Practical and Flexible Kernel CFI Enforcement using eBPF

Jinghao Jia
IBM Research, UIUC
Yorktown Heights, NY, USA
Jinghao.Jia@ibm.com

Michael V. Le
IBM Research
Yorktown Heights, NY, USA
mvle@us.ibm.com

Salman Ahmed
IBM Research
Yorktown Heights, NY, USA
sahmed@ibm.com

Dan Williams
Virginia Tech, IBM Research
Blacksburg, VA, USA
djwillia@vt.edu

Hani Jamjoom
IBM Research
Yorktown Heights, NY, USA
jamjoom@us.ibm.com

ABSTRACT

Enforcing control flow integrity (CFI) in the kernel (kCFI) can prevent control-flow hijack attacks. Unfortunately, current kCFI approaches have high overhead or are inflexible and cannot support complex context-sensitive policies. To overcome these limitations, we propose a kCFI approach that makes use of eBPF (eKCFI) as the enforcement mechanism. The focus of this work is to demonstrate through implementation optimizations how to overcome the enormous performance overhead of this approach, thereby enabling the potential benefits with only modest performance tradeoffs.

CCS CONCEPTS

• **Security and privacy** → **Operating systems security**;

KEYWORDS

CFI, eBPF, On-demand, Targeted, Context-sensitive

ACM Reference Format:
Jinghao Jia, Michael V. Le, Salman Ahmed, Dan Williams, and Hani Jamjoom. 2023. Practical and Flexible Kernel CFI Enforcement using eBPF. In *1st Workshop on eBPF and Kernel Extensions (eBPF '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3609021.3609293

1 INTRODUCTION

Enforcing control flow integrity (CFI) [1] in the kernel (kCFI) can be a critical technique in preventing control-flow hijack attacks [2, 5, 7]. Existing kCFI approaches typically involve compiling a simple policy and enforcement mechanism into the binary to improve execution efficiency [8]. However, this efficiency comes at the cost of deployment flexibility and policy expressiveness. Specifically, with compilation-based approaches, deploying kCFI requires rebooting the system to the new kernel, resulting not only in service interruption but also making it difficult to enable/disable kCFI. Moreover, these simple type-based target policies embedded into the binary are difficult to dynamically adjust and cannot leverage evolving runtime contexts [3, 6, 9].

SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0293-8/23/09…$15.00
https://doi.org/10.1145/3609021.3609293

Our key insight is that programmability of eBPF kernel extensions allows us to safely and efficiently provide the flexibility and context sensitivity needed for kCFI policies in modern, dynamic systems. We propose an eBPF-based kCFI approach called eKCFI that makes use of eBPF as the enforcement mechanism. In effect, each time an indirect control-transfer instruction is reached, an eBPF program is invoked to decide the legitimacy of the target destination. This approach is similar to KRIe [4] except we focus on optimization techniques to support richer CFI use cases.

Leveraging eBPF provides opportunities to extend how kCFI can be used and ease the deployment of kCFI. With our approach, kCFI can be enabled/disabled on-demand, providing security administrators an important tool to adjust their defenses during runtime. In addition, enforcement can be targeted, for example, CFI can be applied to specific areas of the kernel, during a specific time, when a specific network connection is made, or when a specific user/process is run. Furthermore, because of the programmability of eBPF, eKCFI can support a wide variety of policies that can be updated at runtime, from simple target-lookup policies obtained via static or dynamic approaches to more complex and precise policies that utilize execution context.

The main impediment in realizing the outlined opportunities is overcoming the enormous performance overhead that results from invoking an eBPF program at each indirect control-transfer site. The focus of this work is to overcome this impediment. In the next section, we discuss the three main challenges and outline their corresponding solutions.

2 REDUCING ENFORCEMENT COSTS

Reduce invocation cost Invoking an eBPF program is a significant overhead in our approach and one that must be paid regardless of the complexity of the enforcement policy. Hence, we focus our initial efforts on reducing this overhead.

In an obvious, but naïve implementation, a kprobe-based eBPF program is attached to an indirect control-transfer instruction. Invoking an eBPF program in this way can be extremely costly due to the context switches of kprobe. To gauge the overhead and assess the feasibility of our optimization proposal, we measure the latencies associated with different techniques for calling an enforcement routine, shown in Table 1.

The first measurement is of the latency of a vanilla kernel invoking a single indirect function call. This represents the latency of no CFI enforcement and is the best possible performance any kCFI

technique can strive for (unlikely attainable). We then measure the overhead of an efficient and widely available LLVM-based kCFI enforcement mechanism where the integrity check is inlined right before an indirect control-transfer site [8]. Next, we measure the overhead of invoking an eBPF-based enforcement program using the naïve kprobe attaching method. This approach represents the worst case for invoking an eBPF program. Our eBPF program implements a minimal CFI policy — it only checks whether the control transfer target is a null pointer. We register the kprobe program right before the indirect call site. As can be seen in Table 1, the LLVM-based kCFI technique is only slightly worse than no CFI enforcement while the kprobe-based eBPF approach is almost 24x slower.

Clearly, to make our approach feasible, we need to reduce the latency of invoking an eBPF program. We observe that the major overhead of the kprobe-based approach is the taking and handling of the kprobe interrupt. To eliminate this overhead, we can instead insert a tracepoint-style attachment point, which essentially invokes the eBPF program synchronously without requiring an interrupt. The latency of this method is shown in Table 1 and is significantly better than the kprobe approach implementing the same policy. The overhead associated with the tracepoint approach includes checking whether tracepoint is enabled and setting up tracepoint context for the target eBPF program. If we can reduce or eliminate the overhead associated with tracepoint, our remaining overhead would just be the cost of calling an eBPF program. This observation leads us to our most optimized invocation approach, akin to creating a custom kCFI eBPF attachment mode that inlines the eBPF program invocation. As can be seen from Table 1, this technique (with our minimal policy) results in overhead that is very close to the LLVM-based kCFI approach, giving us evidence that there is room for aggressive optimization.

To implement the optimized eBPF invocation, we plan to statically insert `nop` bytes at each indirect control-transfer site. These bytes can be dynamically patched to calls to eBPF handlers, effectively inlining the eBPF program invocation.

Table 1: Overheads for invoking integrity checks.

Invocation technique	Latency (cycles)
no checks	159 ± 2
LLVM-based inline check	176 ± 18
kprobe-based eBPF	4171 ± 1547
tracepoint-based eBPF	638 ± 48
direct (naked) eBPF	201 ± 13

Reduce policy check cost. We can accelerate the policy checks by storing the policies in a hierarchical structure so that it is possible to cache the "fast path" in which some indirect control-transfer sites that exhibit deterministic patterns can be quickly resolved. In addition, we can consolidate the same or similar control flow patterns with intelligent encoding for efficient storage and fast lookup.

Reduce indirect control-transfer checks. We propose the use of subsampling on the kernel indirect control-transfer sites. In other words, we propose the ability to enforce CFI on a select subset of kernel functionalities (targeted regions), and within the selected kernel region(s), a select subset of the CFG *guided* by either statistical means (e.g., randomly select CFG edges, which can reduce security guarantees) or by a set of security principles. For example, only select indirect sites to check that reference data directly writable by users, or similarly, drop checking indirect sites that reference data from read-only data fields.

3 CONCLUSION

Unlike existing approaches, an efficient eBPF-based kCFI mechanism can precisely target, enrich, and fine tune enforcement policies by leveraging runtime context. Furthermore, enforcement policies can change dynamically based on perceived risks such as terminate execution, send an alert, log, or fall back to a less restrictive policy.

All of this complexity will of course increase the enforcement overhead and will need to be further assessed. However, by applying the optimizations above, we believe the performance issues can be tamed. By leveraging eBPF and aggressively optimizing for the kCFI use case, we hope to provide a practical alternative kCFI approach that compliments existing compiler-based approaches.

REFERENCES

[1] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-Flow Integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security.*

[2] John Criswell, Nathan Dautenhahn, and Vikram Adve. 2014. KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. In *2014 IEEE Symposium on Security and Privacy.*

[3] Ren Ding, Chenxiong Qian, Chengyu Song, Bill Harris, Taesoo Kim, and Wenke Lee. 2017. Efficient Protection of Path-Sensitive Control Security. In *26th USENIX Security Symposium (USENIX Security 17).*

[4] Guillaume Fournier. 2022. Return to Sender - Detecting Kernel Exploits with eBPF. https://i.blackhat.com/USA-22/Wednesday/US-22-Fournier-Return-To-Sender.pdf. (2022). Accessed 2023.

[5] Xinyang Ge, Nirupama Talele, Mathias Payer, and Trent Jaeger. 2016. Fine-Grained Control-Flow Integrity for Kernel Software. In *IEEE European Symposium on Security and Privacy.*

[6] Hong Hu, Chenxiong Qian, Carter Yagemann, Simon Pak Ho Chung, William R. Harris, Taesoo Kim, and Wenke Lee. 2018. Enforcing Unique Code Target Property for Control-Flow Integrity. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.*

[7] Jinku Li, Xiaomeng Tong, Fengwei Zhang, and Jianfeng Ma. 2018. Fine-CFI: Fine-Grained Control-Flow Integrity for Operating System Kernels. *IEEE Transactions on Information Forensics and Security* (2018).

[8] LLVM. 2023. Control Flow Integrity Design Documentation. https://clang.llvm.org/docs/ControlFlowIntegrityDesign.html. (2023). Accessed 2023.

[9] Ben Niu and Gang Tan. 2015. Per-Input Control-Flow Integrity. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.*