ISSN 2186-7437

NII Shonan Meeting Report

No. 215

Microarchitectural Attacks and Defenses

Sébastien Bardin
Tamara Rezk
Yuval Yarom

July 22–25, 2024

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

List of participants

Sébastien Bardin - CEA List & Université Paris-Saclay

Chitchanok Chuengsatiansup - The University of Melbourne

Lesly-Ann Daniel - DistriNet, KU Leuven

Jesse De Meulemeester - COSIC, KU Leuven

Thomas Eisenbarth - University of Lübeck

Daniel Genkin - Georgia Tech

Benjamin Gregoire - Inria

Roberto Guanciale - KTH

Marco Guarnieri - IMDEA Software Institute

Yuko Hara - Tokyo Institute of Technology

Johannes Kinder - LMU Munich

David Kohlbrenner - University of Washington

Byoungyoung Lee - Seoul National University (SNU)

Yossi Oren - Ben-Gurion University of the Negev, Israel

Kaveh Razavi - ETH Zurich

Jan Reineke - Saarland University

Tamara Rezk - Inria

Mengjia Yan - MIT

Yuval Yarom - Ruhr University Bochum

Introduction and Meeting Overview

In January 2018, two attacks called Spectre [1] and Meltdown [2] were made public and moved the history of cybersecurity to a new era. Indeed, Spectre and Meltdown, which were quickly followed by many other attacks [3] of the same class coined as transient execution attacks, demonstrated how an attacker could make use of speculative execution to exfiltrate secrets that were otherwise highly protected at the architectural level. The consequences of these attacks can be devastating and they affect most existing modern processors.

Challenges regarding transient execution attacks and defenses include the consideration of hardware speculations, which are microarchitecture optimizations mostly ignored in the area of security before 2018. Hardware speculations are extremely complex to reason about (either for humans or for program analyzers) as they yield a dramatic explosion of the number of potential behaviours to consider [4] and they are complex mechanisms not always well documented by hardware providers. Hardware manufacturers and developers are facing an unprecedented need of security mechanisms to help identify, mitigate, and remove microarchitectural vulnerabilities.

Our seminar proposal is to gather and encourage discussions among researchers and industry leaders in the area, and to provide a forum to:

- Discuss recent developments and issues regarding transient execution attacks, and more in general, microarchitectural attacks (a bigger class that include transient execution attacks);
- Discuss the effectiveness of various security mechanisms, at the hardware, system, and software levels, in the face of the current overall vulnerability landscape.

The seminar brings together leading researchers and practitioners from three different domains: (1) microarchitectural attacks, (2) system designers, and (3) formal methods. We expect that close interaction between these communities will facilitate better understanding of the area and initiate solutions that will allow broad defenses against this class of attacks.

In particular, the seminar plans to address the following questions: What are the latest trends in micro-architectural attacks and hardware speculation mechanisms? Which are the formal semantics appropriate to capture these attacks and speculations? Is it possible to design program analysis techniques that detect these attacks and/or prove their absence ? Which defenses are effective against different kinds of transient execution attacks? Which new hardware or software mechanisms could help mitigate these attacks?

Organization: To promote discussions, the seminar features breakout sessions with time to discuss different topics, and we encourage tutorials, brainstorming and working-group sessions in complement to conference-like presentations.

References

[1] Spectre Attacks: Exploiting Speculative Execution. Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. https://meltdownattack.com/

[2] Meltdown: Reading Kernel Memory from User Space. Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. https://meltdownattack.com/

[3] A Systematic Evaluation of Transient Execution Attacks and Defenses. Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss. USENIX Security Symposium 2019.

[4] Hunting the Haunter - Efficient Relational Symbolic Execution for Spectre with Haunted RelSE. Lesly-Ann Daniel, S´ebastien Bardin, Tamara Rezk. NDSS 2021.

Schedule

Sunday

15h00-23h00 Check-in possible

19h00-21h00 Reception

Monday

09h00-10h00 Opening and Participant Presentations

10h00-10h30 Coffee break

10h30-12h00 Talks
Johannes Kinder **Cats vs. Spectre: An axiomatic approach to modeling speculative execution attacks**
Lesly-Ann Daniel **Libra: Dream of Secure Balanced Execution on High-End Processors? - Let's Make it Real!**
Yuval Yarom **On the Complexity of Cache Attacks**

12h00-13h30 Lunch break

13h30-15h00 Discussion: **Secure Hardware Design** Leader: Kaveh Razavi

15h00-15h30 Coffee break

15h30-17h00 Discussion: **Modeling speculative execution semantics** Leader: Marco Guarnieri

18h00-19h00 Dinner

Tuesday

09h00-10h30 Talks
Yossi Oren **Process isolation is a lie. What else is a lie?**
Marco Guarnieri **Leakage contracts**

Thomas Eisenbarth **How much longer are we going to have to live with high precision side channels on TEEs?**

10h30-11h00 Coffee break

11h00-11h30 Working session in small groups: Hardware security, Weird machines

11h45 Group photo

12h00-13h20 Lunch break

13h30-21h00 Excursion "Visiting Engakuji and Kenchoji Temple" and Banquet

Wednesday

09h00-11h00 Talks
Byoungyoung Lee **Differential fuzzing testing to find Spectre-like vulnerabilities**
Roberto Guanciale **Validation of Side-Channel Models via Observation Refinement**
Tamara Rezk **On Kernel's Safety in the Spectre Era**

11h00-11h30 Coffee break

11h30-12h00 Jan Reineke **Synthesizing HW-SW Leakage Contracts for RISC-V Open-Source Processors**

12h00-13h30 Lunch break

13h30-15h00 Discussion: **Emerging Microarchitecture's Optimizations** Leader: Daniel Genkin

15h00-15h30 Coffee break

15h30-17h00 Discussion: **Secure compilers** Leader: Benjamin Gregoire

18h00-19h00 Dinner

Thursday

09h00-10h45 Talks
Kaveh Razavi **The Story of Branch Type Confusion**
Mengjia Yan **Formal Verification for Secure Speculation: How Can Computer Architects Help?**
Sébastien Bardin **About fault injection in program analysis**

10h45-11h15 Coffee break

11h15-11h45 Working session: Web Security

11h50 Closing

12h00-13h30 Lunch break

End of Seminar.

Discussions

We discussed different questions and ideas related to a specific topic during the seminar. We held discussions for 4 different topics detailed below. For each topic we had a leader who organized and chaired the discussion. For each discussion, we report here the questions discussed as well as the output such as references and identified challenges.

1. Secure Hardware

Leader: Kaveh Razavi

- Are there known static/dynamic metrics that correlate well with different classes of bugs?
    - What type of bugs are we talking about? Architectural vs. microarchitectural
    - Unclear whether we have good metrics (performance counters are good, really? Rebuttal: Google found some bugs with perf counters)
    - Maybe there are interesting signals we can direct fuzzing towards, but what signals? How do we select them?
- Fuzzers want some sort of feedback to improve test cases. Getting feedback from RTL is expensive. Is there hope?

    - We can use FPGAs to make fuzzing faster.
    - Question: how to do instrumentations that run on FPGAs?
    - Maybe we should focus on CPUs that run on FPGAs
    - We need LAVA for RTL, or better yet we want MAGMA.
- Formal verification research seems to always make “some” assumption for verification to become tangible, either for expressing the properties or scaling the model checking. How can we get on the same page on what it is that we are actually checking?
    - The papers are hard to follow
    - Different communities make different definitions about the same things and it is not always clear whether these definitions are equivalent
    - We cannot preserve properties across layers with formal verification of hardware

- We need to have artifacts, without that, the paper should not get accepted. But it is not always not possible to release the artifacts if one uses commercial tools
    - Commercial tools may not be working correctly
    - Open-source verifiers such as the Yosys one have obvious shortcomings and we need someone to plug them into Z3 properly
- How do we do security testing earlier in the design process?
    - Should we do fuzzing at higher levels, such as GEM5
    - Should we fuzz at a higher level of abstraction such as CHISEL?
    - We should compartmentalize different classes of bugs, and fix each compartment with a different fuzzer
- IFT verification is hot right now. What is the experience of everyone? Do we need better tooling?
    - It’s unclear whether applying IFT at a scale of an Intel CPU would work, because there will be lots of things that will get tainted
    - It’s also unclear whether Intel would want to invest so much into fixing all the security problems

- DOIT should already have strong guarantees, how did Intel check these instructions?

References:

1) BOOM is an open-source RISC-V CPU for testing
2) YoSys is standard for open source transformations and synthesis

Challenges

1) The semantic gap between RTL and the properties that we want to check
2) What sort of metrics are associated with different classes of bugs?
3) Formal verification and most fuzzing papers often lack reproducible artifacts and it is difficult to agree on what properties the different papers are actually checking.
4) Contracts in formal verification tools sometimes make it difficult to publish results and artifacts. We cannot also compare the results that we get out of these tools regarding soundness.

2. Emerging Microarchitecture's Optimizations
Leader: Daniel Genkin

- Which emerging optimization violates the leakage assumptions of the standard constant-time leakage model, and what to do about it? (e.g. silent-store optimizations, data-dependent prefetching)

Microarchitectural

- Prefetching (DMPs?)
- Compression
  - Caches
  - Register File
- Silent Stores
- Zero Register
- Lazy Register Deduplication
- Others?

Circuit Level

- Analog mess (HzBleed)
- Rowhammer
- Clock gating

- Testing
  - Finding things that violate CT or other contracts
    - Automatic mutation
    - Evolutionary compilers
    - DFTs
  - Compile it to not leak
  - Test if a CPU has these optimizations
    - Black box testing
    - Standardized benchmarks
    - compiler passes
    - Combine
  - RTL testing
- Interfaces
  - DOIT/DIT
    - Per process
    - Always on in the browser
  - Chicken bits
  - Meltdown / other optimization present bit

- Leakage models
- Engineer education
- Community communication
- Workshop at ISCA/Micro

3. Secure compilers (preservation of security or/and introduction of countermeasures)
Leader: Benjamin Gregoire

1. Protecting Compiler vs. Preserving Compiler

Protecting Compiler: The compiler automatically introduces countermeasures.
Preserving Compiler: The compiler transforms a secure program into another secure program.

2. Design of the Language

Should we design a language that allows for write-protected code (Jasmin), or should we support transformations
that automatically protect arbitrary code (Fact)? In both cases, it is important to be security-aware
(knowing who needs to be protected and when).

What about the efficiency and scalability of the resulting code?

3. Combining Countermeasures for Speculation Mechanisms

Should we study all countermeasures for each speculation mechanism simultaneously, or should we study them independently and then combine the results?
For example, replacing RSB returns with direct jumps plus conditionals, but then conditionals need to be protected as well.
How to be sure that other compilation steps do not remove countermeasures previously introduced?

4. Link Between Software and Hardware for Security Guarantees

There are currently three primary ways to protect code:

- FENCE: This can be very costly and was not initially designed for this purpose.

- Disable Speculative Mechanisms: Using hardware flags can reduce efficiency as speculation generally improves performance (SSBD).
- Software Countermeasures: Techniques like SLH (Speculative Load Hardening) transform control flow dependencies into data dependencies, which can be very efficient for well-designed cryptographic code.

However, they are limited to certain speculation mechanisms (V1, RSB, and V2 to some extent, but not V4).

Question: What kind of primitive instructions do we need to locally (only when needed) protect against speculation?

References:

- Fiat crypto : correctness garanties, cryptopt (optimize code, protection for side channel)
- Jasmin

Secure compilation for:
- Prospect
- STT (speculative taint tracking)

4. Modeling speculative execution semantics

Leader: Marco Guarnieri

What do we want to use speculative semantics for?

- Linux Kernel – “principled foundations” for (sw-level) countermeasures. Countermeasure sequences per threat model/security boundary

Can we learn something from weak-memory models?

- Long process
- Involvement from industry? Incentives? Are Intel/AMD/ARM/… willing to commit to a speculative semantics/model?
- DOIT can be a first step in “some” direction

- How low-level should the semantics be, i.e. what are good levels of abstraction?
    - No one-size-fits-all solution likely
    - It is usually tied to the security property
    - We will have a family of models/hierarchy of models
    - Secure compilation as a guideline?

- What is the connection between axiomatic semantics vs operational ones? And which are the advantages of using one or the other?

- Modeling speculation
    1. how can we model more advanced types of speculation? E.g. jump speculation, value speculation?
        - ProspeCT: predict primitive
        - Value speculation: do we expect to write programs that do not leak against complex value speculations?
    2. Jump speculation: can we do better than the naive “anything can happen”?
    3. Can we get simple and useful models for complex speculative behaviors? Eg jump speculation as anything works is evidently not useful for much :-)
    4. Does non-determinism bring any advantages when modeling speculation against the “always mid predict” approach? What about from an analysis/verification perspective? Perhaps we can just pick “same speculation” on both sides/explicitly make the non-determinism as part of the initial input

- Modeling leaks:
    1. Constant-time leaks are simple and well understood. what about more complex leaks? Eg. See discussion on emerging microarchitectural optimizations/pandora box
    2. Are you”stateless” models enough? Do we gain something by having “stateful leakage models”? Significant over-approximation needed

3. How can we handle “speculative semantics” with stateful “microarchitectural components” across speculation? Eg an rsb, a reuse table caching recent results, a BTB, …
    - In this case, does non determinism help? It might make it easier not modeling the effects of save/restore uarch components no?
    - Do we need accurate models of speculation or can we do with simplified models? What is the impact of imprecise models?
4. Are we doomed to keep coming up with new models as soon as folks discover new speculative leaks?
5. Are there some guiding principles that we can propose? Eg a “the next 700 speculative semantics” paper? Can we come up with lessons learned about our modeling efforts?
6. Where do we draw the line between model and actual system?

Challenges:

- For What/ Who are we writing models for? What is the target setting (automation vs experts manually writing crypto code)
- Are there some guiding principles that we can propose? Eg a “the next 700 speculative semantics” paper? Can we come up with lessons learned about our modeling efforts?
- What is the right level of abstraction for speculative semantics?
- How can speculative semantics guide/help developers (e.g., Linux kernel) in fixing code?
- How can hw vendors help/be involved in the discussion?

References:

- FiatCrypto
- CryptOpt
- Sail
- ProspeCT
- Contracts

Working groups topics

We held several working groups in small groups on Tuesday/Thursday.

Working group on hardware verification

1) **Lack of open source tools: can we do better?**

The lack of an optimized open source tool was raised as a challenge, not only for accessibility and reproducibility of research results, but also for trustworthiness. First, it is not possible to be sure that all optimizations implemented in commercial model checkers are sound. Second, while these commercial model checkers claim unbounded proofs, they do not make the formulas accessible, nor of they provide a proof witness,

This working group discussed the following question: is there any hope for better open-source tools?

Unfortunately, it seems to be hard to compete with commercial tools. Yet, the group identified possible directions for future work in this direction:
- Comparison between commercial tools and open sources alternatives,
- Commercial tools very likely implement hardware-dedicated model checking optimizations. Such optimizations could also be researched and integrated in open source alternatives like Yosis.

2) **Bridge the gap between security and architecture communities?**

Fuzzing and model-checking tools (usually from the security community) are not easily accessible by computer architects. How can we bridge this gap?

The development process of hardware developers usually starts with an early stage design validation for performance in a simple analytical model (Gem5). A RTL implementation only comes later, when the design as this takes more effort.

Is Gem5 a good model for security?
- It seems good for performance analysis,
- Can also be good for security testing (ruling out obvious design flows),
- However, it is very likely not a good target to make any formal security claim.

We need an abstraction close to HW we can use for security validation.
Pensieve seems to go in this direction. This could be used as a bridge between the security and hardware communities?

Working group on web-based attacks

This working group discussed novel directions in the domain of web-based attacks, exploring three directions:

- New web technologies with security impacts, including CSS :has, WebGPU, CSS Paint Worklets and new COOP/COEP headers
- New web deployments, including radio data streams, in-flight entertainment systems, embedded targets and the blockchain
- New web threat models and attacker objectives, including password stealing, deanonymization, tracking and denial of service

Talks: The abstracts of the talk during the seminar are given below

Title: Synthesizing HW-SW Leakage Contracts for RISC-V Open-Source Processors

Jan Reineke

Abstract:
Microarchitectural attacks compromise security by exploiting software-visible artifacts of microarchitectural optimizations such as caches and speculative execution. Defending against such attacks at the software level requires an appropriate abstraction at the instruction set architecture (ISA) level that captures microarchitectural leakage. Hardware-software leakage contracts have recently been proposed as such an abstraction.

In this work, we propose a semi-automatic methodology for synthesizing hardware-software leakage contracts for open-source microarchitectures. For a given ISA, our approach relies on human experts to (a) capture the space of possible contracts in the form of contract templates and (b) devise a test-case generation strategy to explore a microarchitecture's potential leakage. For a given implementation of an ISA, these two ingredients are then used to automatically synthesize the most precise leakage contract that is satisfied by the microarchitecture.

We have instantiated this methodology for the RISC-V ISA and applied it to the Ibex and CVA6 open-source processors. Our experiments demonstrate the practical applicability of the methodology and uncover subtle and unexpected leaks.

Title: Differential fuzzing testing to find Spectre-like vulnerabilities

Byoungyoung Lee

Abstract:
Finding CPU bugs through fuzzing have two challenges compared to finding software bugs. Its vulnerable behavior does not raise any explicit fault or crash, and its coverage metric is not as clear as software. In this talk, I will present about how to handle these issues through taking differential fuzz testing approaches. By running two implementations, RTL simulators and ISA emulators, using the same input, we monitor if two execution results are different. If different, it implies one of two has a bug. While running, the RTL coverage is measured using the control registers, which are registers wired into MUX's select signal. Then I will present how this fuzzer can be used for finding Spectre-like vulnerabilities through differential testing.

Title: Libra: Dream of Secure Balanced Execution on High-End Processors? - Let's Make it Real!

Lesly-Ann Daniel

Abstract:
Control-flow leakage (CFL) attacks enable an attacker to expose control-flow decisions of a victim program via side-channels observations. Linearization (i.e., elimination) of secret-dependent control flow is the main countermeasure against these attacks, yet it comes at a non-negligible cost. Conversely, balancing secret-dependent branches often incurs a smaller overhead, but is notoriously insecure on high-end processors. Hence, linearization has been widely believed to be the only effective countermeasure against CFL attacks. In this talk, I will challenge this belief and investigate an unexplored alternative: *how to securely balance secret-dependent branches on higher-end processors?*

Tutorial title: The Story of Branch Type Confusion (40')

Kaveh Razavi

Abstract:
In 2021, we were happily hijacking return instructions speculatively without knowing what we were actually doing. AMD later called it Branch Type Confusion in their advisory, which explained what was happening quite well, but not why it was happening. In this tutorial, I will briefly discuss Spectre 101 on the kernel before discussing how to bypass the first generation of mitigations that became known as retpoline. I will then discuss the root cause of Branch Type Confusion and how it enables bypassing the second generation of mitigations both in software and hardware.

Title: Leakage contracts: A foundation for microarchitectural security

Marco Guarnieri

Abstract:
Microarchitectural attacks, such as Spectre and Meltdown, illustrate that artifacts of hardware implementations (like speculative and out-of-order execution) can result in measurable side-effects on program execution time that attackers can exploit to compromise a system's security. These attacks arise from emerging behaviors obtained when combining hardware and software. Building systems that are resistant against these attacks requires fundamentally rethinking the design of existing security mechanisms.

In this talk, I will illustrate a principled approach for reasoning about security against microarchitectural attacks. The key component of this approach is a hardware-software leakage contract, a formal ISA-level specification of the information that may be leaked through microarchitectural side-effects. Concretely, I will present (1) how to model the information flows at the core of different microarchitectural attacks (e.g., Spectre), (2) how to formalize the "soundness" of a leakage contracts w.r.t. A given CPU, and (3) how to use contracts to reason about the security of hardware and software (verification, testing, secure programming, …).

Title: How much longer do we have to deal with high precision side channels on TEEs?

Thomas Eisenbarth

Abstract:
Trusted Execution Environments (TEEs), such as Intel SGX and TDX or AMD SEV promise the availability of secure processing of confidential data on remote platforms, making them a promising technology for outsourced computation in the cloud. However, most TEE platforms still feature high-precision side channels that can be exploited by adversaries controlling untrusted parts of the system such as the hypervisor or OS to infer protected code and data. We recall the impact of single stepping attacks and the frameworks SGX-Step and SEV-Step that highlight the severity and practicality of these attacks. We discuss the state of mitigation approaches taken by Intel to prevent single-stepping on their TEEs. Finally, we explore the Obelix framework that provides a comprehensive software solution to transform TEE code into uniform execution blocks to obfuscate leakage patterns, effectively countering a wide range of side channels in TEEs even for non-constant-time implementations.

Title: Validation of Side-Channel Models via Observation Refinement

Roberto Guanciale

Abstract:
Observational models (or leakage contracts) enable the analysis of information flow properties against side channels. Relational testing has been used to validate the soundness of these models by measuring the side channel on states that the model considers indistinguishable. However, unguided search can generate test states that are too similar to each other to invalidate the model. To address this we introduce observation refinement, a technique to guide

the exploration of the state space to focus on hardware features of interest. We refine observational models to include fine-grained observations that characterize behavior that we want to exclude. States that yield equivalent refined observations are then ruled out, reducing the size of the space.

Title: Process isolation is a lie. What else is a lie?

Yossi Oren

Abstract:
The concept of process isolation allows the creation of elegant and efficient complex systems. Unfortunately, process isolation does not hold in practice, with numerous works showing cross-process confidentiality breaches.

What other incorrect concepts guide the design of modern computer systems? How should we respond to them?

Title: Cats vs. Spectre: An axiomatic approach to modeling speculative execution attacks

Johannes Kinder

Abstract:
The SPECTRE family of speculative execution attacks have required a rethinking of formal methods for security. Approaches based on operational speculative semantics have made initial inroads towards finding vulnerable code and validating defenses. However, with each new attack grows the amount of microarchitectural detail that has to be integrated into the underlying semantics. We propose an alternative, light-weight and axiomatic approach to specifying speculative semantics that relies on insights from memory models for concurrency. We use the CAT modeling language for memory consistency to specify execution models that capture speculative control flow, store-to-load forwarding, predictive store forwarding, and memory ordering machine clears. We present a bounded model checking framework parametrized by our speculative CAT models and evaluate its implementation against the state of the art. Due to the axiomatic approach, our models can be rapidly extended to allow our framework to detect new types of attacks and validate defenses against them.

Title: Formal Verification for Secure Speculation: How Can Computer Architects Help? (20')

Mengjia Yan

Abstract:
As computer architects continually push the limits of microarchitectural performance optimizations, securing hardware against speculative execution attacks feels like a Sisyphean challenge. Formal methods offer rigorous, machine-checked verification and have the potential to significantly strengthen the security of processor designs. However, there exist research and

community gaps in adopting these promising techniques. In this talk, I will discuss this existing gap and the potential role computer architects can play in bridging this gap. Moreover, hopefully, the talk can inspire researchers on the formal side to build tools that are more accessible to computer architects.

Title: On Kernel's Safety in the Spectre Era

Tamara Rezk

Abstract:
The efficacy of address space layout randomization has been formally demonstrated in a shared-memory model by Abadi et al., contingent on specific assumptions about victim programs. However, in practice, attacks such as Blindside use speculative execution and side-channels to break layout randomization and lead to memory corruption. In this talk, I will discuss these threats and explore possible paths to recover kernel safety in the Spectre era.

Title: On the Complexity of Cache Attacks

Yuval Yarom

Abstract:
Cache attacks, which leak sensitive information through observation of memory access patterns, have been intensively investigated over the past two decades. Multiple attack methodologies have been devised, a large number of software implementations have been shown vulnerable, and various defenses have been proposed. Yet, little attention has been given to fundamental properties of such attacks and their interaction with cache state. This talk extends our understanding of cache attacks, demonstrating that they can perform arbitrary computation on cache state. It associates a logical value with the property of cache presence of memory location. It then presents a methodology for constructing “weird gates”, which allow performing logical operations on the logical states. Using these weird gates, it then demonstrates how to enable arbitrary computation. Finally, it discusses the security implication of arbitrary computation on microarchitectural states.

Title: About Fault Injection in Program Analysis (10')

Sébastien Bardin

Abstract:
This talk will depart slightly from the core topic of the seminar, by presenting the problem of physical fault injection attacks, where the attacker is able to perturbate the normal flow of the concrete execution of the program in order to reach unintended behaviors, typically bypassing security checks or leaking sensitive information. While historically restricted to highly security-critical domains such as smart cards and very well equipped attackers (laser beam, EM pulse), these classes of attacks are likely to become much more widespread with the advent of software-induced hardware attacks, such as rowhammer. We present the main challenges for handling them in a formal verification setting, i.e. defining a suitable attacker model and scaling the analysis despite the many possible actions from this attacker, as well as some solutions we have developed with the BINSEC framework. Finally, we sketch some connexions with the formal verification of speculative behaviors, possibly tightening together these two trends of security research.

Talk based on:

- Adversarial Reachability for Program-level Security Analysis. Soline Ducousso, Sébastien Bardin, Marie-Laure Potet. The 32nd European Symposium on Programming

- Inference of Robust Reachability Constraints. Yanis Sellami, Guillaume Girol, Frédéric Recoules, Damien Couroussé, Sébastien Bardin. 51st ACM SIGPLAN Symposium on Principles of Programming Languages

- Hunting the Haunter -- Efficient Relational Symbolic Execution for Spectre with Haunted RelSE. Lesly-Ann Daniel, Sébastien Bardin, Tamara Rezk. The 28th Network and Distributed System Security Symposium