CAUID: Chip Against Unintended Information Disclosure

Bachelor Semester Project S5 (Academic Year 2025/26), University of Luxembourg

Bernard Steenis
bernard.steenis@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Georgi Bozhkov
georgi.bozhkov.001@student.uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Abstract

Modern malware commonly targets software security systems that operate within the same execution environment as the protected system. This creates an inherent asymmetry: the defenses depend on software components that can be disabled, bypassed, or deceived by privileged or well-crafted attacks. CAUID (Chip Against Unintended Information Disclosure) explores a hardware-based approach to attack detection that operates outside the conventional software stack, with the goal of reducing reliance on operating system integrity and narrowing the trusted computing base.

This Bachelor Semester Project continues the development of CAUID as a research system concept for signature-based monitoring.

The report first focuses on the topic of multi-pattern matching with variable-length signatures on a scientific level, highlighting how signature-length diversity affects scalability and how prior work addresses the resulting memory and performance trade-offs. In addition, the report covers system-level deployability aspects related to configuration and verification workflows during development.

The second important section, namely the technical, explains the implementations made on CAUID to improve not only its adaptability but also its configurability.

Introduction

Security mechanisms are often deployed at the same privilege level, or within the same software ecosystem, as the systems they defend. While software-based antivirus solutions are widely used, they inherently depend on the correctness and availability of the operating system, drivers, and update mechanisms. This dependency increases the attack surface: malware may attempt to tamper with monitoring components, obscure its behaviour, or exploit the update pipeline itself.

Hardware-based monitoring offers a complementary perspective. By placing parts of detection and enforcement outside the normal software execution context, a design can reduce the trusted computing base and make certain forms of tampering more difficult. CAUID follows this direction as a research prototype that aims to detect suspicious activity through signature-based matching of observed execution patterns. In principle, such designs can be lightweight and deterministic, but they also face practical challenges, particularly when signature sets grow and when signatures are not uniform in length.

To address this from a scientific perspective, the scientific part of this BSP starts by clarifying what malware represents and what is meant by a malware "signature," including why signatures can depend on context. It then states a short research question on scalability when signatures have different lengths and reviews prior work on multi-pattern matching so that the question can be answered explicitly in a concluding subsection.

In parallel, the technical part of this BSP focuses on improving the practicality and portability of the implemented security chip. It will be explained how the chip is enhanced to support signatures of different sizes and how the configuration is organised during development. This includes how signatures are entered, stored, and inspected.

The remainder of this project first introduces the required background knowledge and development envirenvironment and states the project tasks and goals.

Project Description

This Bachelor Semester Project is structured into four main parts:

- **1. Foundational Requirements** — Presents the essential background knowledge assumed for this BSP and lists the equipment and toolchain used for development and testing.
- **2. Project Tasks** — States the concrete objectives of this Bachelor Semester Project.
- **3. Scientific Part** — The scientific deliverable begins with a short subsection on malware and malware signatures, motivating why signatures can be syntactic or context-dependent. It then introduces the research question *"Does supporting many signature lengths make hardware signature matching inefficient?"* and analyzes prior work on multi-pattern matching and variable-length signatures. The scientific part ends with a concluding subsection that answers the question.
- **4. Technical Deliverable** — Describes the technical deliverable of this BSP, including how the chip supports signatures of different sizes and how the configuration and verification workflow is implemented.

1 Foundational Requirements

Knowledge

The project assumes familiarity with the following areas:

- **Hardware description languages (HDL):** ability to read and reason about modular VHDL designs and interfaces.

- **Computer architecture basics:** instruction execution, control/data paths, interrupts/context switching at a conceptual level, and basic micro-architectural observability.

Equipment

Development and experimentation were conducted using:

- **FPGA development board:** Digilent Basys 3 (used for user I/O through switches, buttons, LEDs, and on-board display resources).
- **Toolchain:** FPGA synthesis and implementation tools (Vivado) and simulation for functional validation.
- **Environment:** a simple CPU/SoC setup used to integrate CAUID as an external monitoring component and to exercise representative execution traces.

2 Project Tasks

This BSP is structured around two tasks which are withing the scientific and technical deliverables.

Task 1: Scientific deliverable

In this section, the task is to find an answer to the question: **"Does supporting many signature lengths make hardware signature matching inefficient?"** To make this possible, the scientific deliverable first introduces what malware represents and what is meant by a malware "signature," including why signatures are not always only raw code sequences but can depend on context. It then examines prior work on multi-pattern matching and variable-length signature handling, assessing how signature-length diversity influences scalability and which factors are reported as dominant bottlenecks in practice (e.g., memory growth, throughput constraints, and verification cost). The scientific section concludes with a dedicated subsection that answers the stated question based on the analyzed literature.

Task 2: Technical Deliverable

The second task is to extend CAUID as a reusable hardware security component with stronger adaptability and configurability. The goal is to support integration into a broader range of CPU-based systems while preserving a clear module boundary. In addition, the technical deliverable introduces support for signatures of varying sizes rather than a single fixed signature length. This increases flexibility for different monitoring granularities and allows the system to represent a wider variety of signature patterns. The technical deliverable also describes the configuration and verification workflow used during development, including how signatures are entered, stored, and inspected in a controlled manner to reduce the likelihood of operator error.

3 Does supporting many signature lengths make hardware signature matching inefficient?

Malware and malware signatures

In security literature, malware is commonly defined as software with malicious intent that can harm a system or its users, typically by undermining confidentiality, integrity, or availability. [7] This umbrella term includes families such as viruses, worms, and Trojan horses, but the defining characteristic is not a specific technical form; rather, it is the intentional adverse effect and the attacker's goals. [7]

A signature is a recognizable pattern used to identify known attacks or attack classes, and signature-based detection is widely used in intrusion detection and prevention systems. [8] In the simplest case, a signature is a distinctive byte sequence (or instruction sequence) that appears in a malware sample or exploit payload. However, purely syntactic signatures are fragile: malware authors can apply transformations that alter the low-level byte or instruction representation while preserving behavior, thereby evading detectors that rely only on surface form. [9]

For this reason, many works treat a malware signature as more general than a fixed substring and consider *semantics-based* characterization, where the aim is to capture behavior (what the program does) rather than the exact code (how it is written). Dalla Preda et al. argue that program semantics provides a formal model of behavior and propose a framework to reason about robustness of malware detectors under classes of obfuscations, using trace semantics and abstraction to ignore irrelevant syntactic variation. [9] Similarly, Feng et al. explicitly distinguish between syntactic signatures (e.g., instruction sequences) and semantic signatures (e.g., control-flow or data-flow properties), and synthesize signatures that describe shared malicious functionality in terms of semantic metadata and relationships between components. [10]

This also clarifies why a signature can depend on context. In networked settings, signatures are often defined over payload and protocol structure (e.g., fields and sequencing constraints), because the same byte pattern may be benign or malicious depending on where and how it appears. [8] In host-based settings, context can include execution traces, control-flow structure, and data-flow properties, where the goal is to identify a malicious computation even if its byte-level encoding changes. [9, 10] Consequently, when a system claims to perform "signature matching," the scientific question is not only *how* matches are computed, but also what kind of pattern is being matched and which contextual signals are part of the signature definition. [8, 10]

Scientific Question

Signature-based detection can be modeled as multi-pattern exact matching: given a stream of symbols (e.g., bytes), the system must determine whether any of a large set of signatures appears as a contiguous substring. In realistic rule sets, signatures are not uniform in size; instead, they span a wide range of lengths, from short patterns to significantly longer ones [4]. This length diversity is not a minor formatting issue; it changes what it means for a design to be scalable because it interacts directly with memory organization, throughput constraints, and the amount of work required to confirm matches [3, 4]. Empirical IDS studies and systems papers motivate this concern by showing that matching can dominate software processing costs and that scaling to large, diverse rule sets becomes a bottleneck [3].

A first major family of solutions uses deterministic automata for exact matching, most notably Aho–Corasick (AC) style constructions [2]. The scientific appeal of AC is that it provides predictable per-symbol processing: the stream is processed in a single pass and matching time is linear in the input length [2]. The practical drawback, emphasized in IDS-oriented research, is that AC-style automata representations can become memory-hungry as rule sets grow [3, 5, 6]. When signatures have many different lengths and share relatively little structure, the number of states and transitions that must be represented can expand substantially, translating into memory pressure (tables, pointers, transition encodings) on the hardware side [3, 5, 6]. This is why several works focus on compressing or restructuring AC-related representations: the dominant limiter is frequently memory footprint rather than raw computation [3, 5, 6].

A second family of solutions approaches the problem differently: instead of committing to a fully deterministic exact-matching structure for the entire rule set, it uses a compact *prefilter* to reject the majority of non-matching substrings quickly and performs exact verification only for the remaining candidates [1, 4]. Bloom filters provide the theoretical foundation for this idea by enabling a controlled space–error trade-off: membership queries are memory-efficient and fast, but can return false positives at a rate determined by the filter parameters [1]. For signature matching, Bloom-filter-style prefiltering is scientifically relevant because it shifts the design problem from "store everything exactly" to "store a compact summary and verify selectively" [1, 4].

Length diversity becomes a central issue again in prefilter-based designs. A straightforward method would maintain separate filtering structures for each signature length, but real rule sets are unevenly distributed across lengths, making "one structure per length" inefficient [4]. The Extended Bloom Filter (EBF) approach addresses this by augmenting filter buckets with additional metadata so that candidate verification remains small, and by reducing the need to maintain a separate structure for every possible length [4]. The EBF results show that, with suitable parameters, false positive rates can be made low and the average verification work can remain small, supporting the claim that scalability can be achieved without storing a large exact structure for all lengths simultaneously [4].

Handling *long* signatures highlights why variable-length support is non-trivial in hardware. Long patterns are costly if matched naïvely at full length because they increase storage and may require deeper verification pipelines [4]. A common strategy is to introduce a threshold length and represent long signatures via bounded-length segments, with bookkeeping to confirm that segments occur in the correct alignment and order [4]. This reframes "arbitrary-length matching" into bounded-length matching plus verification state, reducing the number of lengths the fast front-end must support directly [4].

Across these approaches, the scientific picture is consistent: variable-length signatures do not merely increase the number of stored patterns; they increase structural diversity, which stresses memory and complicates efficient matching if handled naïvely [3–6]. Deterministic automata-based designs respond by compressing transitions and restructuring state representation to fit within feasible memory budgets [3, 5, 6]. Prefilter-and-verify designs respond by limiting the work done per input symbol and accepting a controlled verification stage grounded in Bloom filter theory [1, 4]. Later work targeting large rule sets emphasizes that scalability is achieved by controlling memory growth and keeping per-symbol processing bounded, rather than treating every length independently [3, 6].

Conclusion

Yes, supporting many signature lengths *can* make hardware signature matching inefficient, primarily because length diversity amplifies memory requirements and design complexity when represented directly. [3–6]. The literature also shows that this inefficiency is not unavoidable: it can be mitigated by compressing and restructuring deterministic automata to reduce memory growth [3, 5, 6], and/or by using compact prefiltering with targeted verification (including segmentation-based methods that support long signatures through bounded-length matching plus lightweight validation) [1, 4]. In other words, signature-length diversity is a key driver of the resource–throughput trade-off, and scalable hardware solutions are those that control the memory impact of that diversity while preserving correct matching [3, 4, 6].

4 Technical Implementation

Conclusion

Acknowledgments

The author of this BSP expresses their gratitude to the supervisor, Bernard Steenis, for their provision of support, feedback, and suggestions for improvement during the making of this project.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[3] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms for intrusion detection. In *Proceedings of IEEE INFOCOM*, 2004.

[4] H. Song and J. W. Lockwood. Multi-pattern signature matching for hardware network intrusion detection systems. In *Proceedings of IEEE GLOBECOM*, 2005.

[5] V. Dimopoulos, G. Papadopoulos, and D. Pnevmatikatos. Split-AC: A memory-efficient version of the Aho–Corasick algorithm for reconfigurable hardware intrusion detection. *IEEE*, 2007.

[6] T. Song, W. Zhang, D. Wang, and Y. Xue. A memory efficient pattern matching architecture for network security applications. 2008.

[7] M. Souppaya and K. Scarfone. *Guide to Malware Incident Prevention and Handling for Desktops and Laptops (NIST SP 800-83 Rev. 1).* National Institute of Standards and Technology, 2013. DOI: 10.6028/NIST.SP.800-83r1.

[8] K. Scarfone and P. Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS) (NIST SP 800-94).* National Institute of Standards and Technology, 2007.

[9] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. A Semantics-Based Approach to Malware Detection. In *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2007. DOI: 10.1145/1190216.1190270.

[10] Y. Feng, O. Bastani, R. Martins, I. Dillig, and S. Anand. Automated Synthesis of Semantic Malware Signatures using Maximum Satisfiability. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017. DOI: 10.14722/ndss.2017.23379.

Appendices