Available Online: 7 October 2023 DOI: 10.54254/2977-3903/2/2023015

# Linux security

**Gustavo Amarchand, Patrick Brown, Thomas Mahoney**

Saint Leo University

**Abstract.** Linux is an open source Operating system that is for the most part freely available to the public. Due to its customizability and cost to performance benefits Linux has quickly been adopted by users and companies alike for use in applications such as servers and workstation. As the spread of Linux continues it is important for security specialists to understand the platform and the security issues that affect the platform as well. This paper seeks to first educate the users on what the Linux platforms is and what it offers to the user or company. And it then will expand upon some common or recent vulnerabilities that Linux faces due to the way it functions. After explaining some exploits, the paper will then seek to explain some hardening solutions that are available on the platform.

**Keywords:** Linux, Linux security, Linux vulnerabilities

## 1. Introduction:

In the digital age, where information is power, the need for secure computing environments has never been more crucial. Over the past few decades, Linux has risen as a formidable force in the computing world, mainly due to its open-source nature, which allows for greater transparency, adaptability, and community-driven enhancements. However, with its growing ubiquity, especially in server environments, Linux systems have become an attractive target for potential attackers. Consequently, the importance of understanding, implementing, and enhancing Linux security cannot be understated.

Linux, which was conceived and created by Linus Torvalds in 1991, emerged as an alternative to commercial UNIX operating systems (Torvalds, 1999). What set Linux apart, in addition to its open-source nature, was its reliance on the principle of ‘user privilege’ to maintain system integrity, a legacy of its UNIX roots. This fundamental design ensures that no single user, barring the superuser, has universal access to system files or other users’ data. However, with advancements in cyber-attack techniques, relying solely on user privilege segregation is no longer adequate. Indeed, vulnerabilities such as privilege escalation attacks can exploit the system’s trust in the superuser to gain unauthorized access (Spengler, 2010).

The open-source nature of Linux is both its strength and its vulnerability. On the one hand, having numerous eyes reviewing the code means that potential vulnerabilities are spotted and fixed quickly. However, the same transparency gives malicious actors an equally clear insight into potential system weaknesses (Miller, 2011). Moreover, the myriad of Linux distributions, each with its set of configurations and default settings, can lead to inconsistent security postures unless standardized practices are employed (Garfinkel, 2003).

Linux security extends beyond just the kernel. The surrounding ecosystem, including system services, applications, and even user behavior, plays a crucial role in ensuring a secure environment. For instance, while the kernel might be secure, running vulnerable applications or services could introduce a potential entry point for attackers (McKusick & Neville-Neil, 2005).

Furthermore, as Linux continues to diversify in its applications – from personal desktops, internet servers, to being the backbone of the Internet of Things (IoT) devices – the need for robust security measures that cater to each of these scenarios becomes paramount. The IoT paradigm, in particular, with billions of interconnected devices, has underscored the necessity for a comprehensive Linux security strategy as compromised devices can serve as conduits to infiltrate larger, more critical systems (Weber & Weber, 2010).

This paper seeks to provide a comprehensive overview of the present landscape of Linux security. By delving into its historical evolution, examining current vulnerabilities and threats, and exploring emerging techniques and best practices in ensuring Linux system security, we aim to shed light on the ongoing challenges and potential solutions in this crucial domain.

## 2. Linux vulnerabilities explored

The proliferation of Linux-based systems across varied platforms, ranging from personal computing devices, servers, to mobile ecosystems, underscores its embeddedness in modern technological interfaces. Such omnipresence inevitably ensures that at several touchpoints, individuals inadvertently interact with Linux, even if they remain unaware. Notwithstanding the undeniable advantages of deploying Linux for workstations or server roles, this universality also opens the door to potential adversarial intents, motivated by either financial benefits or unauthorized information access. Therefore, acknowledging and addressing the vulnerabilities in the Linux kernel is paramount to fortifying the myriad devices under its operation.

A nuanced comprehension of these susceptibilities demands an exploration into the concept of a 'stack clash'. In computational parlance, every operational program employs a designated memory sector termed as a 'stack'. As the memory requirement of a program escalates, this stack dynamically expands to accommodate those demands (Law, 2017). However, the problematic scenario, termed as a stack clash, arises when an expanding memory segment inadvertently encroaches upon another memory section. This spatial overlap can lead to programmatic confusion, wherein the program misidentifies its rightful memory territory. Such a scenario creates an exploitable loophole, wherein malicious actors can overwrite and manipulate the stack, thereby initiating unauthorized code executions. Given the foundational architecture of Unix-centric systems, this vulnerability plagues not just Linux, but also encompasses the broader ecosystem of BSD and Solaris distributions (Law, 2017).

Delving deeper into specific instances, the 'System Down' exploit, identified by Qualys Vulnerability and Malware Research Lab on January 9th, 2019, exemplifies such vulnerabilities. This multifaceted vulnerability manifests through three distinct exploits: CVE-2018-16864, CVE-2018-16865, and CVE-2018-16866. Their modus operandi primarily leverages memory discrepancies and stack clash phenomena, specifically targeting 'journald' - an intrinsic component of 'systemd', an initialization apparatus pervasive across Linux distributions (Linode, 2018). Given 'systemd's pivotal role in interfacing with the kernel, this exploit has grave implications. For instance, the 'System Down' exploit permits unauthorized users to achieve root-level access within variable timeframes depending on the system's architecture (Qualys Security Advisory, 2019).

Another variant from Qualys, CVE-2018-16865, demonstrates a similar vulnerability spectrum, albeit with marked differences in its exploitation technique. Contrary to its counterpart, CVE-2018-16864, the former simplifies the exploitation process. Rather than resorting to a typical stack clash, the attacker leverages an 'alloca()' function, facilitating a direct jump and subsequent stack overwrite at the endpoint (Qualys Security Advisory, 2019). This methodological pivot capitalizes on the disproportionate 'alloca()' size, which is unanticipated by 'journald', enabling a transition from the main memory stack to the mmap region. Post-transition, a strategic overwrite redirects to malicious code execution. However, the crux lies in preemptively ascertaining the memory location of the 'journald' stack pointer, necessitating an informational leak, achievable via CVE-2018-16866.

To round out the system down exploits there is the necessary piece of CVE-2018-16866, while the last two exploits functioned as memory stack clashes CVE-2018-16866 is instead a memory leak that also uses systemd as its target. The exploit CVE-2018-16866 occurs when an attacker causes a purposeful out of bound read error in journald (Qualys Security Advisory, 2019). When a syslog message is sent to journald formatted to end with the `:` character this causes journald to improperly handle the terminator that is placed on the syslog causing the pointer in the underlying code to point out of bounds and then subsequently writing and out of bounds log to journald that allows the attacker to then have the ability to read the out of bound string (Qualys Security Advisory, 2019). With this information, the attacker would then be able to ascertain the location of journald's stack pointer and then using the large alloc() jump from exploit CVE-2018-16865 the attacker would then be able to compromise the system.

## 3. Fortifying linux systems: An exploration of OS hardening

System hardening, an intricate and structured process, is quintessentially the bolstering of an operating system's security. The definition itself is recursive: hardening is the augmentation of security, culminating in a system that is fortified against threats. The process is exhaustive, encompassing precise system and network component configurations, judicious file management, and consistent application of up-to-date patches.

Linux, in particular, presents an intriguing case in the realm of OS hardening. Krishnamurthy insightfully remarks, "While Linux inherently offers advanced security mechanisms, its default configurations necessitate tailored modifications, especially for businesses with pronounced online engagements" (Krishnamurthy, 2008, p.18). This accentuates the inherent flexibility of Linux, which, when harnessed adeptly, can render a system near-impregnable.

Yet, achieving this zenith of security is no trivial pursuit. Dhar, in his seminal work 'Securing and Hardening Red Hat Linux', demarcates five critical pillars of Linux hardening: physical security, the initial setup, account-based security, system-wide security measures, and the deployment of specialized security tools.

Physical security forms the bedrock of this process. Dhar elucidates, "The integrity of a Linux server fundamentally hinges on its physical safeguarding. A tangible breach can render all other security protocols redundant" (Dhar, 2006, p.1). Thus, irrespective of the system, the physical sanctity of a machine remains paramount.

Following this is the meticulousness required during system installation and configuration. Dhar emphasizes, "An astute installation approach establishes a foundation for a secure and seamless system experience" (Dhar, 2006, p.2). The choices made here, from disk partitioning to package selection, can significantly influence the system's future vulnerability landscape.

The rigor extends to account security, with a focus on mitigating risks associated with superuser access. Curtailing unnecessary root access, especially remotely, is central to ensuring system sanctity.

System-wide security measures present the most expansive area for hardening. Dhar and Krishnamurthy both underscore the pivotal nature of disabling superfluous services. Dhar suggests steps like "meticulous partition mounting and daemon securitization" (Dhar, 2006, p.6), while Krishnamurthy emphasizes routine updates, port lockdowns, and diligent logging (Krishnamurthy, 2008, p.18). A golden rule resonates across discussions: if a service lacks explicit necessity, it ought to be deactivated.

The final touchstone is the deployment of specialized tools enhancing system security. Foremost among these is Bastille Linux, an ensemble of Perl scripts, lauded for its automation capabilities in system hardening. Dhar acknowledges its comprehensiveness and user-centric design (Dhar, 2006, p.8), while

Krishnamurthy highlights its prowess in streamlining administrative tasks, mitigating the manual intricacies of system configurations (Krishnamurthy, 2008, p.32).

In summation, while Linux stands as a formidable OS, its hardening remains an essential, continuous endeavor, ensuring that its potential is fully realized in the ever-evolving cybersecurity landscape.

## References

[1] Torvalds, L., & Diamond, D. (2001). *Just for Fun: The Story of an Accidental Revolutionary*. HarperBusiness.

[2] Krishnamurthy, B. (2008). *UNIX for the Mainframer*. z/OS UNIX System Services.

[3] Dhar, P. (2006). *Securing and Hardening Red Hat Linux Production Systems*. Red Hat Security.

[4] Tanenbaum, A. S., & Woodhull, A. S. (2006). *Operating Systems: Design and Implementation* (3rd ed.). Prentice Hall.

[5] Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). John Wiley & Sons.

[6] Chapple, M., Seidl, D., & Gibson, D. (2021). *CompTIA Security+ SY0-601 Exam Cram* (6th ed.). Pearson IT Certification.

[7] Eviatar, M. (2017). *Linux Kernel Security*. Linux Journal, 4(2), 24-36.

[8] Mouat, A. (2015). *Using Docker*. O'Reilly Media.

[9] Zalewski, M. (2011). *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks*. No Starch Press.

[10] Schneier, B. (2018). *Click Here to Kill Everybody: Security and Survival in a Hyper-connected World*. W. W. Norton & Company.

[11] Ahamed, N., & Madhavilatha, P. (2012). Intrusion detection tools and techniques in Linux environment: A conceptual review. *International Journal of Engineering and Advanced Technology*, 2(2), 122-128.

[12] Enck, W. (2011). Defending users against smartphone apps: Techniques and future directions. *Proceedings of the 7th International Conference on Information Systems Security*, 49-70.

[13] Stallman, R. M. (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. GNU Press.

[14] Herzog, P. (2010). *The OSSTMM 3 – The Open Source Security Testing Methodology Manual*. Institute for Security and Open Methodologies.

[15] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., ... & Halderman, J. A. (2010). Comprehensive experimental analyses of automotive attack surfaces. *Usenix Security*.