Eliminating Timing Channels in Microkernels

MSc Project Description

Simon Vinding Brodersen(rzn875)

December 9, 2025

1 Description

1.1 Background and Motivation

For decades, the dominant threat model for software security has centered on memory-safety vulnerabilities, including buffer overflows, format-string bugs, use-after-free errors, and integer conversion flaws. These vulnerabilities have historically been the industry's "low-hanging fruit" for attackers, because exploitation is straightforward and consequences are severe. Modern cyberattacks still rely heavily on these vulnerabilities, which remain in systems software written in C and C++.

In response, the industry is increasingly adopting strong countermeasures, including memory-safe languages like Rust, sophisticated compiler mitigations (e.g., stack canaries), and dynamic analysis tools. As these defenses mature and memory exploits become harder to execute, attackers are shifting their attention to "higher-hanging fruit" including but not limited to: timing channels, cache-based side channels and interrupt-based leaks all part of the term "side-channel attacks".

Unlike memory corruption, which alters the program's state or control flow, timing attacks exploit the physical execution of the software. Timing channels arise when execution time depends on secret or security-relevant internal states. An attacker can observe execution time, and infer sensitive information such as private keys, memory-access patterns, scheduling decisions, or resource contention. Timing channels violate intended isolation boundaries at the process, VM, or network level, allowing seemingly independent programs to leak information indirectly. Such attacks have already been shown to be detrimental and as such it is not something that can be left untouched [3, 4].

A variety of countermeasures have been proposed to address these issues. In cryptography domain-specific languages such as FaCT provide transformations of potentially timing-sensitive high-level code to low-level constant-time code [1]. This ensures that in an isolated environment the low-level code will not leak information about its secrets through timing side-channels. However, practical systems rarely operate in such controlled environments: operating systems provide no guarantees regarding scheduling decisions and do not flush caches or registers on context switches. Consequently, opportunities for microarchitectural channel leaks can still occur.

Previous work has been done to attempt to prevent such leaks. For instance hardware specific extensions have been implemented which provide mechanisms for clearing vulnerable micro architectural state and guaranteeing a history-independent context-switch latency [5]. Other work has examined the work required for time protection and have provided an implementation for the seL4 microkernel showing it is possible to protect against with little impedance on performance [2].

1.2 Problem statement

Despite their importance, timing side channels remain poorly defended in mainstream systems. There exists little to no tools to stop these attacks and many programs run as if these vulnerabilities do not exist.

The central problem this thesis addresses is: *To what extent can timing side-channels be mitigated through a combination of hardware aware software design and program transformation?*

Specifically, this project investigates whether it is viable to transform standard programs into "timing-secure" variants that are resilient to observation, and how the responsibility for this security should be distributed between the underlying hardware platform and the software running on top of it.

This will be done by surveying existing microkernel implementations - with a particular focus on their context-switching mechanisms - and identifying concrete examples of how timing channels can arise in practice. Building on these observations, the project will evaluate the feasibility of extending constant-time program transformations, similar to those seen in cryptography, to general-purpose software on microkernels. The project aims to examine the hardware requirements necessary for such transformations to be efficient and what division of responsibility between hardware and software is required to enable practical timing-secure execution in microkernel based systems.

1.3 Project-Specific Learning Objectives

- Explain and understand the theoretical and practical mechanisms of side-channel leaks across hardware, OS and program layers.
- Survey existing side-channel mitigation techniques including but not limited to: hardware mechanisms and OS-level strategies.
- Design and implement a proof-of-concept program transformer to explore the practical viability of program transformation as a defense strategy.
- Evaluate the security and performance trade-offs of the implementation. In particular on an established microkernel.
- Gather the findings into design recommendations for future operating systems, hardware abstractions or tool chains, showing how side channel-secure execution could be supported in the future.

References

[1] Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. Fact: a dsl for timing-sensitive computation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 174–189, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Qian Ge, Yuval Yarom, Tom Chothia, and Gernot Heiser. Time protection: the missing os abstraction, 2018.

[3] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[4] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[5] Nils Wistoff, Moritz Schneider, Frank K. Gürkaynak, Gernot Heiser, and Luca Benini. Systematic prevention of on-core timing channels by full temporal partitioning. *IEEE Transactions on Computers*, 72(5):1420–1430, May 2023.