V., Devadas, S.: Intel SGX Explained. Cryptology ePrint Archive, Report2016/086 (2016).