EE 156: Advanced Topics in Computer Architecture

Spring 2023
Tufts University

Instructor: Prof. Mark Hempstead
mark@ece.tufts.edu

Lecture 10: Secure Archiectures and Meltdown/Spectre solutions (InvisiFence)

Outline

Unit 3: Security from the Hardware/Systems Perspective (2 weeks)

- Security Principles [SLCA: R. Lee]
- **Secure Architectures** and Secure Memory
- Side-Channels and Examples
  - Rowhammer
  - EM (Eddie)
- Hardware Security and Side-Channel Attacks
  - **Spectre and Meltdown**
  - **Mitigations**

Security References

- At least in Computer Architecture the field is relatively new and moving quickly. It's not really covered in your textbook
  - Spectre and Meltdown were published Jan 2018
  - Side channel attacks have gained prevalence over the last few years: Timing, Rowhammer, RF/EM, Power and Thermal
- Useful References:
  - Ruby B. Lee Security Basics for Computer Architects Synthesis Lectures on Computer Architecture September 2013 (https://doi.org/10.2200/S00512ED1V01Y201305CAC025)
  - Prof. Simha Sethumadhavan: http://www.cs.columbia.edu/~simha/
  - Prof. Srini Devadas: https://people.csail.mit.edu/devadas/

Caveats

- This unit will not cover all aspects of security
  - Network Security and Intrusion Detection
  - Cryptography
  - Software exploits (buffer overflow etc..)
- We will cover basic principles found in other aspects of security but take a hardware view
  - Hardware can help security by providing trust and hardware support for security (encryption, authentication and security levels)
  - Hardware can be exploited through physical access and physical side channels

Secure Architectures

Principles of Secure Processor Architecture Design

Slides and information available at:

http://caslab.csl.yale.edu/tutorials/hpca2019/

Brief History of Secure Processor Architectures

Starting in late 1990s or early 2000s, academics have shown increased interest in secure processor architectures:

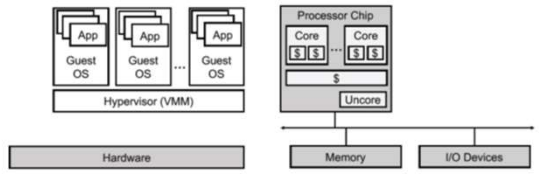
XOM (2000), AEGIS (2003), Secret-Protecting (2005), Bastion (2010), NoHype (2010), HyperWall (2012), CHERI (2014), Sanctum (2016), Keystone (about 2017), MI6 (2018)

Commercial processor architectures have also included security features:

LPAR in IBM mainframes (1970s), Security Processor Vault in Cell Broadband Engine (2000s), ARM TrustZone (2000s), Intel TXT & TPM module (2000s), Intel SGX (mid 2010s), AMD SEV (late 2010s)

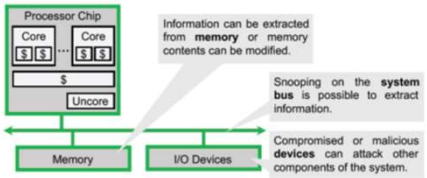
Baseline (Unsecure) Processor Architecture

A simplified view of a processor and the software stack in a general-purpose computer:



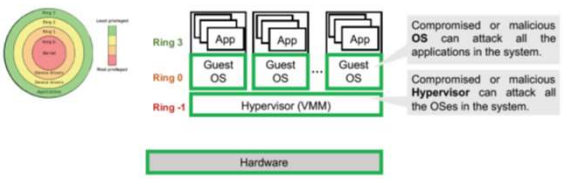
Baseline (Unsecure) Processor Hardware

Typical computer system with no secure components nor secure processor architectures considers all the components as trusted:



Baseline (Unsecure) Processor Software

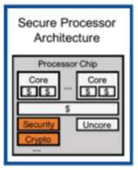
Typical computer system uses ring-based protection scheme, which gives most privileges (and most trust) to the lowest levels of the system software:



Protecting from Software and Hardware Attacks

Secure Processor Architectures add new hardware and software features to provide **Trusted Execution Environments (TEEs)** wherein software executes protected from some of the software and hardware threats.

- Enhance general-purpose processor with new protection features
- Provide new or alternate privilege levels
- Utilize software and hardware changes
- Facilitate attestation of the protected software



Tutorial on Principles of Secure Processor Architecture Design © Jakub Szefer (ver. HPCA 2018)

18

New Privilege Levels

Modern computer systems define protections in terms of **privilege level** or protection rings, new privilege levels are defined to provide added protections.

Ring 3	Application code, least privileged.
Rings 2 and **1**	Device drivers and other semi-privileged code, although rarely used.
Ring 0	Operating system kernel.
Ring -1	Hypervisor or virtual machine monitor (VMM), most privileged mode that a typical system administrator has access to.
Ring -2	System management mode (SMM), typically locked down by processor manufacturer
Ring -3	Platform management engine, retroactively named "ring -3", actually runs on a separate management processor.

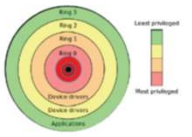


Image: https://commons.wikimedia.org/wiki/File:Priv_rings.svg

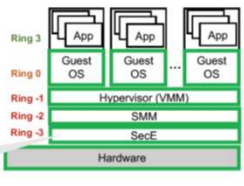
Tutorial on Principles of Secure Processor Architecture Design © Jakub Szefer (ver. HPCA 2018)

19

Extend Linear Trust with New Protection Levels

The hardware is most privileged as it is the lowest level in the system.

- There is a linear relationship between protection ring and privilege (lower ring is more privileged)
- Each component **trusts** all the software "below" it



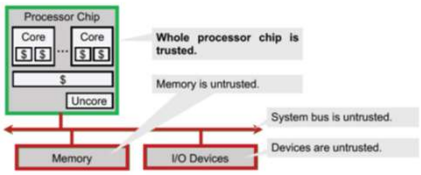
Security Engine (SecE) can be something like Intel's ME or AMD's PSP.

Tutorial on Principles of Secure Processor Architecture Design © Jakub Szefer (ver. HPCA 2018)

20

Trusted Processor Chip Assumption

Key to most secure processor architecture designs is the **trusted processor chip assumption**.



Trusted Computing Base

Trusted Computing Base, or **TCB**, is the sum total of all the hardware and software which work together to realize the protections offered by the system.

- TCB is trusted
- TCB may not be trustworthy, if is not verified or is not bug free

TCB contains:

- All trusted hardware – typically the processor chip
- All trusted software – some software levels may be untrusted (e.g. OS in SGX)

Today's Limitations of Secure Architectures

Threats which are outside the scope of secure processor architectures:

- Bugs or Vulnerabilities in the TCB — TCB hardware and software is prone to bugs just like any hardware and software.
- Hardware Trojans and Supply Chain Attacks — Modifications to the processor after the design phase can be sources of attacks.
- Physical Probing and Invasive Attacks — At runtime hardware can be probed to extract information from the physical realization of the chip.

Threats which are underestimated when designing secure processor architectures:

- Side Channel Attacks — Information can leak through timing, power, or electromagnetic emanations from the implementation

Trusted Execution Environments and TCB

The goal of **Trusted Execution Environments (TEEs)** is to provide protections for a piece of code and data from a range of software and hardware attacks.

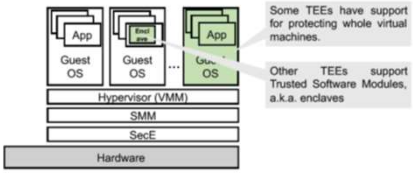
- Multiple mutually-untrusting pieces of protected code can run on a system at the same time

The **Trusted Computing Base (TCB)** is the set of hardware and software that is responsible for realizing the TEE:

- TEE is created by a set of all the components in the TCB
- TCB is trusted to correctly implement the protections
- Vulnerability or successful attack on TCB nullifies TEE protections

TEEs and Software They Protect

Different architectures mainly focus on **protecting Trusted Software Modules** (a.k.a. enclaves) or **whole Virtual Machines** or containers.



Some TEEs have support for protecting whole virtual machines.

Other TEEs support Trusted Software Modules, a.k.a. enclaves

Protections Offered by Secure Processor Architectures

Security properties for the TEEs that secure processor architectures aim to provide:

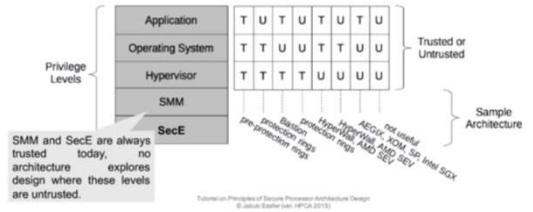
- Confidentiality — Confidentiality is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities.
- Integrity — Integrity is the prevention of unauthorized modification of protected information without detection.
- Availability (next slide)

The C. I. A. properties are with respect to components or participants of the system, commonly named Alice, Bob, Charlie, Eve, Malory, etc., in different protocols

Confidentiality and integrity protections are from attacks by other components (and hardware) not in the TCB. **There is typically no protection from malicious TCB.**

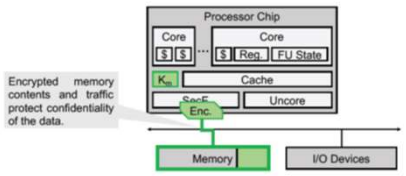
Sample Protections Categorized by Architecture

Secure processor architectures break the linear relationship (where lower level protection ring is more trusted):



Enforcing Confidentiality through Encryption

Symmetric key cryptography should be used to protect data going off chip to prevent hardware attacks.



Performance Overhead of Securing TCB

Impact of threat model on performance:

- Protecting against more threats typically adds more overhead
- Memory encryption and integrity checking are the most expensive part, but really depends on how defense is implemented

- Secure caches: 1~10% overhead
- Spectre protections: initially stated >10%, now most <10%
- Memory encryption: can be >100%

More protections, must not mean less performance:

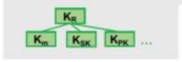
- Partitioning
- Randomization is not always bad

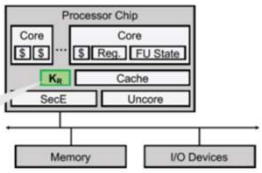
Root of Trust and the Processor Key

Security of the system is derived from a **root of trust**.

- A secret (cryptographic key) only accessible to TCB components
- Derive encryption and signing keys from the root of trust

Hierarchy of keys can be derived from the root of trust





Tutorial on Principles of Secure Processor Architecture Design © Jakub Szefer (ver. HPCA 2018)

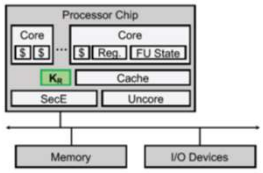
62

EE156/CS140 Mark Hempstead 22

Root of Trust and Processor Key

Each processor requires a unique secret.

- **Burn in at the factory** by the manufacturer (but implies trust issues with manufacturer and the supply chain)
  - E.g. One-Time Programmable (OTP) fuses
- Use **Physically Uncloneable Functions** (but requires reliability)
  - Extra hardware to derive keys from PUF
  - Mechanisms to generate and distribute certificates for the key



Tutorial on Principles of Secure Processor Architecture Design © Jakub Szefer (ver. HPCA 2018)

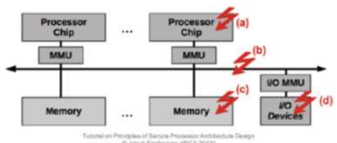
63

EE156/CS140 Mark Hempstead 23

Sources of Attacks on Memory

Memory is vulnerable to different types of attacks:

a) Untrusted software running no the processor
b) Physical attacks on the memory bus, other devices snooping on the bus, man-in-the-middle attacks with malicious device
c) Physical attacks on the memory (Coldboot, …)
d) Malicious devices using DMA or other attacks



Tutorial on Principles of Secure Processor Architecture Design © Jakub Szefer (ver. HPCA 2018)

79

EE156/CS140 Mark Hempstead 24

Types of Attacks on Memory

Different types of attacks exist (very similar to attacks in network settings):

- Snooping — Passive attack, try to read data contents.
- Spoofing — Active attack, inject new memory commands to try to read or modify data.
- Splicing — Active attack, combine portions of legitimate memory commands into new memory commands (to read or modify data).
- Replay — Active attack, re-send old memory command (to read or modify data).
- Disturbance — Active attack, DoS on memory bus, repeated memory accesses to age circuits, repeated access to make Rowhammer, etc.

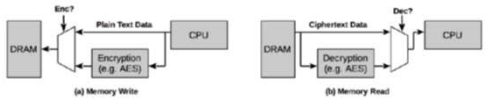
80

Confidentiality Protection with Encryption

Contents of the memory can be protected with encryption. Data going out of the CPU is encrypted, data coming from memory is decrypted before being used by CPU.

a) Encryption engine (usually AES in CTR mode) encrypts data going out of processor chip
b) Decryption engine decrypts incoming data

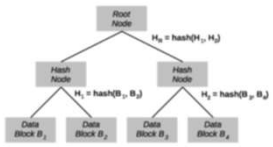
Pre-compute encryption pads, then only need to do XOR; speed depends on how well counters are fetched / predicted.



81

Integrity Protection with Hash Trees

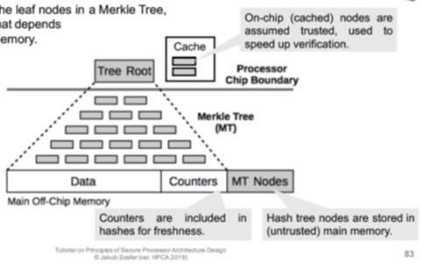
Hash tree (also called **Merkle Tree**) is a logical three structure, typically a binary tree, where two child nodes are hashed together to create parent node; the root node is a hash that depends on value of all the leaf nodes.



82

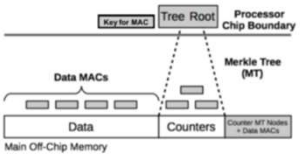
Integrity Protection with Hash Trees

Memory blocks can be the leaf nodes in a Merkle Tree, the tree root is a hash that depends on the contents of the memory.



Integrity Protection with Bonsai Hash Trees

Message Authentication Codes (MACs) can be used instead of hashes, and a smaller "Bonsai" tree can be constructed.



InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan† , Jiho Choi† , Dimitrios Skarlatos,
Adam Morrison⁎, Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign ⁎Tel Aviv University

†Authors contributed equally to this work.

Motivation: Speculative Execution Attacks

- Hardware speculative execution offers a big attack surface for covert and side channels

- Speculative execution attacks exploit the side effects of instructions on incorrect speculative

Compilers and programmers can not reason about it.

- It is crucial to fix this vulnerability efficiently

Speculative Execution Attacks

- An example of Spectre Variant 1 (array bound checking attack).

Victim code

```
1:  if (x < array1_size) {
2:      val = array1[x]
3:      ld array2[val]
4:  }
```

Leaves side effects in cache

Attack to read arbitrary memory:

1) Train branch predictor
2) Trigger branch misprediction
3) Side channel

Speculative execution attacks exploit side effects of instructions on paths that will be squashed

Generalization of Speculative Execution Attacks

- Transient instructions: speculatively-executed instructions that are destined to be squashed.
- Speculative execution attack exploits side effects of transient instructions.

```
1:  if (x < array1_size) {
2:      val = array1[x]
3:      ld array2[val]
4:  }
```

Transient Instruction

Attack	Sources of Transient Instructions
Spectre	Control-flow misprediction
Meltdown	Virtual memory exception
L1 Terminal Fault	
Speculative Store Bypass	Address alias between a load and an earlier store

Futuristic Speculative Attack Model

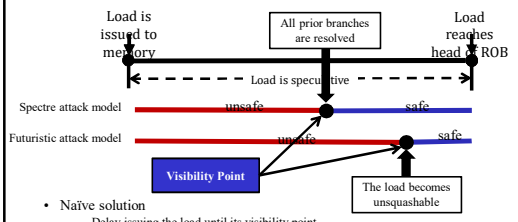
34

- Futuristic speculative attack model
  - An attacker can exploit ***any*** speculative load (load not at the head of ROB).
  - It includes all existing attacks and future speculative execution attacks

Attack Model	Sources of Transient Instructions
Futuristic	Various events, such as: • Exceptions • Control-flow mispredictions • Address alias between a load and an earlier store • Address alias between two loads • Memory consistency model violations • Interrupts

Lifetime of a load instruction

35

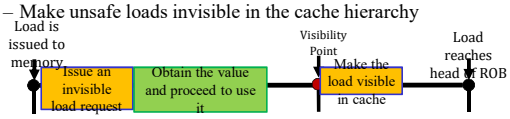


- Naïve solution
  - Delay issuing the load until its visibility point

InvisiSpec

36

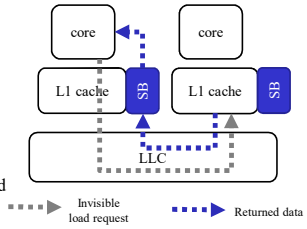
- The first holistic defense mechanism against speculative execution attacks
- Key idea:
  - Make unsafe loads invisible in the cache hierarchy



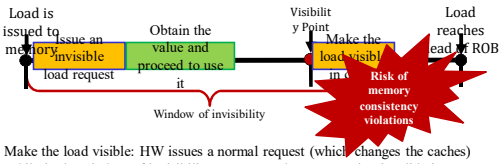
Speculative loads are issued as early as in a conventional machine

Making Unsafe Loads Invisible

- Invisible load request
  - No modification to cache states, including
    - Cache occupancy
    - Replacement information
    - Coherence information
    - TLB state
  - Bring the data, store in Speculative Buffer (SB) and in register



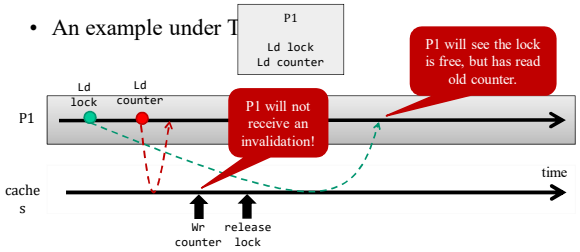
Making Safe Loads Visible



- Make the load visible: HW issues a normal request (which changes the caches)
- While in the window of invisibility, processor does not receive invalidations

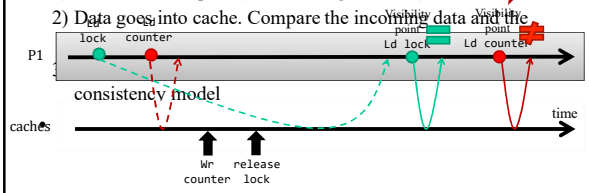
An Example of Memory Consistency Violation

- An example under T



Maintaining Memory Consistency

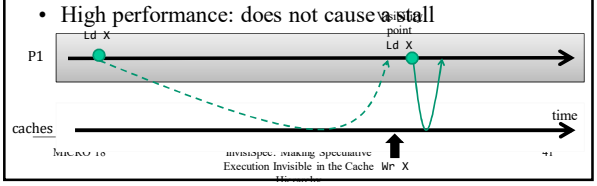
- Need to issue another access at Visibility Point: Validation
  1) HW issues a request (which changes caches)
  2) Data goes into cache. Compare the incoming data and the consistency model



Maintaining Memory Consistency (II)

- For loads with no risk of memory consistency violation: Exposure
  - HW issues a request at the Visibility Point
  - Load does not need to wait for r
  - Data goes into cache. No need t
- High performance: does not cause a stall

See the paper for the many cases where a load can use exposures



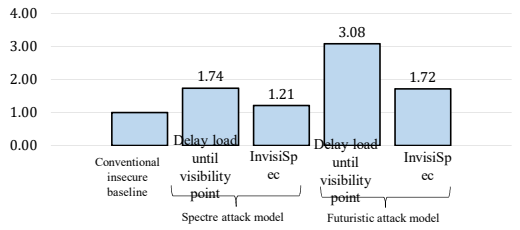
Pros & Cons of InvisiSpec

- Security
  - Successfully prevent attacks in both Spectre and Futuristic attack models
- High performance
  - Speculative loads are issued as early as in a conventional machine
- Applicability
  - Handle multi-threaded issues
- No software changes

- Performance overhead
  - Double accesses
  - May stall due to validations

BUT:
- Many can be converted to exposures
- Most hit in L1, and return very quickly

Average Execution Time for SPEC and PARSEC (Normalized)



Conclusion

44

- InvisiSpec is the first comprehensive defense mechanism against speculative execution attacks in the cache hierarchy
- We published the code of our architecture at:

https://github.com/mjyan0720/InvisiSpec-1.0