Thesis Description
Københavns Universitet - DIKU

Francisco Correia jvz689

December 12, 2024

The widespread adoption of extended Berkeley Packet Filter, aka eBPF ([1]), programs for high-performance, programmable kernel applications has introduced new security challenges, particularly in ensuring secure information flows. eBPF programs deal with sensitive data, and since they are executed in a privileged context such as the operating system kernel, there is the possibility of unintended information leakage. As the complexity of programs eBPF supports increase, the existing tools, such as the eBPF verifier, face limitations, including high rate of false positives, poor scalability and lack of tracking for sensitive data propagation effectively. These limitations highlight the need for new tools, such as an improved verifier ([2]). This thesis focus on the latter issue, by targeting the gap in the safety of information flows within eBPF programs.

This thesis proposes a solution for this problem, the formalization and development of a framework for information flow analysis ([3]) in eBPF programs. Flow analysis refers to the examination and evaluation of data or control flow within a program or system to identify potential issues, such as resource leaks and other vulnerabilities. The framework will employ taint analysis ([4]) techniques to track the flow of sensitive data, classifying it using binary security levels, Low and High. By providing detailed insights into the security classification of the registers and memory cells at various points of execution, the framework aims to prevent potential leaks of secret information, therefore improving the security and reliability of eBPF programs. Additionally, the framework will ensure non-inference, meaning that *"Low-security behavior of the program is not affected by any high-security data."* ([5]).

This framework will be modeled and formalized using operational semantics and context-sensitive flow rules, using mathematical constructs to prove soundness, ensuring no violations are missed, and also completeness, detecting all violations. Additionally, Fixpoint convergence techniques will also be employed, ensuring that the analysis reaches a stable state after a finite number of steps.

Since the analysis works with assembly code, the flows are more complex than those in higher-level languages, which use constructs like while loops, if statements and for loops. In this case, the analysis works with jumps, which can create unusual high-security contexts, these happen when a conditional jump that relies on a secret occurs, meaning that inside that context changes can be used to identify the value of a secret. To handle this contexts correctly, I will utilize a dominance frontier analysis. the dominance frontier is an important concept as it helps to precisely indicate the border between a high context region and a low context region.

The analysis of an eBPF program follows a straightforward workflow. The original program is converted into bytecode, which is then disassembled into an assembly program, using ([6]), and finally this program is analyzed by the framework. Real-world scenarios, such as credential leaks in network traffic, unauthorized memory access through system calls and sandboxed execution environments, will validate the framework's utility.

This research addresses a rather untouched gap in eBPF studies, offering both theoretical contributions to information flow security and practical tools for enhancing eBPF's safety in production environments. Furthermore, this framework can be adapted to accommodate advances in the eBPF capabilities, ensuring that the security remains robust. As the threats become more sophisticated, the framework can

be update accounting for those, by renovating the analysis techniques, ensuring the framework remains relevant in protecting the sensitive data.

Learning goals:

1. Be capable of formalizing the analysis.
2. Prove the soundness of the analysis using operational semantics.
3. Be able to Design the information flow analysis using techniques such as, taint analysis, non-inference and fix point computation.
4. Implement the taint analysis handling security contexts.
5. Analyze test outputs in real-world scenario to evaluate the effectiveness of the analysis.

References

[1] *eBPF:* `https://ebpf.io`

[2] Gershuni, E., Amit, N., Gurfinkel, A., Narodytska, N., Navas, J. A., Rinetzky, N., Ryzhyk, L., & Sagiv, M. (2019). *Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions.* Proceedings of the 2019 ACM SIGPLAN Conference on Programming Language Design and Implementation, 1–14. DOI: 10.1145/3314221.3314590.

[3] Sabelfeld, A., & Myers, A. C. (2003) *Language-based information-flow security.* IEEE Journal on Selected Areas in Communications, 21(1), 5–19.

[4] Livshits, B. (2012). *Dynamic Taint Tracking in Managed Runtimes.* Microsoft Research, Tech. Rep. MSR-TR-2012-114. Available: `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-6.pdf`.

[5] Goguen, J. A., & Meseguer, J. (1982). *Security Policies and Security Models.* Proceedings of the IEEE Symposium on Security and Privacy, 11–18.

[6] Ken Friis Larsen. (2020). *ebpf-tools* [Source code]. GitHub. Available at: `https://github.com/kfl/ebpf-tools`. Accessed: December 9, 2024.