Elevating SysCalls to Privileged Actions to Prevent Unauthorized Software Attacks

Parth Thakre[1], Atharva Bhajan[2]
[1]*IT, Government Polytechnic Nagpur, India*
E-mail: parthakre114@gmail.com
[2]*IT, Government Polytechnic Nagpur, India*
E-mail: atharvabhajan0574@gmail.com

Abstract — ***NexPro is an advanced system call interception framework designed to address modern security threats by offering enhanced control and protection mechanisms. Building upon the foundational work of nexpoline, NexPro introduces fine-grained policies, behavioural learning, process lineage verification, enhanced cryptographic signatures, modular architecture, and user-space control. These improvements enable more secure and efficient system call management, making NexPro a viable solution for preventing unauthorized system access and detecting anomalies in real-time. By adopting these new features, NexPro ensures greater flexibility, scalability, and adaptability to dynamic computing environments, including cloud and container-based systems.***

Keywords*:*

System Call interception Mechanism, Security Policies for Syscalls, Endpoint Protection, Syscall hooking, Kernel Security

I. INTRODUCTION

System call (syscall) interception has become crucial in modern cybersecurity due to the rise of increasingly sophisticated cyberattacks, including ransomware and advanced persistent threats (APTs). Attackers often exploit vulnerabilities at the system level by injecting malicious code or unauthorized software that executes dangerous syscalls, granting access to critical resources such as files, network operations, and system memory. Ransomware, for example, leverages system calls to encrypt files and disable system recovery mechanisms, holding valuable data hostage.

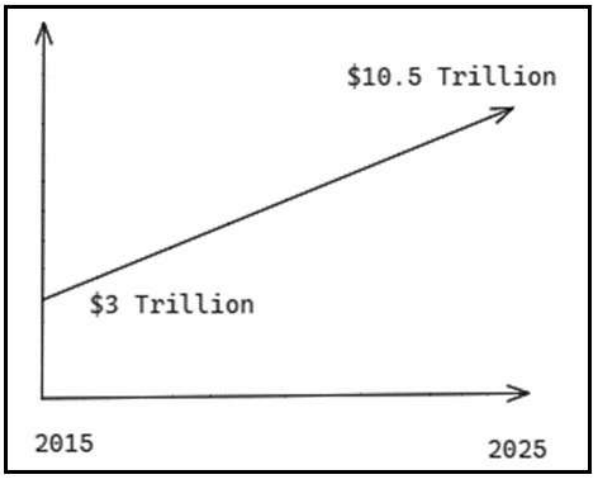

Fig. 1: Growth of Damages Costs due to cybercrime

According to recent reports, ransomware attacks surged by over 150% globally in 2023, with damages exceeding $20 billion.[5] Moreover, APTs persistently target enterprise and governmental systems, executing syscalls to escalate privileges, steal sensitive data, and maintain long-term access. Traditional security measures often fail to detect or block these threats at the syscall level, leading to widespread system compromise. NexPro addresses these challenges by intercepting and controlling syscalls with fine-grained policies, behavioral learning, and cryptographic verification, effectively preventing unauthorized system access and mitigating ransomware and APT attacks before they cause damage. By implementing advanced syscall interception, NexPro serves as a crucial defense against modern cyber threats, providing real-time protection and reducing the success rate of malicious activities.

Early frameworks primarily focused on monitoring system calls to detect unauthorized access or malware activity, using static rule-based approaches to allow or block specific system calls. However, as attacks grew more sophisticated, these frameworks struggled to adapt, leading to the development of more dynamic and context-aware solutions.

Frameworks like ptrace, seccomp, and SELinux introduced syscall interception mechanisms, adding the ability to enforce access control policies. Yet, despite their usefulness, they often lacked adaptability, especially in environments where dynamic workloads and rapidly evolving threats demand more robust solutions. Over time, the need for behavioural analysis, machine learning, and real-time anomaly detection emerged as critical factors in enhancing these frameworks' effectiveness.

2. Background:

System call interception frameworks have evolved significantly in response to growing security challenges and the need for fine-grained control over processes.

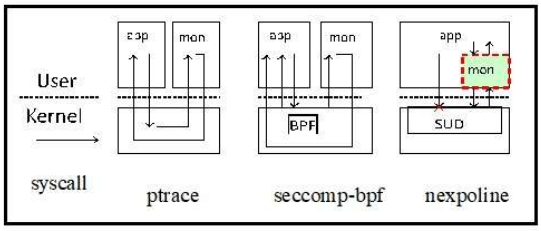

Fig. 1: Secure Syscall Interception Mechanism[4]

Nexpoline[4]

Nexpoline, as a system call interception mechanism, offers a structured approach to controlling system calls, introducing elements of policy-based management to enhance system security. It was a significant step forward by focusing on the interception of system calls based on predefined policies, ensuring that unauthorized entities were blocked from making critical system calls. This approach helped reduce the attack surface by preventing malware

and unauthorized applications from gaining access to privileged operations.

Nexpoline has several limitations:

1. Static Policy Management
2. No Behavioural Learning
3. Limited Process Verification
4. Cryptographic Weaknesses
5. Monolithic Architecture

NEXPRO

NexPro is an advanced syscall interception and monitoring framework designed to provide enhanced security by making system calls a **privilege**, not a right. Unlike traditional syscall interception mechanisms like **seccomp** or **ptrace**, which rely on static filtering or user-space debugging, NexPro introduces a **fine-grained, policy-driven** approach that enforces strict control over syscalls based on behavioral learning, process lineage, and cryptographic verification.

By using techniques like **Memory Protection Keys (MPK)** and **Supervisor User Data (SUD)**, NexPro ensures syscall validation occurs with minimal performance overhead while offering more robust protection against **unauthorized** and **anomalous** system calls. This makes it superior to existing solutions, providing both **security** and **efficiency** at a higher level than conventional syscall interception methods.

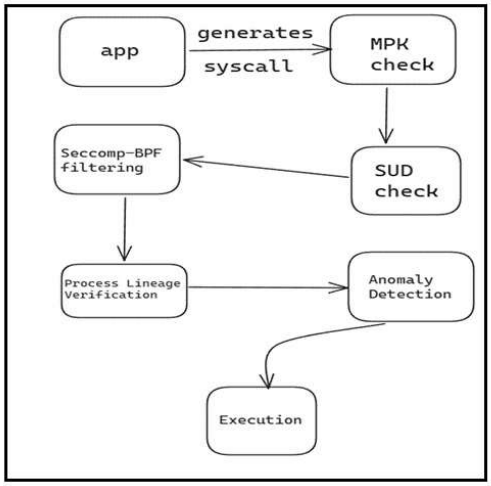

Fig 1. NexPro flow of control of syscall

A. Flow of nexpro

1. Application Generates a Syscall

2. Memory Protection Keys (MPK) Check

PKRU	pkey 0	1	2
$trusted_pkru	rw	rw	rw
$untrusted_pkru	ro	-	rw

TABLE I: MPK Permission Assignment

3. Syscall User Dispatch (SUD) Check

5. Seccomp-BPF Filtering (Optional)

6. Process Lineage Verification

7. Behavioral Learning and Fine-Grained Policies (Future Enhancements)

8. Syscall Execution & Response

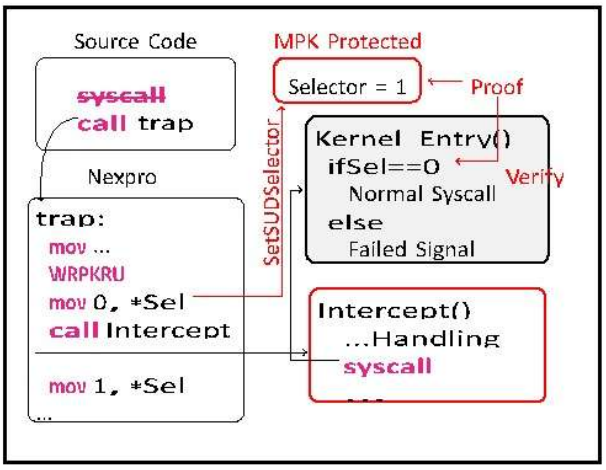

Fig. 2: **Nexpro Syscall interception**

NexPro incorporates a syscall logging mechanism designed for security auditing and policy creation. Every syscall that passes through NexPro is logged with details such as the application name, process ID, user context, syscall number, arguments, and target resources. Additionally, results from checks (MPK, SUD, Trampoline, seccomp filters) and the decision outcome—whether the syscall was allowed or blocked—are recorded. These logs are securely stored, protected by cryptographic signatures to prevent tampering, and periodically reviewed to identify any anomalies or potential security threats.

The logged data also supports behavioral analysis, enabling NexPro to identify legitimate syscall patterns over time. This data is leveraged for dynamic policy creation. NexPro can automatically suggest policies based on observed patterns, reducing manual administrative effort. For example, if an application consistently generates trusted syscalls, NexPro can propose adding them to an allowlist.

Administrators can create and modify policies manually, defining which syscalls are permitted or blocked based on criteria like user groups or execution contexts. NexPro's policies are structured into allowlists for trusted syscalls and denylists for blocking risky or unknown syscalls, ensuring flexible and secure management of syscall behavior. This dynamic policy framework, informed by real-time logs and behavioral learning, ensures NexPro stays adaptable to evolving security needs while maintaining robust protections**.**

Policy Creation and Enforcement in NexPro

NexPro provides a robust mechanism for managing syscall access through customizable policies tailored for each application. Applications can have their own specific syscall policies, structured into allowlists and denylists. The **allowlist** defines which syscalls are trusted and permitted based on predefined patterns, while the **denylist** blocks syscalls from applications, user contexts, or memory regions that are considered risky. NexPro's automated policy creation is powered by its behavioral learning system, which monitors syscall logs to suggest policies. For instance, if an application consistently generates trusted syscalls, NexPro can propose an allowlist entry automatically. Alternatively, administrators can manually create or modify policies based on real-time security needs, blocking specific syscall types that are known for exploitation by malware.

When a syscall is initiated, NexPro consults its policy database to determine whether the syscall

should be allowed or blocked. If a syscall matches an allowlist entry, it is immediately permitted, while any syscall on the denylist or violating specified conditions is promptly blocked. This dynamic yet controlled policy enforcement ensures that only legitimate syscalls are processed, maintaining system security without manual intervention in most cases.

Performance and Impact of NexPro

Despite its stringent security measures, NexPro is designed to operate without significantly affecting system performance. It uses highly optimized syscall filtering techniques, such as **Memory Protection Keys (MPK)**, **Supervisor User Data (SUD)**, and **seccomp filters**. These mechanisms function efficiently at the kernel level, reducing the need for frequent context switching and ensuring minimal latency during syscall interception and filtering. By leveraging memory protection mechanisms like MPK and SUD, NexPro minimizes the computational burden, ensuring syscalls are processed quickly without sacrificing security.

One of the key performance optimizations in NexPro is its reliance on **pre-compiled policies**, which are stored in fast-access structures like hash tables or trees. This setup allows NexPro to make decisions in real-time without requiring heavy computations. As soon as a syscall occurs, NexPro rapidly checks the pre-compiled policies to determine whether the syscall is allowed or blocked, ensuring instant access and minimal delay.

NexPro's **behavioral learning system** operates in the background during periods of low CPU usage, processing syscall logs and suggesting policies without affecting active syscall performance. This ensures that learning-based tasks, such as pattern analysis and policy recommendations, occur without interrupting or slowing down critical system operations. Furthermore, NexPro's **modular architecture** allows for independent execution of different components, such as syscall filtering, logging, and policy enforcement, reducing the chances of performance bottlenecks. For instance, while policy enforcement occurs in real-time, it does not interfere with the syscall logging process, ensuring smooth system operations.

The use of a **trampoline mechanism** for syscall interception further enhances performance by quickly passing legitimate syscalls to the system kernel once they are verified. This low-latency approach ensures that normal operations are not hindered by NexPro's security checks. To prevent logging overhead from affecting syscall speed, NexPro employs an **asynchronous logging system**. Logs are buffered and written in batches by a separate thread, ensuring that syscall handling is not delayed by the logging process.

Overall, NexPro's performance optimizations ensure that it operates with minimal **CPU and memory overhead**. The lightweight security checks and efficient memory management keep resource utilization low, even when enforcing complex policies or logging a high volume of syscall data. NexPro's design also allows it to

scale effectively in **multi-core systems**, distributing workloads efficiently across cores and maintaining consistent performance, even under heavy syscall traffic. This scalability makes NexPro ideal for high-performance environments where both security and speed are critical.

IX. CONCLUSION

The idea of nexpro, a mechanism that utilizes MPK, Seccomp and SUD to make syscall a privilege.

It allows for exhaustive, safe, and low-overhead system call interception without requiring to modify the kernel. It can be used with other binary rewriting tools to enhance their security. Nexpro provides secure and reliable support for system call policies in tools such as sandboxes, in-process monitors, and operating system emulation.

REFERENCES

[1] Linux, Syscall user dispatch — the linux kernel documentation, https://docs.kernel.org/admin-guide/syscall user-dispatch.html, (Accessed on 12/21/2023), 2023.

[2] byValentin Rothberg, Improving linux container security with seccomp — enable sysadmin, https://www. redhat . com / sysadmin / container-security- seccomp, (Accessed on 12/21/2023), 2020.

[3] W. Drewry, Dynamic seccomp policies (using bpf fil ters) [lwn.net], https : / / lwn . net / Articles / 475019/, (Accessed on 12/21/2023), 2012.

[4] Making 'syscall' a Privilege not a Right by Fangfei Yang, Anjo Vahldiek-Oberwagner, Chia-Che Tsai, Kelly Kaoudis, Nathan Dautenhahn 2406.07429 (arxiv.org)

[5] 130 Cybersecurity Statistics: 2024 Trends and Data (terranovasecurity.com)

[6] T. Kim and N. Zeldovich, "Practical and effective sand boxing for non-root users," in 2013 USENIX Annual Technical Conference (USENIX ATC 13), San Jose, CA: USENIX Association, Jun. 2013, pp. 139–144. [Online].Available:https://www.usenix.org/conference/atc13/technical-sessions/presentation/kim.

[7] B. Brimhall, J. Garrard, C. De La Garza, and J. Coffman, "A comparative analysis of linux mandatory access control policy enforcement mechanisms," in Proceedings of the 16th European Workshop on System Security, ser. EUROSEC '23, Rome, Italy: Association for Computing Machinery, 2023, pp. 1–7. [Online]. Available: https://doi.org/10.1145/3578357.3589454.

[8] B. McCarty, SELinux: NSA's Open Source Security Enhanced Linux. O'Reilly Media, Inc., 2004.

[9] Firejail, Firejail — security sandbox, https://firejail. wordpress.com/, (Accessed on 01/10/2024), Oct. 2023.

[10] A. Vahldiek-Oberwagner, E. Elnikety, N. O. Duarte, M. Sammler, P. Druschel, and D. Garg, "ERIM: Secure, efficient in-process isolation with protection keys (MPK)," in 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1221–1238. [Online]. Available: https : / / www. usenix.org/conference/usenixsecurity19/presentation/ vahldiek-oberwagner.

[11] K. Cook, Pku usage improvements for threads- kees cook, https : / / lore . kernel . org / lkml / 202208221331 . 71C50A6F @ keescook/, (Accessed on 12/30/2023), Aug. 2022.