1. Application overview

1a. Details about the applicant

First name:	Mathijs
Initials:	M.J.
Surname:	De Kremer
Contact info:	m.j.dekremer@student.vu.nl

1b. Title of research proposal

Spectrebuster: proving leakage-free speculative execution

1c. Public summary of your research proposal (1 §, max. 50 words, used 50)

When a CPU encounters a branch instruction, it predicts the next instruction. If it predicted the wrong instruction, the mistake is reverted. However, some changes to the micro-architectural state are not reverted which can leak data and comprise security. We will create formal proofs of when this is (not) possible.

1d. Technical summary of research proposal (Max. 200 words, used 200)

Modern-day CPUs start executing future instructions before the current instruction has completed. When a CPU encounters a branch instruction, it speculates on which instruction to execute next. If it mispredicted, the mistake is reverted. However, some changes to the micro-architectural state are not reverted and can be observed by an adversary. For example, an adversary can time accesses to the CPU cache to learn that a memory address was accessed during a misprediction, which can be used to leak secrets from a program. We should not try to completely prevent leaking information because that is detrimental to performance. Instead, we should prove that specific programs are free of speculative vulnerabilities on a specific CPU since every CPU implements speculative execution slightly differently. We can formally specify what kind of information is leaked in a leakage contract [1]. Currently, we lack a method of combining a leakage contract with software instructions. With Spectrebuster we will be able to prove that a program does not leak information under speculation on a specific CPU. We can omit costly lfence or cpuid instructions for programs that are proven to be safe. Spectrebuster will also be useful in finding Spectre gadgets in existing software.

Keywords (1 line, max. 5 keywords, used 5): Transient-execution attack, Spectre, Meltdown, Leakage-contract, Formal-methods

2. Research proposal (max 3 pages)

2a. Introduction to the proposed research

My long-term vision is a software-hardware ecosystem wherein hardware vendors provide leakage contracts of their CPUs and the scientific community provides tools that take as inputs software and those leakage contracts and give as output formal guarantees about speculation safety.

Context: CPUs use speculative execution to improve performance. When a CPU makes the wrong prediction about the next instruction to execute, the predicted instruction has to be reverted. Some effects, like cache state changes, remain visible even after reverting the instruction, which can be used to attack a system. In the case of Spectre, we can observe which memory addresses were accessed during speculative execution, even if the speculation was aborted. Concretely, a memory address that was accessed during speculation will have a lower access time compared to a memory address that was not accessed. We can exploit this by encoding a secret as an index into a memory address range. Before the CPU detects that reading the secret is prohibited, the secret will have been used as an index into a memory range. The index and consequently the secret can be found by timing the memory access latency for all possible addresses in the range.

Problem: An obvious solution to prevent speculative execution attacks is to disable speculative execution. However, this would have already been done if it was not detrimental to performance. The problem that we are trying to solve is: how can we prevent speculative execution attacks while at the same time not harming performance?

The fundamental research challenge this project addresses is:

How can we prevent transient-execution vulnerabilities?

State of the art:
The scientific community has focused on preventing transient-execution vulnerabilities both formally and informally. Informally and experimentally, the community has tried to modify software to run safer on vulnerable hardware. One example of this is inserting *LFENCE* instructions. Another example is *array index masking*, which prevents speculative out-of-array indexing. An overview of practical mitigations is given in [4]. These mitigations come with several drawbacks. For one, they do not provide formal guarantees. Additionally, they potentially make dangerous or incorrect assumptions about the micro-architecture. Finally, they usually only mitigate specific variants of transient-execution vulnerabilities while keeping other vulnerabilities present.
Formally, a framework to reason about micro-architectural leakage has been introduced in [1]. The notion of leakage contracts can be used to formally provide guarantees on the amount and type of leakage of hardware designs. It contains several simplifications, such as a fixed three-stage pipeline. Leakage contracts can for example express that a processor leaks (exposes) the addresses of loads during speculative execution but during sequential execution only leaks the program counter [1]. Building on [1], methods for verifying chip designs at the register-transfer level (RTL) against their specified leakage contracts have been introduced in [2]. Practical secure speculation for constant-time programs has been proven in [5].
Reverse engineering the leakage behavior of an ARM Cortex processor has been shown in [9].

2b. Proposed research

Overall aim: In Spectrebuster, I aim to:

Develop a semantics that can be used to prove the non-existence of speculative execution vulnerabilities in software.

I envision x **key objectives** (**research tracks**, RTs):

RT-1. The first research track will focus on developing a semantics for reasoning about speculation safety of a toy instruction set architecture (ISA) with a toy leakage contract.
RT-2. The second research track will focus on capturing leakage contracts on closed source ISAs via reverse engineering.
RT-3. The third research track will generalize the toy ISA semantics to support arbitrary ISAs.
RT-4. The fourth research track will introduce and compare methods for making leaky programs leakage free.

General Approach and Evaluation Methodology: My general approach is to start small and abstract and then extend to tailor to specific hardware and ISAs. I will use **general methodologies** commonly used in my field:

M1. Design of semantics [8] that are composed of leakage contracts and formal methods on assembly. Implementation as prototype by using μASM [1] to show proof-of-concept.
M2. Experimental research [6,9] to test and evaluate the mitigations from **RT-4** and to reverse engineer the leakage contracts on closed source CPUs.
M3. M3. Open-source software development [7] to provide the developed tools to the community.

Detailed approach:
RT-1 will develop formal methods that combine simple leakage contracts with a toy ISA. The leakage contract used will contain a single notion of speculation on a pipeline with a fixed number of stages. We will use μASM as a toy ISA to show proof of concept. We will prove whether a program is safe to execute under speculation for arbitrary, terminating programs. A novel contribution will be the notion of composable leakage freedom on the hardware side. We will define leakage interfaces per component of the CPU using methods described in [10]. For example, the branch predictor will have different leakage characteristics compared to the translation-lookaside buffer. Composable leakage freedom will enable us to first reason individually about all components that could leak and then combine the results into a single leakage contract. In the case of Spectre, this leakage contract will exactly describe under which conditions the hardware could encode a secret into the cache.
RT-2 will reverse engineer the leakage of a commercial Intel processor that is vulnerable to Spectre. We will start by modeling the architecture, then reverse-engineer the leakage for each pipeline stage separately, continue by combining the results from each pipeline stage and finally provide evidence for our results by observing the actual leakage from running exemplary programs. This approach has been shown to work in [9].
RT-3 will extend the toy ISA used in **RT-1** to an abstract ISA that can be used to model arbitrary ISAs. We will use a minimal set of abstract instructions that can be combined to model the complete set of concrete instructions. We will use Cambridge Sail to model x86_64 [11]. Note that since the speculative window in which an attack can occur is usually small, we can optimize our solution by only considering a small amount of sequential instructions at a time, instead of the entire program. Composable leakage freedom will also help to prevent state explosion. The developed model will be released publicly in the form of Sail specification files [11].
In **RT-4** we will compare methods for making leaky programs leakage free. In this research track, we will solely focus on software mitigations. Methods that we will explore include inserting memory fences and reordering instructions. A challenging aspect of this research track is the difference in speculative window sizes between CPUs. We will prove that the mitigations make specific examples leakage free. Another challenge of this research track is gathering a representative dataset of leaky programs. We will consult the Linux kernel (history) to find examples. We will conclude this research track with a performance comparison of the original leaky programs and their leakage free counterparts to quantify the cost of making programs leakage free. The benchmarks to reproduce the comparisons will be made open-source and we will strive to make the results fully reproducible.

2c. Research plan

Figure 1: Research project timeline

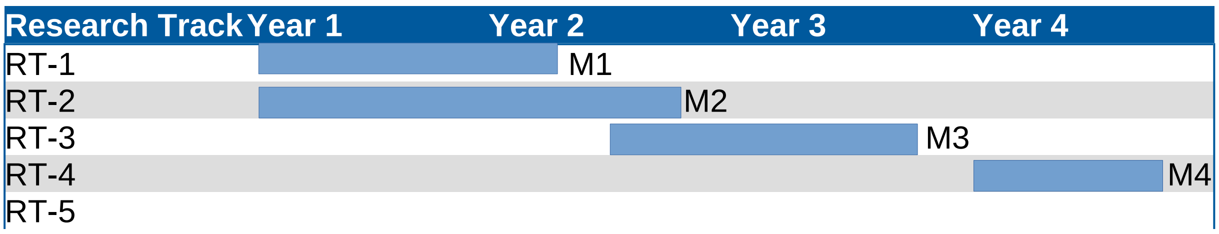

Figure 1 depicts the **practical timeline over the project duration, and the workplan for the PhD**. **RT-1** and **RT-2** are started in parallel. The other research tracks are blocked by **RT-1** and **RT-2**. All research tracks are concluded respectively at **M1** to **M4** with an article that will be presented at a conference that is relevant to the field.

KEY OUTPUT FOR THIS PROJECT:

- The first semantics that can give formal guarantees on speculation safety for software on specific hardware.
- Reversed engineered leakage contract of a recent Intel processor.

Reducing the main risks: <table with risk-mitigation plan. Each row: risk, severity/probability of occurrence/probability of detection, mitigation plan.>

Risk	**Sev./Prob.o./det./**	**Mitigation plan**
The leakage contract of the Intel processor violates intellectual property rights.	High / Med / High	We will only disclose the methodology that we used to find the leakage contract and results that do not violate intellectual property rights.
Defining a composable notion of leakage freedom is ineffective.	High / Low / High	Instead of defining a composable notion of leakage freedom, we will reason about systems in a monolithic manner. We will limit ourselves to tractable, small systems such as RISCV processors.
We fail to reverse engineer the leakage contract of the Intel processor.	High / Med / High	We will first try to contact Intel to give us information on the leakage of the processor. If Intel refuses to give this information, we will continue with an open-source RISCV processor, such as Ibex.

2d. Knowledge utilization

This project's **concrete added value** includes:

RT-1: formal methods that prove Speculation safety	**RT-2:** reverse engineering leakage contract of Intel processor	**RT-3:** extending **RT-1** with x86_64 support	**RT-4:** software mitigations
- Gives the community methods for formally reasoning about speculation safety	- Improves understanding of leakage in the community	- Helps hardware vendors to design secure processors - Optimizes hardware and software mitigations	- Improves software security for end users - Enables the community to reason about the tradeoff between security and performance

IMPORTANT: This line must appear before page 5.

3. Ethical considerations (max. 0.5 pages)

The design of a CPU is usually protected intellectual property. Since reverse engineering leakage for a CPU provides insight into the chip design, we risk disclosing protected intellectual property. Therefore, approval from an ethics committee is required before we can commence with **RT-2.**
Furthermore, our project has the potential to discover new variants of transient execution vulnerabilities. These vulnerabilities could be used by an adversary to compromise systems and leak sensitive data. Therefore, it is important that we use responsible disclosure when discovering new vulnerabilities. This means that we will first consult the ethics committee before publishing a novel vulnerability.

4. Literature references (unlimited)

[1] Marco Guarnieri, Boris Köpf, Jan Reineke, and Pepe Vila. Hardware-Software Contracts for Secure Speculation
[Online] Available: https://spectector.github.io/papers/hwsw-contracts.pdf/

[2] Zilong Wang, Gideon Mohr, Klaus von Gleissenthall, Jan Reineke, and Marco Guarnieri. Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts
[Online] Available: https://gleissen.github.io/papers/ccs2023.pdf

[3] Xaver Fabian, Marco Guarnieri, and Marco Patrignani. Automatic Detection of Speculative Execution Combinations
[Online] Available: https://dl.acm.org/doi/pdf/10.1145/3548606.3560555

[4] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L. Titzer, and Toon Verwaest. Spectre is here to stay
An analysis of side-channels and speculative execution
[Online] Available: https://arxiv.org/pdf/1902.05178.pdf

[5] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, Sophia Antipolis, and Frank Piessens. ProSPeCT: Provably Secure Speculation for the Constant-Time Policy
[Online] Available: https://www.usenix.org/system/files/usenixsecurity23-daniel.pdf

[6] Gernot Heiser. Systems Benchmarking Crimes
[Online] Available: https://gernot-heiser.org/benchmarking-crimes.html

[7] Mark Aberdour. Achieving Quality in Open-Source Software
[Online] Available: https://ieeexplore-ieee-org.vu-nl.idm.oclc.org/abstract/document/4052554

[8] Alessandro Cimatti and Stefano Tonetta. A Property-Based Proof System for Contract-Based Design
[Online] Available: https://ieeexplore-ieee-org.vu-nl.idm.oclc.org/abstract/document/6328123

[9] Si Gao, Elisabeth Oswald, and Dan Page.
Reverse Engineering the Micro-Architectural Leakage Features of a Commercial Processor.
[Online] Available: https://eprint.iacr.org/2021/794

[10] Ezio Bartoccim, Thomas Ferrère, Thomas A. Henzinger, Dejan Nickovic, and Ana Oliveira da Costa
Information-Flow Interfaces
[Online] Available: https://arxiv.org/pdf/2002.06465.pdf

[11] Kathryn E. Gray, Peter Sewell, Christopher Pulte, Shaked Flur, and Robert Norton-Wright
The Sail instruction-set semantics specification language
[Online] Available: https://www.cl.cam.ac.uk/~pes20/sail/manual.pdf

5. Statement by the applicant

I have completed this form truthfully, on my own.

Number of words used:
- public summary: **50** (max. 50 words)
- technical summary: **200** (max. 200 words)
- keywords: **5** (max. 5 keywords)

Number of pages used:
- Sections 1 and 2: **4** (max. 4 pages)
- Ethical considerations **0.5** (max. 0.5 pages)

Name: Mathijs De Kremer

Place: Amsterdam

Date: December 18, 2023