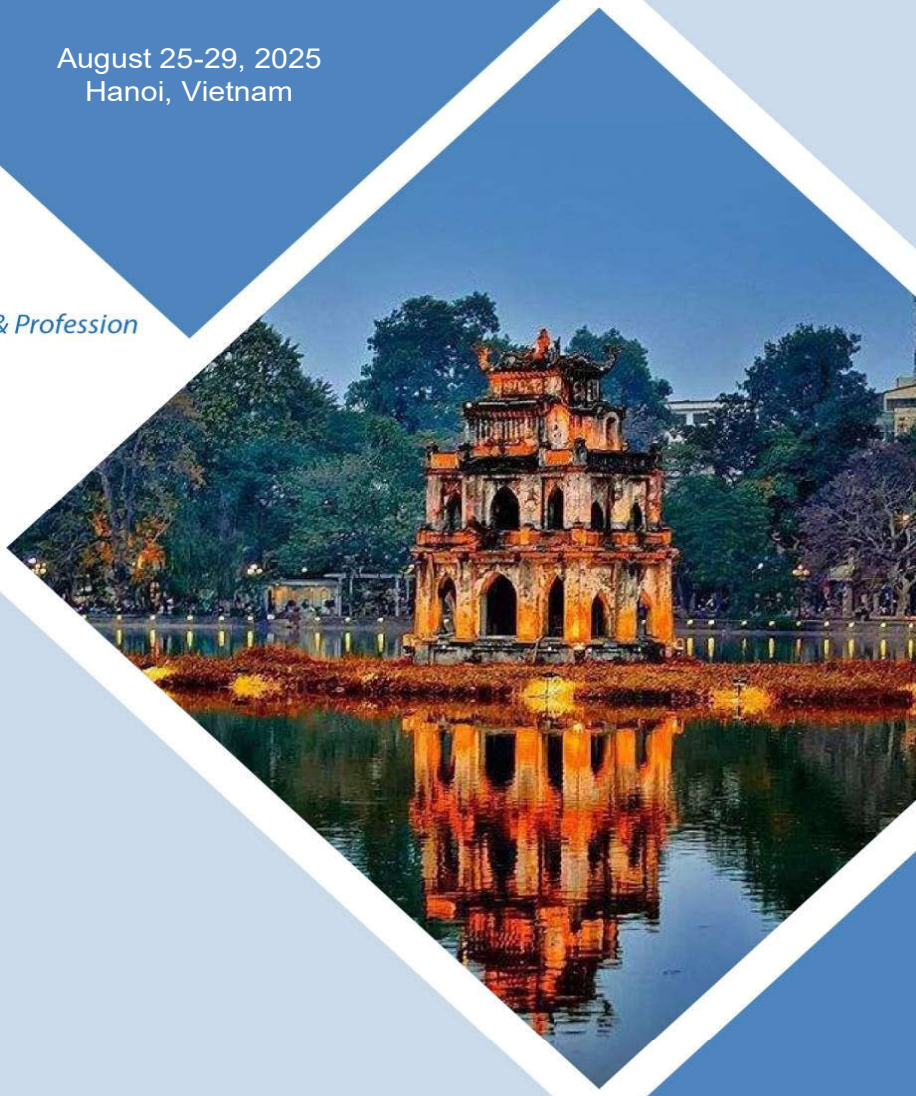
August 25-29, 2025
Hanoi, Vietnam



Association for Computing Machinery

Advancing Computing as a Science & Profession

ACM ASIACCS 2025

Proceedings of the 20th ACM ASIA
Conference on Computer and Communications Security

Sponsored by:
ACM SIGSAC

General Chairs:
Huynh Quyet Thang, HUST, Vietnam
Phan Duong Hieu, Institut Polytechnique de Paris, France

Program Chairs:
Michail Maniatakos, NYU Abu Dhabi, United Arab Emirates
Yinqian Zhang, SUSTech, China

The Association for Computing Machinery
1601 Broadway, 10th Floor
New York, New York 10019, USA

ACM ISBN: 979-8-4007-1410-8/25/08

Message from General Co-Chairs

It is our great pleasure to welcome you to AsiaCCS 2025, the 20th ACM Asia Conference on Computer and Communications Security (AsiaCCS 2025). This year's event will be held at the Meliá Hanoi in Hanoi, Vietnam, from 25 to 29 August 2025.

AsiaCCS 2025 marks a special milestone - its 20th anniversary - and we are proud of how the conference has grown into the premier venue for cybersecurity research in the Asia-Pacific region. To commemorate two decades of development, AsiaCCS 2025 is introducing a Test-of-Time Award and will host a special 20th-birthday celebration. This year also marks the first time Vietnam will host AsiaCCS, and we hope AsiaCCS 2025 will further strengthen Vietnam's cybersecurity community and foster new collaborations across the region.

The main conference features three keynote talks by world-renowned researchers - Prof. Moti Yung (Google / Columbia University), Prof. Wenyuan Xu (Zhejiang University), and Prof. Yier Jin (Huawei) - alongside our full-paper presentations and a vibrant poster session.

As in past years, we offer a rich program of workshops and keynotes. On the first day, eight workshops will explore cutting-edge topics:

- 12th ACM ASIA Public-Key Cryptography Workshop (APKC)
- 7th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI)
- 11th ACM Cyber-Physical System Security Workshop (CPSS)
- Workshop on Privacy in Large Language Models (LLM) and Natural Language Processing
- 2nd Workshop on Security-Centric Strategies for Combating Information Disorder (SCID)
- 3rd ACM Workshop on Secure and Trustworthy Deep Learning Systems (SecTL)
- 4th Workshop on Security Implications of Deepfakes and Cheapfakes
- International Workshop on Secure and Efficient Federated Learning

We would like to express our deep gratitude to our organizing committee, including

- **Program Chairs:** Michail Maniatakos and Yinqian Zhang
- **Local Organizing Chairs:** Huynh Thi Thanh Binh and Tran Quang Duc
- **Workshop Chair:** Khoa Nguyen
- **Poster Chairs:** Pham Van Thuan and Yue Duan
- **Sponsorship Chair:** Le Xuan Thanh
- **Web Chair:** Tran Hai Anh
- **Publicity Chairs:** Vo Quang Viet and Cao Minh Phuong
- **Publication Chairs:** Dinh Tien Tuan Anh and Tong Van Van

In particular, we owe a special debt of gratitude to Tran Quang Duc as a lead coordinator, his tireless efforts behind the scenes have kept every piece of this conference running smoothly - without him, AsiaCCS 2025 simply would not have happened.

We would also like to acknowledge our generous supporters: Calif Global Inc. as Supporting Partner, sponsoring a large portion of local participants, and the Singapore Institute of Technology (SIT) as a sponsor.

We hope you enjoy the program, the city of Hanoi, and that AsiaCCS 2025 provides an inspiring forum for collaboration for all participants.

General Chairs
Huynh Quyet Thang and Phan Duong Hieu

Message from Program Co-Chairs

It has been an exciting journey for us to serve as PC Co-Chairs of AsiaCCS 2025. Selecting the papers to be presented is the ultimate responsibility and the most important part of a conference organization. We would like to express our gratitude to all authors, PC members, the organizing committee, as well as the steering committee for their great support in the past months. This team effort led to the formation of an exciting program!

Submissions were received from researchers worldwide, from both academia and industry. These submissions present novel contributions related to real-world aspects of security, privacy, and cryptography, with real world impact and applications. A total of 558 submissions were received: 289 in the first Cycle and 269 in the second Cycle. The reviewing process for each Cycle consisted of two rounds.

- In the first Cycle, 12 papers were desk rejected for violation of submission policies, 133 papers were early rejected in the first round, and 80 were rejected in the second round. 34 papers were invited for a major revision, 19 were accepted with shepherding, and 11 were accepted straight away. Out of 34 major revision papers, 31 were accepted, for a total of 61 papers accepted in Cycle 1.
- In the second Cycle, 24 papers were desk rejected for violation of submission policies, 109 papers were early rejected in the first round, and 83 were rejected in the second round. There was no Major Revision process in Cycle 2. 39 papers were accepted with shepherding, and 14 were accepted straight away, for a total of 53 papers accepted in Cycle 2.

In total 114/558 submissions were accepted, for an acceptance rate of 20.4%.

The AsiaCCS 2025 technical program consists of 114 talks corresponding to the accepted papers, a poster session, as well as three keynote talks by internationally prominent and active researchers. Also, for the first time in the history of AsiaCCS a Test-of-Time nomination process was initiated with the awards to be given during the conference.

We offer our deepest gratitude to:

- Authors of every submission to AsiaCCS 2025. We thank them for considering AsiaCCS as their preferred venue for demonstrating their research outputs and for their trust in us and the PC to thoroughly and faithfully conduct the reviewing process.
- Local Chairs Tran Quang Duc and Huynh Thi Thanh Binh for being very responsive, answering every query promptly, supporting us and keeping us on track with timely follow ups!

- Publication Chairs Dinh Tien Tuan Anh and Tong Van Van for taking care of the proceedings and handling numerous requests from the authors.
- The Test-Of-Time award committee who joined us, namely Debin Gao, Alvaro Cardenas, Surya Nepal and Gene Tsudik, for their input, nominations, and participation in the award selection process.
- General Chairs Phan Duong Hieu and Huynh Quyet Thang, who demonstrated great leadership and dealt with all the logistical and organizational matters.
- The AsiaCCS Steering Committee for their confidence in selecting us as PC Co-Chairs and their support throughout the process leading to the program and successful conference.
- And last but certainly not least, all PC members who are the main driving force of success and whose hard work yielded the excellent program. Special thanks to the recipients of the “Outstanding PC Members”, William Blair, Kristen Moore, Andreas Kogler, George Stergiopoulos, Nir Drucker, Ding Wang, Sofia Celi, Xianghang Mi, Jianliang Wu, for going the extra mile to provide high-quality reviews and volunteering to shepherd and review more papers.

In closing, we look forward to the exciting conference days in beautiful Vietnam in August and hope that all attendees enjoy the conference.

PC Co-Chairs
Mihalis Maniatakos and Yinqian Zhang

Table of Contents

Homomorphic Encryption and Zero knowledge

Multi-party Computation

Applied Crypto

Post-Quantum

LLM for Security

ML Security

ML Applications to Security

Privacy 1

Privacy 2

Blockchain 1

Blockchain 2

Blockchain 3

IoT Security

CPS Security

Hardware Security

Fault Injection and Side Channels

Web Security

Network Security

Usable Security and Privacy

Software and OS Security

Binary Security

Poster

ASIA CCS'25 Organisation

General Chairs	Huynh Quyet Thang (Hanoi University of Science and Technology, Vietnam) Phan Duong Hieu (Institut Polytechnique de Paris, France)
Program Chairs	Michail Maniatakos (New York University Abu Dhabi, UAE) Yinqian Zhang (Southern University of Science and Technology, China)
Local Organising Chairs	Huynh Thi Thanh Binh (Hanoi University of Science and Technology, Vietnam) Tran Quang Duc (Hanoi University of Science and Technology, Vietnam)
Workshop Chair	Khoa Nguyen (University of Wollongong, Australia)
Poster Chairs	Pham Van Thuan (University of Melbourne, Australia) Yue Duan (Singapore Management University, Singapore)
Sponsorship Chair	Le Xuan Thanh (Hanoi University of Science and Technology, Vietnam)
Web Chair	Tran Hai Anh (Hanoi University of Science and Technology, Vietnam)
Publicity Chairs	Vo Quang Viet (Swinburne University of Technology, Australia) Cao Minh Phuong (University of Illinois at Urbana-Champaign, USA)
Publication Chairs	Dinh Tien Tuan Anh (Deakin University, Australia) Tong Van Van (Hanoi University of Science and Technology, Vietnam)
Steering Committee	Gail-Joon Ahn (Arizona State University, USA) Robert Deng (Singapore Management University, Singapore) Adrian Perrig (ETH Zürich, Switzerland) Kui Ren (Zhejiang University, China) Shiuhpyng Shieh (National Chiao Tung University, Taiwan) Xiaofeng Wang (Indiana University Bloomington, USA) Jianying Zhou (Singapore University of Technology and Design, Singapore)
Program Committee	Abdelrahaman Aly, Technology Innovation Institute, UAE Abhishek Bichhawat, IIT Gandhinagar Adithya Vadapalli, IIT Kanpur Alejandro Cuevas, Carnegie Mellon University Alessandro Brighente, University of Padova Alptekin Küpçü, Koç University Andreas Kogler, Graz University of Technology Anjia Yang, Jinan University Annabelle McIver, Macquarie University Awais Rashid, University of Bristol, UK Bo Chen, Michigan Technological University Carlos Rubio-Medrano, Texas A&M University- Corpus Christi Changhai Ou, Wuhan University Charalambos Konstantinou, KAUST Chenglu Jin, CWI Amsterdam

	Chia-Mu Yu, National Yang Ming Chiao Tung University Coby Wang, Visa Research Daisuke Mashima, Singapore University of Technology and Design Debin Gao, Singapore Management University Ding Wang, Nankai University Duc Le, Visa Research (Cycle 1 only) Fabio De Gaspari, Sapienza University of Rome Fan Zhang, Zhejiang University Fei Zuo, University of Central Oklahoma Gabriele Oligeri, Hamad bin Khalifa University George Stergiopoulos, University of the Aegean Ghada Almashaqbeh, University of Connecticut Guoxing Chen, Shanghai Jiao Tong University Gustavo Banegas, INRIA Haifeng Yu, National University of Singapore Hao Zhou, The Hong Kong Polytechnic University Haoyu Ma, Zhejiang Lab Haoyu Wang, Huazhong University of Science and Technology Hervé Debar, Télécom SudParis Huawei Huang, Sun Yat-sen University Hung Nguyen, The University of Adelaide Hyoungshick Kim, Sungkyunkwan University Ioannis Demertzis, UCSC Jianliang Wu, Simon Fraser University Jianyu Niu, Southern University of Science and Technology Jin-Hee Cho, Virginia Tech Joaquin GARCIA ALFARO, Institut Polytechnique de Paris Juanru Li, Shanghai Jiao Tong University Jun Sakuma, Tokyo Institute of Technology Kan Yang, University of Memphis Kasper Rasmussen, University of Oxford Katerina Mitrokotsa, University of St. Gallen, Switzerland Katsunari Yoshioka, Yokohama National University Kehuan Zhang, The Chinese University of Hong Kong Kevin Leach, Vanderbilt University Kristen Moore, CSIRO's Data61 Kun Sun, George Mason University Kwok-Yan Lam, Nanyang Technological University, Singapore Lei Xu, Nanjing University of Science and Technology Lei Yu, Rensselaer Polytechnic Institute Leo Zhang, Griffith University Lilas Alrahis, Khalifa University Man Ho Au, The Hong Kong Polytechnic University (Cycle 1 only) Manaar Alam, New York University Abu Dhabi Marcus Botacin, Texas A&M University Mengyuan Li, USC

	Michele Carminati, Politecnico di Milano Min Chen, CISPA Helmholtz Center for Information Security (Cycle 1 only) Minghong Fang, University of Louisville Minghui Xu, Shandong University Mohammad Ashiqur Rahman, Florida International University Nalin Arachchilage, RMIT University, Australia Natalia Stakhanova, University of Saskatchewan, Canada Neetesh Saxena, Cardiff University Nektarios Tsoutsos, University of Delaware Ning Wang, University of South Florida Ning Zhang, Washington University in St. Louis Ningyu He, The Hong Kong Polytechnic University Nir Drucker, IBM Research – Israel Olga Gadyatskaya, Leiden University Peng Gao, Virginia Tech Pengfei Hu, Shandong University Prabhu Karthikeyan Rajasekaran, Google Prashant Hari Narayan Rajput, InterSystems Qingyang Wang, Louisiana State University Rakesh Bobba, Oregon State University Roberto Guanciale, KTH Roopa Vishwanathan, New Mexico State University Rui Ning, Old Dominion University (Cycle 1 only) Runchao Han, Babylon Labs Salil Kanhere, University of New South Wales Sandra Rueda, Universidad de los Andes, Colombia Sanghyun Hong, Oregon State University (Cycle 1 only) Satoshi Obana, Hosei University Seetal Potluri, University at Albany, SUNY Seunghoon Woo, Korea University Shalabh Jain, Bosch Research Sherman S. M. Chow, The Chinese University of Hong Kong Shih-Wei Li, National Taiwan University Shuo Wang, Shanghai Jiao Tong University Siqi Ma, The University of New South Wales Sofia Celi, Brave Song Fang, University of Oklahoma Soteris Demetriou, Imperial College London Steve Granda, National Renewable Energy Laboratory Sven Dietrich, City University of New York Sze Yiu Chau, The Chinese University of Hong Kong Tingmin Wu, CSIRO's Data61 Viet Vo, Swinburne University of Technology Weijia Wang, Shandong University William Blair, SpaceX

	Xiang Li, Nankai University Xianghang Mi, USTC Xiangkun Jia, Institute of Software Chinese Academy of Sciences Xiaokuan Zhang, George Mason University Xiaoli ZHANG, University of Science and Technology Beijing Xiaoning Liu, RMIT University, Australia Xiapu Luo, The Hong Kong Polytechnic University Xinda Wang, University of Texas at Dallas Xingliang Yuan, University of Melbourne Xinlei He, Hong Kong University of Science and Technology (Guangzhou) Xueqiang Wang, University of Central Florida Xuhua Ding, SMU Yan Lin, Jinan University Yanjiao Chen, Zhejiang University Yansong Gao, Data61, CSIRO Yifeng Zheng, The Hong Kong Polytechnic University Yinzhi Cao, Johns Hopkins University Yuanchao Xu, University of California Santa Cruz Zhe Wang, ICT, CAS Zhenyu Ning, Hunan University Zhi Zhang, The University of Western Australia Zhibo Wang, Zhejiang University Ziming Zhao, Northeastern University Zubair Baig, Deakin University
Poster Reviewers	Amirmohammad Pasdar (The University of Melbourne, Australia) Soohyeon Choi (Singapore Management University, Singapore) Pham Van Thuan (University of Melbourne, Australia) Yue Duan (Singapore Management University, Singapore)

ASIA CCS'25 Sponsor and Supporters

Main Sponsors

Supporter

Supporting Partner

Bronze Sponsor

POSTER

POSTER: Transparent Temporally-Specialized System Call Filters

MATTHEW ROSSI, University of Bergamo, Bergamo, BG, Italy

MICHELE BERETTA, University of Bergamo, Bergamo, BG, Italy

DARIO FACCHINETTI, University of Bergamo, Bergamo, BG, Italy

STEFANO PARABOSCHI, University of Bergamo, Bergamo, BG, Italy

Published: 25 August 2025

ASIA CCS '25: 20th ACM Asia Conference on Computer and Communications Security
August 25 - 29, 2025
Hanoi, Vietnam

Conference Sponsors:
SIGSAC

ASIA CCS '25: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (August 2025)
https://doi.org/10.1145/3708821.3735342
ISBN: 9798400714108

POSTER: Transparent Temporally-Specialized System Call Filters

Matthew Rossi
Università degli Studi di Bergamo
Bergamo, Italy
matthew.rossi@unibg.it

Michele Beretta
Università degli Studi di Bergamo
Bergamo, Italy
michele.beretta@unibg.it

Dario Facchinetti
Università degli Studi di Bergamo
Bergamo, Italy
dario.facchinetti@unibg.it

Stefano Paraboschi
Università degli Studi di Bergamo
Bergamo, Italy
stefano.paraboschi@unibg.it

Abstract

Reducing the attack surface of the OS kernel is an effective technique to enhance the security of application workloads. In Linux systems, developers can restrict the set of available system calls by using seccomp. Although being widely adopted in browsers, container runtimes, and sandboxing tools, this approach presents some challenges: (i) applying precise filters often requires significant application modifications, which can impede developer productivity, and (ii) the transparent enforcement of filters is bound to use a single, static list with every syscall the application might ever need, resulting in overly permissive and less effective security boundaries.

In this paper, we propose an automated method to generate temporally-specialized seccomp filters tailored to the current application state. This significantly enhances the effectiveness of filters, and overcomes the major limitations associated with a single, static filter. We implement our solution by leveraging the eBPF subsystem in the Linux kernel. Specifically, we use in-kernel eBPF programs to monitor the application state and dynamically enable or disable specialized seccomp filters in response to state transitions. We discuss how this approach addresses the limitations of state-of-the-art solutions. Finally, we validate the feasibility of our proposal and show that it introduces a limited overhead.

CCS Concepts

• **Security and privacy → Software and application security**; **Access control**.

Keywords

Syscall filtering, Temporal specialization, Attack surface reduction, eBPF, Seccomp

ACM Reference Format:
Matthew Rossi, Michele Beretta, Dario Facchinetti, and Stefano Paraboschi. 2025. POSTER: Transparent Temporally-Specialized System Call Filters. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3708821.3735342

ASIA CCS '25, Hanoi, Vietnam

ACM ISBN 979-8-4007-1410-8/25/08
https://doi.org/10.1145/3708821.3735342

1 Introduction

Several research works (e.g., [7, 11]) have shown that access to unnecessary syscalls increases the risk of privilege escalation and correlates with a higher frequency of zero-day vulnerabilities. The reason is that less commonly used kernel APIs are more susceptible to bugs, whereas popular ones are more robust and tested [12].

The introduction of *seccomp* [1] represents a key advancement in safeguarding the kernel from potentially vulnerable unprivileged user space applications. Indeed, seccomp enables the specification of a syscall filter that is evaluated by the kernel with minimal overhead whenever an application invokes a syscall. This mechanism is widely employed by various applications, such as browsers, container runtimes, and sandboxing tools. However, it also introduces some challenges. Indeed, once activated, seccomp filters can only be further restricted, requiring the implementation of tight security boundaries through significant application restructuring. To become *seccomp-aware*, an application must separate its functions into distinct compartments and manage different security profiles at runtime. In practice, most (if not all) seccomp users apply the filter once, often at application startup, and never change it afterwards. This means that all required syscalls are included in a single, overly permissive filter, which exposes a wider attack surface, and diminishes the security benefits. Another issue is that developers often struggle to craft effective syscall filters, since they typically operate at higher level of abstractions (i.e., developers directly invoke library APIs rather than syscalls).

To improve the current scenario, novel research [9, 10] has introduced the promising concept of *temporally-specialized* filters that are tailored around specific application compartments. Despite being innovative, both works have shortcomings. For instance, Ghavamnia et al. [9] provide tools to generate the filters, but still require the developer to restructure the application manually. Jia et al. [10] instead greatly enhance seccomp's flexibility, but their work requires several kernel architectural changes.

This paper advances the state-of-the-art with an approach to enforce temporally-specialized filters that does not require application changes nor kernel modifications. The design relies on the eBPF kernel subsystem, and permits to transparently apply filters to the application based on its current state. In addition, we provide the developer with tools to automatically generate the filters associated with every application state. To this end, the developer is only required to identify functions that trigger state changes (e.g., any

functions that transitions a web server from an initialization to a serving phase), and to run the application in a test environment.

In the following we detail our approach and discuss the experimental evaluation, which confirms the merits of our solution and its limited performance overhead.

2 Background

This section provides a concise overview of eBPF, detailing the essential information needed to understand the rest of this paper.

eBPF [5] is a Linux subsystem that allows programs to run within the kernel in response to the execution of kernel functions. Specifically, eBPF programs are attached to designated *hook points*, and are executed whenever these are triggered. This allows for the inspection of the hook's input arguments and, for functions allowing error injection, the modification of the return value. Essentially, these programs permit to alter the kernel behavior without changing its source code or introducing custom modules. For the purposes of this paper it is important to note that (i) data structures called *maps* can be used to maintain state across multiple invocations of eBPF programs and to share data with user space, and (ii) eBPF programs loaded into the kernel undergo rigorous validation to attest their safety (e.g., a program cannot crash due to memory errors). Modern eBPF development is supported by *frontends* such as libbpf, i.e., frameworks that allow developers to write eBPF programs in a C-like dialect which is then converted into cross-platform bytecode.

3 Our approach

This section clarifies the threat model and presents our solution. We begin with a high-level overview, followed by detailed explanations of the generation and enforcement of filters.

Threat model. Similarly to the seccomp framework, our proposal limits the set of system calls available to user space applications. We consider the kernel trusted, although it may be affected by vulnerabilities. Our goal is to reduce the attack surface and better defend the kernel against an attacker gaining remote code execution due to vulnerabilities in unprivileged user space applications.

3.1 Overview

From a high-level perspective, we replace static filters, traditionally set by manually invoking the seccomp syscall, with dynamic filters enforced through eBPF. This approach requires loading a set of eBPF programs that are evaluated whenever syscalls are invoked. Note that loading eBPF programs into the kernel is a privileged operation, as it demands the `CAP_BPF` capability, and hence we delegate this task to a system administrator. This operation occurs only once at application deployment, and simply involves executing the application with a provided loader.

The uploaded eBPF programs operate either (i) in *tracing* mode, in which they collect all syscalls executed by the application and categorize them by application state, enabling the generation of filters, or (ii) in *enforcement* mode, in which the previously generated filters are activated in response to state transitions. Filters are recorded (during tracing) and loaded (during enforcement) in dedicated maps. The architecture of our solution is shown in Figure 1.

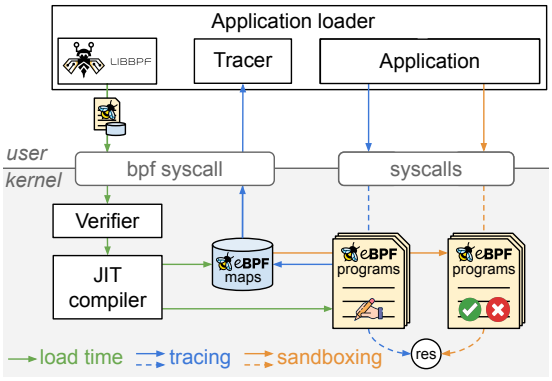


Figure 1: Tracing and sandboxing of a target application

3.2 Monitoring the application lifetime

The implementation of our approach presents two primary technical challenges: (i) tracing the processes of the target application, and (ii) detecting transitions in the application's state.

Processes are traced by injecting in the kernel a set of eBPF programs to monitor the application's lifetime. These programs are attached to the `sched_process_fork` and the `sched_process_exit` tracepoints, which are triggered by both the `fork` and `clone` syscalls, and also at process termination. Using the application's main thread id as a starting point, the eBPF programs track all child processes by adding or removing their ids in a `TYPE_HASH` map.

To detect transitions we rely on eBPF's ability to probe user space processes. Specifically, a second set of eBPF programs are activated when the application executes a transitioning function. These programs maintain the mapping between application threads and states. Every thread is associated with a single state, but multiple threads with their respective states and syscalls may coexist at runtime. We consider the definition of state transition functions a developer-provided input, which entails sharing its name with the system administrator. No application changes are required.

3.3 Generating the filters

To generate the filters, the developer runs the application in a test environment with a provided binary, which (i) prepares a process for the application's execution, (ii) records its process id in the application monitoring maps detailed in § 3.2, and (iii) launches the application via an `execve` syscall. An eBPF program is attached to the `tp_btf/sys_enter` tracepoint to capture all the syscalls issued by the application. When executed, it first reconstructs the mapping between thread id and application state using the monitoring maps, and then extracts the requested syscall id, storing it in a backing array. Upon application termination, this information is retrieved by the tracer to generate the corresponding filters (see Figure 1).

3.4 Activating the filters

The generated filters (§ 3.3) are applied at runtime without modifying the application's structure or manually invoking the seccomp syscall. The developer just provides the system administrator with the application binary and the generated filters. The system administrator then executes the application binary using the provided

Table 1: Performance of local web servers without syscall filtering, with seccomp, and with our eBPF-based solution

Software	P99 latency [ms]			Throughput [req/s]		
	Native	Seccomp	eBPF	Native	Seccomp	eBPF
Apache 2.4.58	18.64	19.11	19.51	3.46K	2.72K	2.39K
Lighttpd 1.4.74	2.79	2.91	3.34	5.45K	5.23K	4.69K
Nginx 1.24.0	6.32	6.58	6.96	3.09K	2.99K	2.93K

loader (§ 3.1). This process is sufficient to enforce the filters automatically at runtime, and requires no additional actions.

Filter enforcement is handled by a separate eBPF program attached to kernel probes that monitors all syscalls issued by the application's threads. If a thread's application state does not allow a specific syscall, i.e., it is absent from the backing array representing the filter, the eBPF program invokes the `bpf_override_return` helper function to inject an error. This means that the kernel code implementing the syscall is not run at all, and instead a permission denied error is returned, resulting in a failed syscall invocation.

4 Evaluation

To test our approach, we first isolated the overhead introduced by eBPF with a microbenchmark that measures the performance of the lightweight `getpid` syscall, and then evaluated the impact of syscall filtering on popular web servers (Apache, Lighttpd, Nginx). In both experiments, we compared a test without protections (native case) against the use of seccomp and our eBPF-based solution. Experiments were conducted on an Ubuntu 24.04 server, kernel 6.8.0, an AMD 7985WX CPU, 256 GiB of DDR5 RAM, and 2 TB SSD.

Microbenchmark. In this preliminary experiment we measured the time to execute the `getpid` syscall over 1 million invocations. The average time was 207 *ns* when running without protections, 241 *ns* (+16.4%) with seccomp, and 309 (+28.2% w.r.t. seccomp) with our solution. This result was expected, since `getpid` is one of the shortest-lived syscalls, and because seccomp is currently the most efficient approach to filter syscalls directly within the kernel.

Web server. In this test we evaluated the overhead introduced by syscall filtering techniques on web servers. In detail, we used Wrk[1] to request the default web server page for 30 seconds, using 100 connections parallelized over 12 threads. Table 1 reports the 99th percentile of latency and the average throughput in the three different configurations. The result we obtained are promising: comparing our solution with seccomp, latency increased by 2.1% (−12.1% throughput) for Apache, by 14.7% (−10.3% throughput) for Lighttpd, and by 5.8% (−2.0% throughput) for Nginx.

5 Related Work

While seccomp remains a fundamental technique to safeguard the kernel, recent research has explored precise *temporal specialization* [9, 10] to enhance its effectiveness, with the goal of enabling fine-grained seccomp filters, tailored to specific application states. Although innovative, these approaches have limitations. For instance, Ghavamnia et al. [9] propose the generation of temporally-specialized policies, but the solution requires developers to make applications seccomp-aware. Jia et al. [10] significantly enhance seccomps's programmability, but introduce several kernel architectural changes, hindering adoption. Extensive research [7, 13, 14] has also investigated user space solutions. This offers great flexibility, but it incurs substantial overhead due to frequent context switches between kernel and user space. A promising binary rewriting technique named *zpoline* [15] can be used to avoid them, but unfortunately it cannot filter syscalls issued by dynamic libraries.

The automatic generation of syscall filters has also been widely studied. Proposed methods fall into two main categories: static and dynamic generators. Static generators [6, 8] extract the syscalls required by an application directly from its code, whereas dynamic generators [2–4] are based on runtime application monitoring. Both approaches have limitations. For instance, static generators struggle with interpreters or managed runtimes, while the coverage of dynamic generators depends on the runtime tests conducted.

[1] https://github.com/wg/wrk

6 Conclusions and future work

This paper proposed an approach to transparently apply temporally-specialized seccomp filters, without requiring modifications to the application or to the kernel. Preliminary results confirmed the introduction of a limited overhead. In future work we aim to explore various hooking strategies and investigate the integration of our solution into orchestration frameworks.

Acknowledgments

This work was supported in part by the EC under project GLACIATION (01070141), by the Italian MUR under PRIN project POLAR (2022LA8XBH), and by projects SERICS (PE00000014) and GRINS (PE00000018) in the NRRP MUR program funded by the EU–NGEU.

References

[1] 2025. *Seccomp BPF.* https://docs.kernel.org/userspace-api/seccomp_filter.html
[2] 2025. *SlimToolkit.* https://slimtoolkit.org/
[3] M. Abbadini, M. Beretta, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi. 2023. Lightweight Cloud Application Sandboxing. In *CLOUDCOM.*
[4] M. Abbadini, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi. 2023. NatiSand: Native Code Sandboxing for JavaScript Runtimes. In *RAID.*
[5] J. Corbet. 2014. *BPF: the universal in-kernel virtual machine.* https://lwn.net/Articles/599755/
[6] N. DeMarinis, K. Williams-King, D. Jin, R. Fonseca, and V.P. Kemerlis. 2020. Sysfilter: Automated system call filtering for commodity software. In *RAID.*
[7] T. Garfinkel, B. Pfaff, and M. Rosenblum. 2004. Ostia: A delegating architecture for secure system call interposition. In *NDSS.*
[8] S. Ghavamnia, T. Palit, A. Benameur, and M. Polychronakis. 2020. Confine: Automated system call policy generation for container attack surface reduction. In *RAID.*
[9] S. Ghavamnia, T. Palit, S. Mishra, and M. Polychronakis. 2020. Temporal system call specialization for attack surface reduction. In *USENIX.*
[10] J. Jia, Y. ZhuFei, D. Williams, A. Arcangeli, C. Canella, H. Franke, T. Feldman-Fitzthum, D. Skarlatos, D. Gruss, and T. Xu. 2023. Programmable system call security with eBPF. *arXiv* (2023).
[11] V. Kemerlis, Vasileios P., M. Polychronakis, and A. D. Keromytis. 2014. ret2dir: Rethinking kernel isolation. In *USENIX.*
[12] Y. Li, B. Dolan-Gavitt, S. Weber, and J. Justin. 2017. Lock-in-Pop: Securing privileged operating system kernels by keeping on the beaten path. In *USENIX ATC.*
[13] C. Linn, M. Rajagopalan, S. Baker, C. Collberg, and S. Debraya nd J.H. Hartman. 2005. Protecting Against Unexpected System Calls. In *USENIX.*
[14] S. Pailoor, X. Wang, H. Shacham, and I. Dillig. 2020. Automated policy synthesis for system call sandboxing. OOPSLA (2020).
[15] K. Yasukata, H. Tazaki, P.L. Aublin, and K. Ishiguro. 2023. zpoline: a system call hook mechanism based on binary rewriting. In *USENIX ATC.*