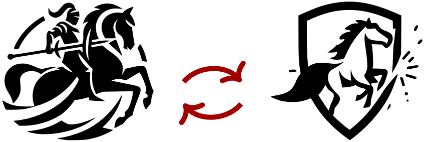


On the Interplay between Attacks and New Defenses

The Story of SGX-Step and Transferable Insights for Other Architectures

Jo Van Bulck

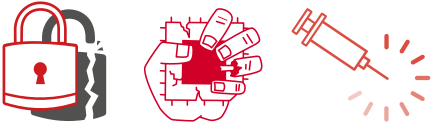
Huawei – KU Leuven research collaboration workshop, Leuven, March 6, 2024

🏠 DistriNet, KU Leuven, Belgium ✉ jo.vanbulck@cs.kuleuven.be 🐦 jovanbulck

About imec-DistriNet enclave research

https://distrinet.cs.kuleuven.be/

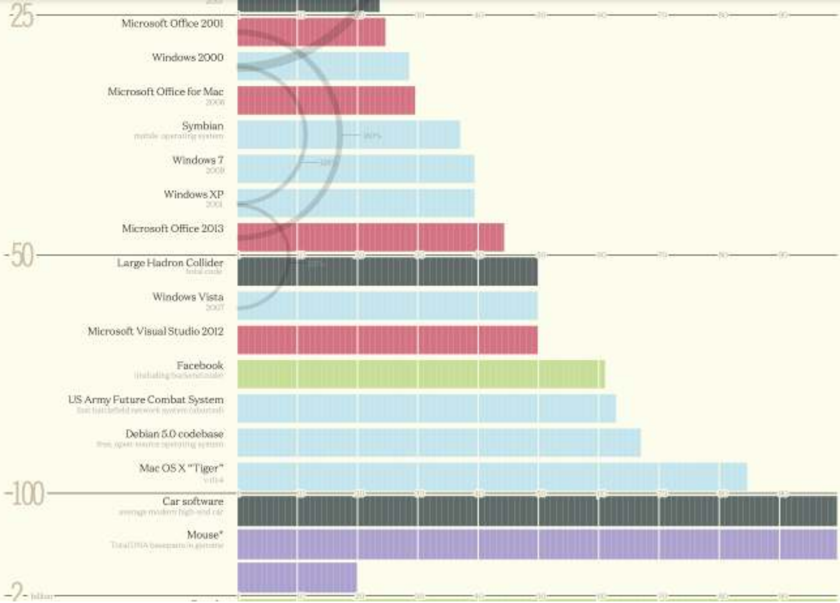
- Trusted computing **across the system stack:** hardware, compiler, OS, apps
- Integrated **attack-defense** perspective and **open-source** prototypes



Transient-execution attacks

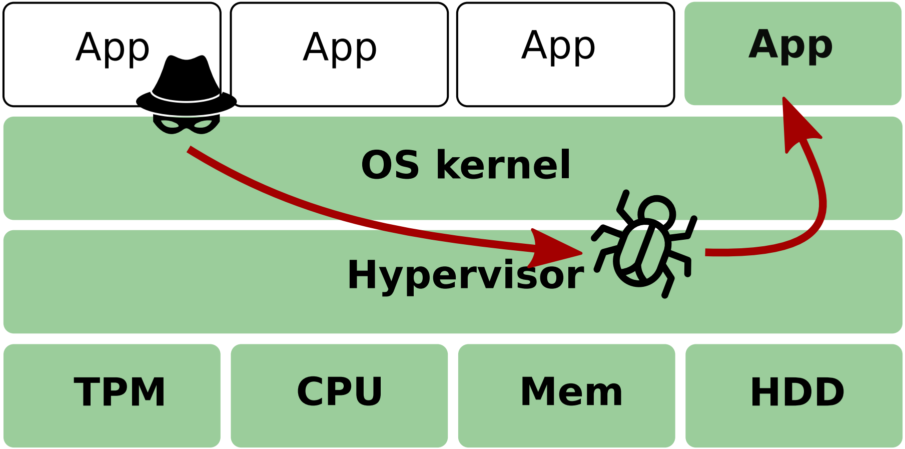
Side-channel attacks

Sancus TEE processor



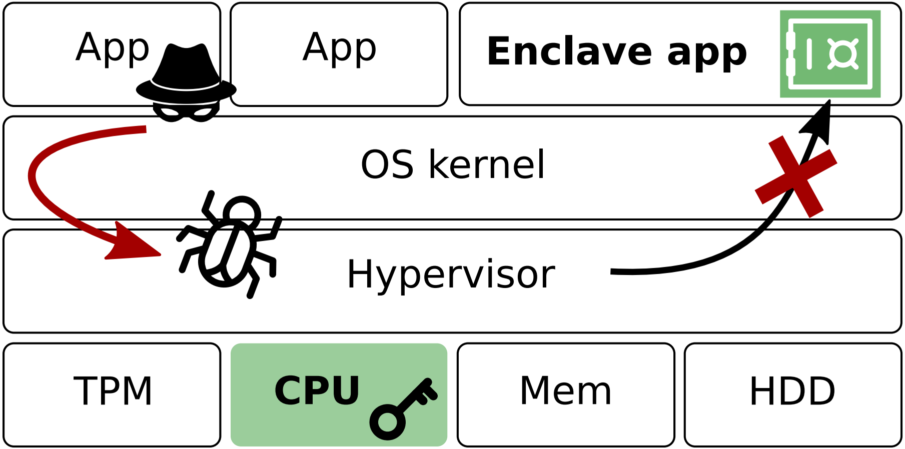
https://informationisbeautiful.net/visualizations/million-lines-of-code/

The big picture: Reducing attack surface with enclaves



Traditional **layered designs:** Large trusted computing base

The big picture: Reducing attack surface with enclaves



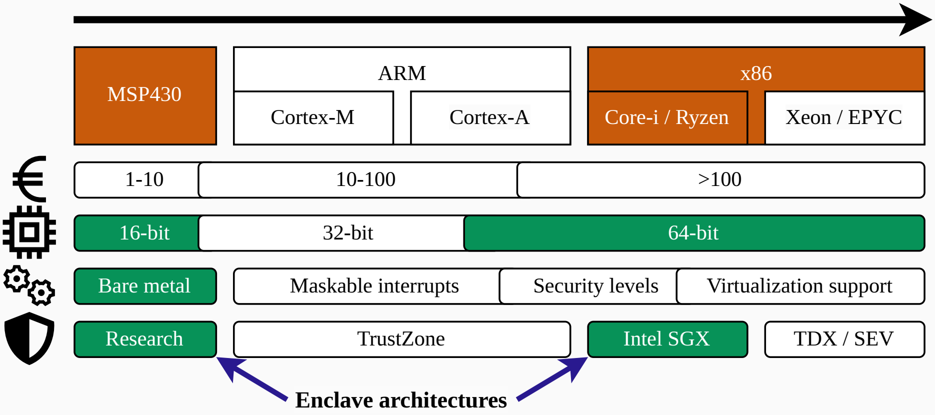
Intel SGX promise: Hardware-level **isolation and attestation**

The rise of trusted execution environments

- 2004: ARM TrustZone
- 2015: Intel Software Guard Extensions (SGX)
- 2016: AMD Secure Encrypted Virtualization (SEV)
- 2017: AMD SEV with Encrypted State (SEV-ES)
- 2018: IBM Protected Execution Facility (PEF)
- 2020: AMD SEV with Secure Nested Paging (SEV-SNP)
- 2022: Intel Trust Domain Extensions (TDX)

🔔 **TEEs are here to stay. . .**

Computing spectrum



MSP430

ARM

Cortex-M

Cortex-A

x86

Core-i / Ryzen

Xeon / EPYC

€ 1-10 | 10-100 | >100

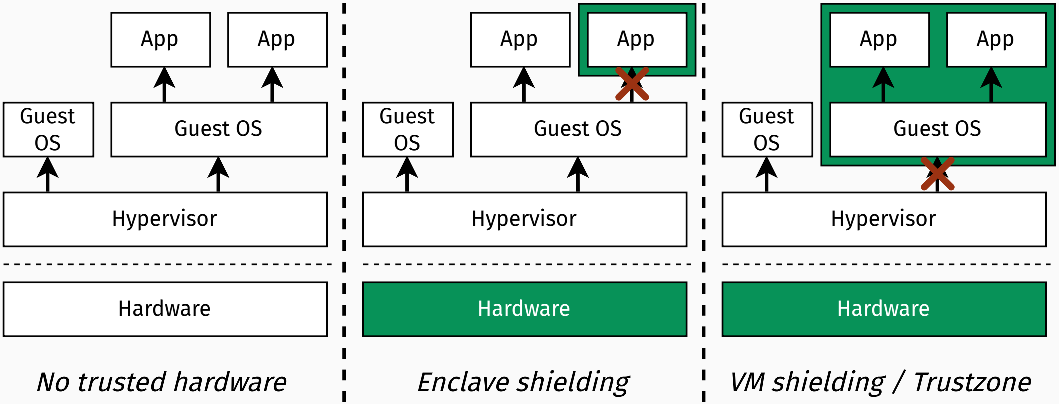
16-bit | 32-bit | 64-bit

Bare metal | Maskable interrupts | Security levels | Virtualization support

Research | TrustZone | Intel SGX | TDX / SEV

Enclave architectures

Trusted execution environment types



No trusted hardware

Enclave shielding

VM shielding / Trustzone

Highlight #1: Impact on Attacks

Vulnerable platforms: Intel Software Guard Extensions (SGX)

intel

Microsoft Azure

Signal

Enarx (Red Hat)

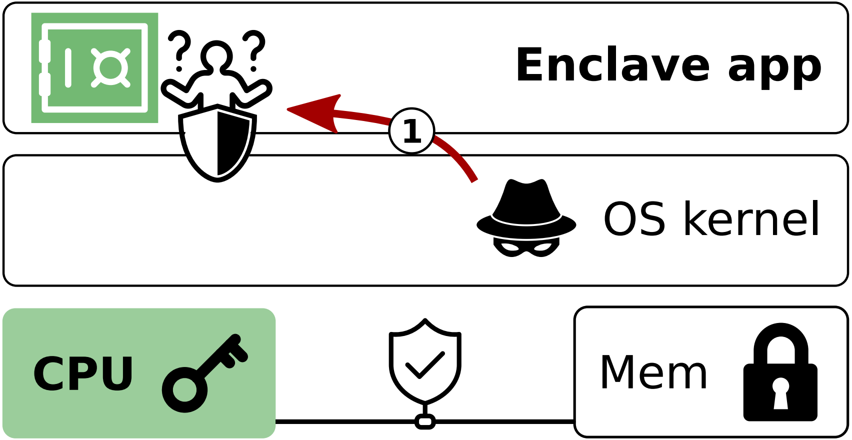
Asylo (Google)

wolfSSL

Fortanix®

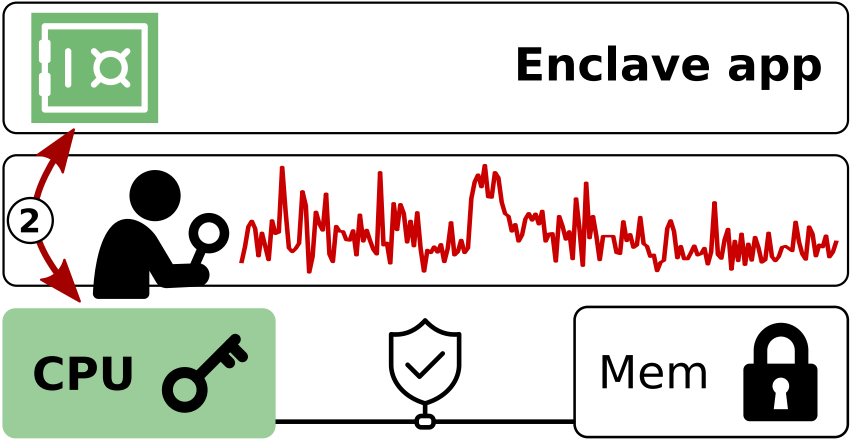
DELL

Software interface attacks (part 1)



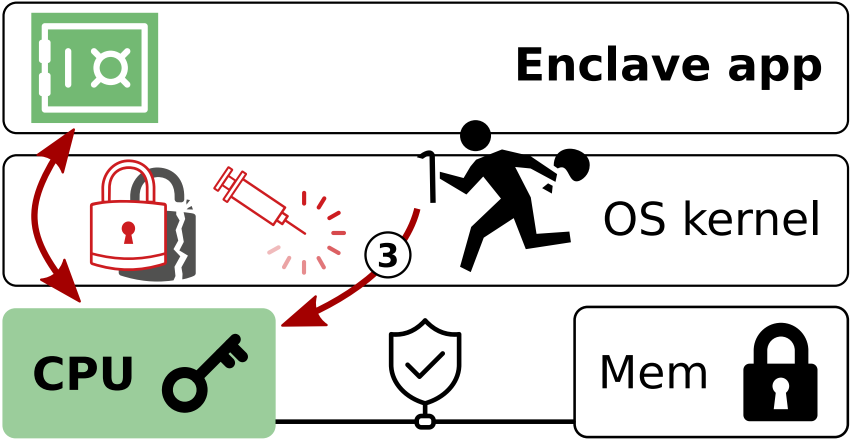
SGX not immune to **interface sanitization** oversights in enclave software

Privileged side-channel attacks (part 2)



Privileged side channels to spy on enclave-CPU interaction metadata

Transient-execution attacks (part 3)



Transient-execution data extraction from CPU to break enclave confidentiality

Research agenda: Understanding privileged attack surface

1. **Which** novel privileged attacks exist?

   → Uncover previously unknown attack avenues

2. **How** well can they be exploited in practice?

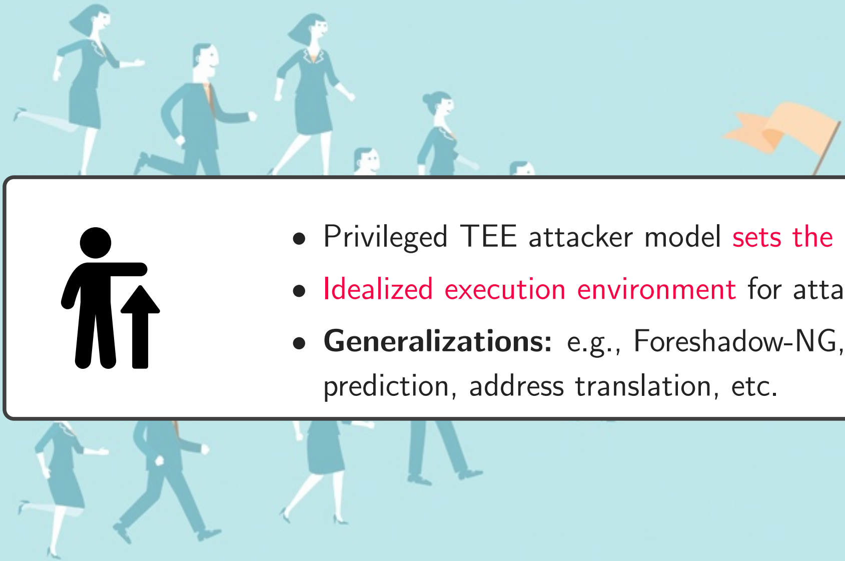
   → Develop new techniques and practical attack frameworks

3. **What** can be leaked?

   → Leak metadata and data

TEE attack research leads the way . . .

TEE attack research leads the way . . .



- Privileged TEE attacker model sets the bar!
- Idealized execution environment for attack research
- **Generalizations:** e.g., Foreshadow-NG, branch prediction, address translation, etc.

Challenge: Side-channel Sampling Rate

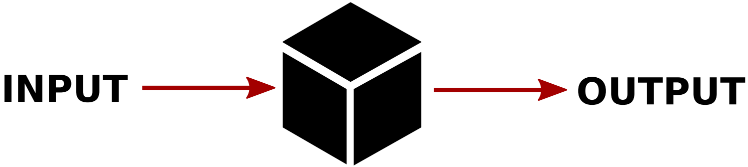
Slow shutter speed

Medium shutter speed

Fast shutter speed

CC-BY-SA Nevit Dilmen

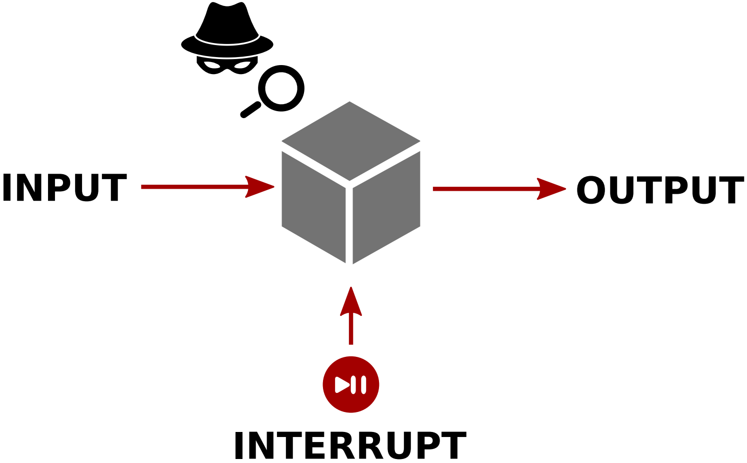
SGX-Step: Executing enclaves one instruction at a time



INPUT

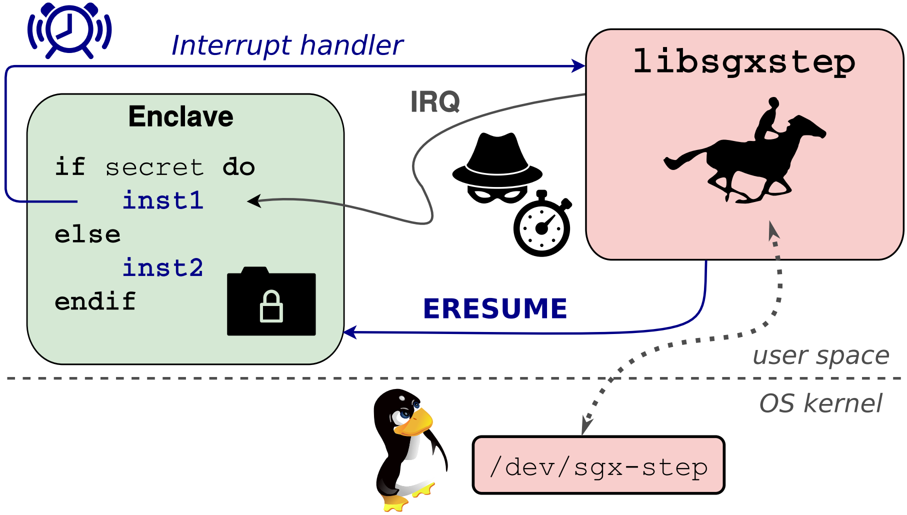
OUTPUT

SGX-Step: Executing Enclaves one Instruction at a Time



🗋 Van Bulck et al., “SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control”, SysTEX 2017.

SGX-Step: Executing Enclaves one Instruction at a Time



🗋 Van Bulck et al., "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control", SysTEX 2017.

A Retrospective of 5 Years of SGX-Step Development

SGX-Step

https://github.com/jovanbulck/sgx-step

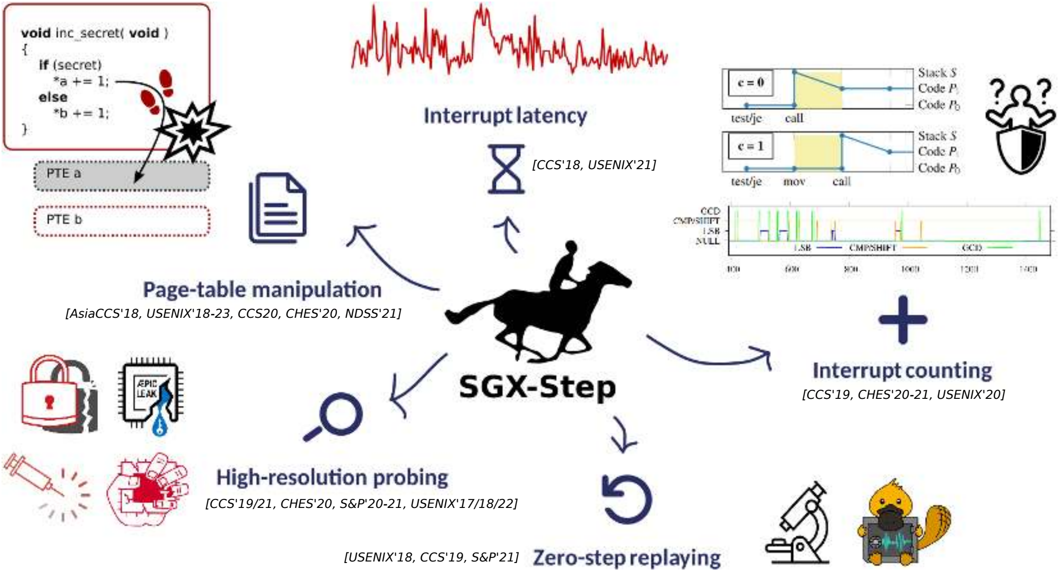
- Became **de-facto standard** for interrupt-driven attacks
- Actively maintained & supported
- Widely recognized:
  - > 400 GitHub stars
  - > 215 academic citations
- Marked influence on both **attacks & defenses** on SGX and beyond

SGX-Step: Enabling a New Line of High-Resolution Attacks

Yr	Venue	Paper	Step	Use Case	Drv
'15	S&P	Ctrl channel [XCP15]	~ Page	*Probe* (page fault)	✓ ⊞
'16	ESORICS	AsyncShock [WKPK16]	~ Page	*Exploit* (mem safety)	– 🐧
'17	CHES	CacheZoom [MIE17]	✗ >1	*Probe* (L1 cache)	✓ 🐧
'17	ATC	Hahnel et al. [HCP17]	✗ 0 - >1	*Probe* (L1 cache)	✓ ⊞
'17	USENIX	BranchShadow [LSG+17]	✗ 5 - 50	*Probe* (BPU)	✗ 🐧
'17	USENIX	Stealthy PTE [VBWK+17]	~ Page	*Probe* (page table)	✓ 🐧
'17	USENIX	DarkROP [LJJ+17]	~ Page	*Exploit* (mem safety)	✓ 🐧
'17	SysTEX	SGX-Step [VBPS17]	✓ 0 - 1	*Framework*	✓ 🏇
'18	ESSoS	Off-limits [GVBPS18]	✓ 0 - 1	*Probe* (segmentation)	✓ 🏇
'18	AsiaCCS	Single-trace RSA [WSB18]	~ Page	*Probe* (page fault)	✓ 🏇
'18	USENIX	Foreshadow [VBMW+18]	✓ 0 - 1	*Probe* (transient exec)	✓ 🏇
'18	EuroS&P	SgxPectre [CCX+19]	~ Page	*Exploit* (transient)	✓ 🐧
'18	CHES	CacheQuote [DDME+18]	✗ >1	*Probe* (L1 cache)	✓ 🐧
'18	ICCD	SGXlinger [HZDL18]	✗ >1	*Probe* (IRQ latency)	✗ 🐧
'18	CCS	Nemesis [VBPS18]	✓ 1	*Probe* (IRQ latency)	✓ 🏇
'19	USENIX	Spoiler [IMB+19]	✓ 1	*Probe* (IRQ latency)	✓ 🏇
'19	CCS	ZombieLoad [SLM+19]	✓ 0 - 1	*Probe* (transient exec)	✓ 🏇
'19	CCS	Fallout [CGG+19]	–	*Probe* (transient exec)	✓ 🏇
'19	CCS	Tale of 2 worlds [VBOM+19]	✓ 1	*Exploit* (mem safety)	✓ 🏇
'19	ISCA	MicroScope [SYG+19]	~ 0 - Page	*Framework*	✗ 🐧
'20	CHES	Bluethunder [HMW+20]	✓ 1	*Probe* (BPU)	✓ 🏇
'20	USENIX	Big troubles [WSBS19]	~ Page	*Probe* (page fault)	✓ 🏇
'20	S&P	Plundervolt [MOG+20]	–	*Exploit* (undervolt)	✓ 🏇
'20	CHES	Viral primitive [AB20]	✓ 1	*Probe* (IRQ count)	✓ 🏇
'20	USENIX	CopyCat [MVBH+20]	✓ 1	*Probe* (IRQ count)	✓ 🏇
'20	S&P	LVI [VBMS+20]	✓ 1	*Exploit* (transient)	✓ 🏇

Yr	Venue	Paper	Step	Use Case	Drv
'20	CHES	A to Z [AGB20]	~ Page	*Probe* (page fault)	✓ 🏇
'20	CCS	Déjà Vu NSS [uHGDL+20]	~ Page	*Probe* (page fault)	✓ 🏇
'20	MICRO	PTHammer [ZCL+20]	–	*Probe* (page walk)	✓ 🏇
'21	USENIX	Frontal [PSHC21]	✓ 1	*Probe* (IRQ latency)	✓ 🏇
'21	S&P	CrossTalk [RMR+21]	✓ 1	*Probe* (transient exec)	✓ 🏇
'21	CHES	Online template [AB21]	✓ 1	*Probe* (IRQ count)	✓ 🏇
'21	NDSS	SpeechMiner [XZT20]	–	*Framework*	✓ 🏇
'21	S&P	Platypus [LKO+21]	✓ 0 - 1	*Probe* (voltage)	✓ 🏇
'21	DIMVA	Aion [HXCL21]	✓ 1	*Probe* (cache)	✓ 🏇
'21	CCS	SmashEx [CYS+21]	✓ 1	*Exploit* (mem safety)	✓ 🏇
'21	CCS	Util::Lookup [SBWE21]	✓ 1	*Probe* (L3 cache)	✓ 🏇
'22	USENIX	Rapid prototyping [ESSG22]	✓ 1	*Framework*	✓ 🏇
'22	CT-RSA	Kalyna expansion [CGYZ22]	✓ 1	*Probe* (L3 cache)	✓ 🏇
'22	SEED	Enclyzer [ZXTZ22]	–	*Framework*	✓ 🏇
'22	NordSec	Self-monitoring [LBA22]	~ Page	*Defense* (detect)	✓ 🏇
'22	AutoSec	Robotic vehicles [LS22]	✓ 1 - >1	*Exploit* (timestamp)	✓ 🏇
'22	ACSAC	MoLE [LWM+22]	✓ 1	*Defense* (randomize)	✓ 🏇
'22	USENIX	AEPIC [BKS+22]	✓ 1	*Probe* (I/O device)	✓ 🏇
'22	arXiv	Confidential code [PSL+22]	✓ 1	*Probe* (IRQ latency)	✓ 🏇
'23	ComSec	FaultMorse [HZL+23]	~ Page	*Probe* (page fault)	✓ 🏇
'23	CHES	HQC timing [HSC+23]	✓ 1	*Probe* (L3 cache)	✓ 🏇
'23	ISCA	Belong to us [YJF23]	✓ 1	*Probe* (BPU)	✓ 🏇
'23	USENIX	BunnyHop [ZTO+23]	✓ 1	*Probe* (BPU)	✓ 🏇
'23	USENIX	DownFall [Mog23]	✓ 0 - 1	*Probe* (transient exec)	✓ 🏇
'23	USENIX	AEX-Notify [CVBC+23]	✓ 1	*Defense* (prefetch)	✓ 🏇

A Versatile Open-Source Attack Toolkit



```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

Interrupt latency

[CCS'18, USENIX'21]

Page-table manipulation

[AsiaCCS'18, USENIX'18-23, CCS20, CHES'20, NDSS'21]

SGX-Step

Interrupt counting

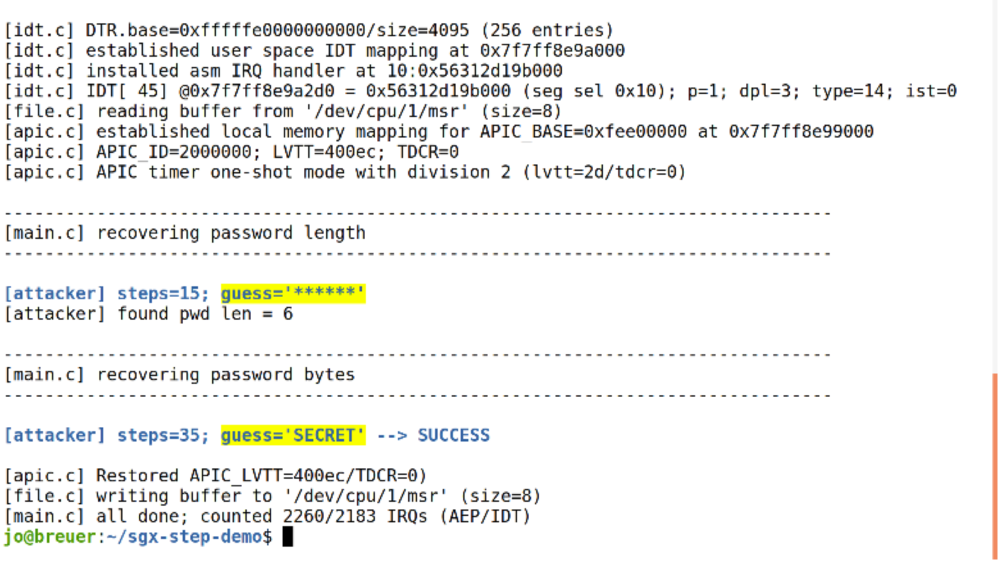
[CCS'19, CHES'20-21, USENIX'20]

High-resolution probing

[CCS'19/21, CHES'20, S&P'20-21, USENIX'17/18/22]

[USENIX'18, CCS'19, S&P'21] Zero-step replaying

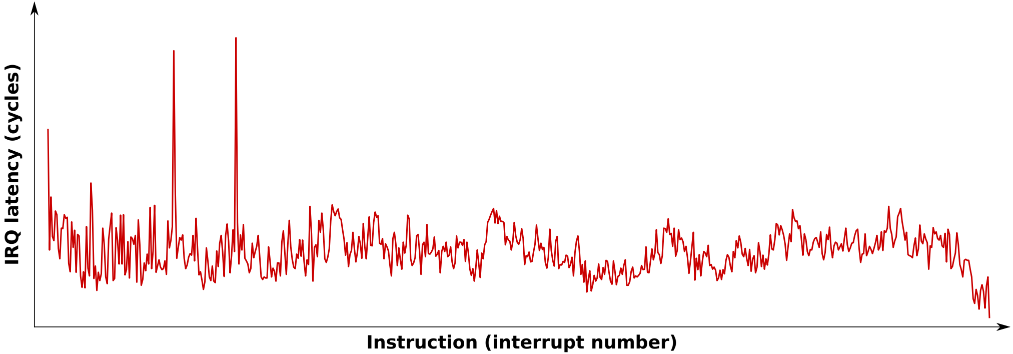
SGX-Step demo: Building a `memcmp()` Password Oracle



```
[idt.c] DTR.base=0xfffffe0000000000/size=4095 (256 entries)
[idt.c] established user space IDT mapping at 0x7f7ff8e9a000
[idt.c] installed asm IRQ handler at 10:0x56312d19b000
[idt.c] IDT[ 45] @0x7f7ff8e9a2d0 = 0x56312d19b000 (seg sel 0x10); p=1; dpl=3; type=14; ist=0
[file.c] reading buffer from '/dev/cpu/1/msr' (size=8)
[apic.c] established local memory mapping for APIC_BASE=0xfee00000 at 0x7f7ff8e99000
[apic.c] APIC_ID=2000000; LVTT=400ec; TDCR=0
[apic.c] APIC timer one-shot mode with division 2 (lvtt=2d/tdcr=0)

--------------------------------------------------------------------------------
[main.c] recovering password length
--------------------------------------------------------------------------------

[attacker] steps=15; guess='******'
[attacker] found pwd len = 6

--------------------------------------------------------------------------------
[main.c] recovering password bytes
--------------------------------------------------------------------------------

[attacker] steps=35; guess='SECRET' --> SUCCESS

[apic.c] Restored APIC_LVTT=400ec/TDCR=0)
[file.c] writing buffer to '/dev/cpu/1/msr' (size=8)
[main.c] all done; counted 2260/2183 IRQs (AEP/IDT)
jo@breuer:~/sgx-step-demo$ █
```

Nemesis: Extracting IRQ latency traces with SGX-Step

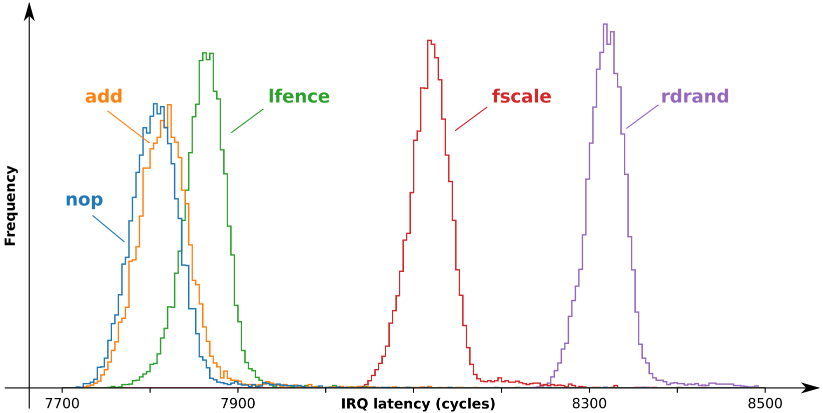
☢ **Enclave x-ray:** IRQ latency leaks instruction-level μ-arch timing!



🗋 Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018..

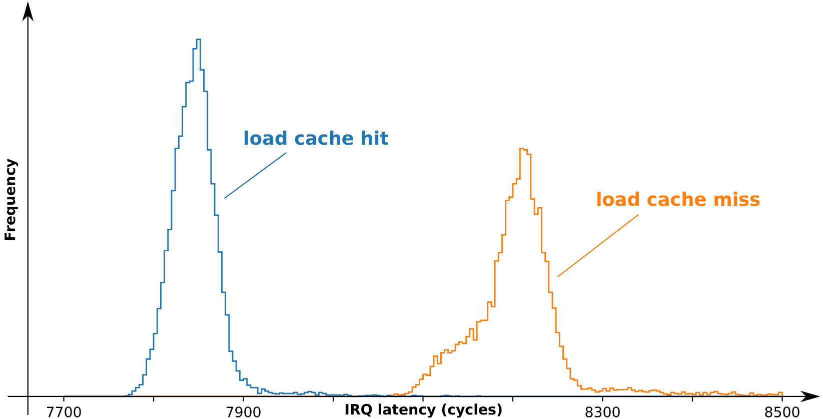
Nemesis microbenchmarks: Measuring x86 operands

Instruction timing leak: Reconstruct x86 operand class



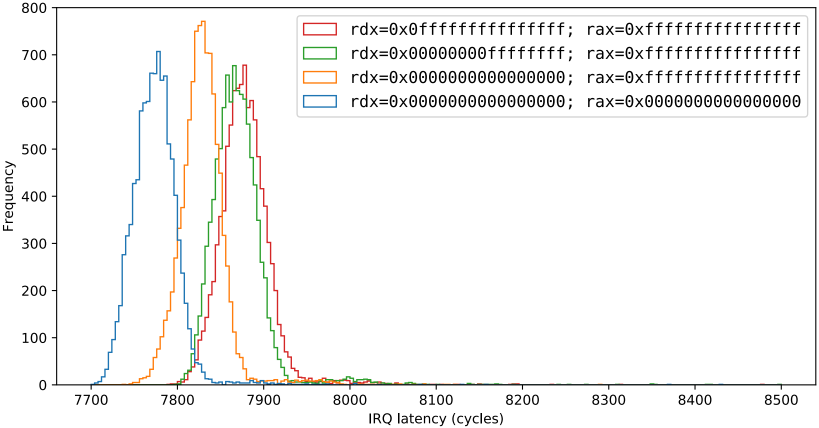
Nemesis microbenchmarks: Measuring x86 cache misses

Instruction timing leak: Reconstruct *microarchitectural state*



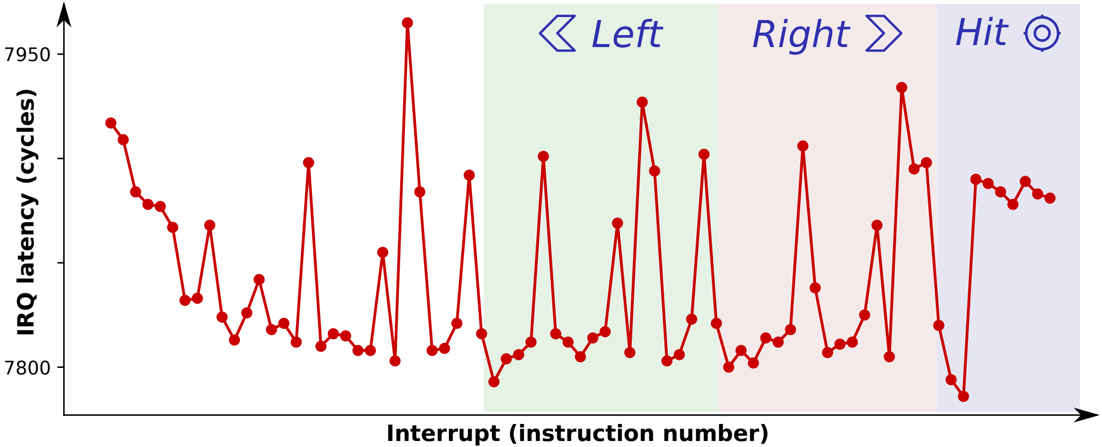
Nemesis microbenchmarks: Measuring x86 data dependencies

Instruction timing leak: Execution time ≈ dividend significant bits



De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct `bsearch` control flow



Highlight #2: Impact on Defenses

Hardening Enclaves against Single-Stepping

SGX-Step sets the **bar for adequate side-channel defenses!**

→ (e.g., LVI, compiler, static analysis, constant-time, etc.)

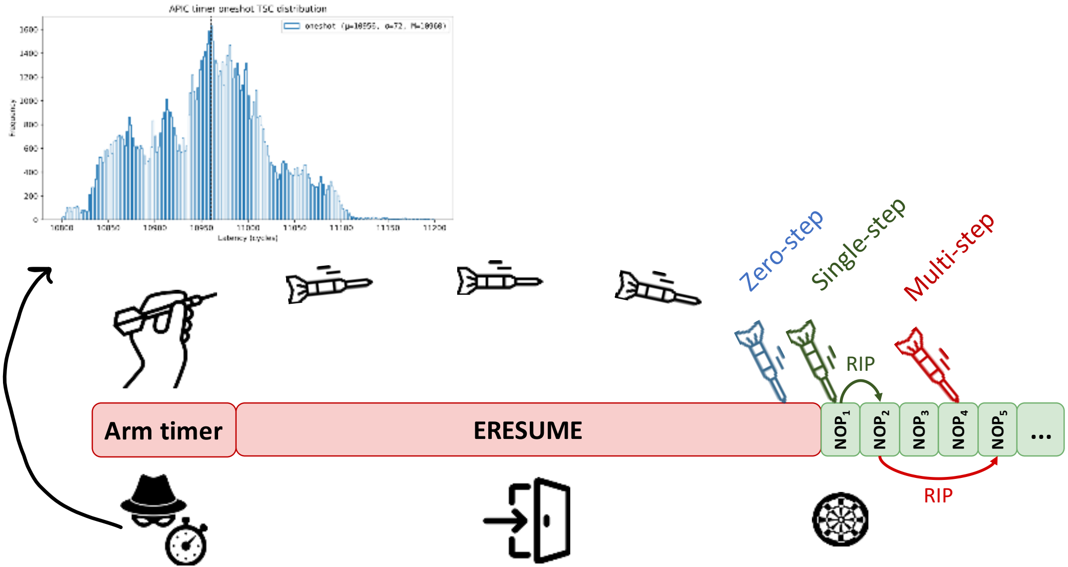
> *"ineffective if the attacker can single-step through the enclave using the recent SGX-Step framework. Taking into account these stronger attacker capabilities, we propose a new defense. . . " [HLLP18]*

SGX-Step inspired several dedicated **hardware-software mitigations**

→ Collaboration with Intel on AEX-Notify: Innovative hardware-software co-design included in recent processors

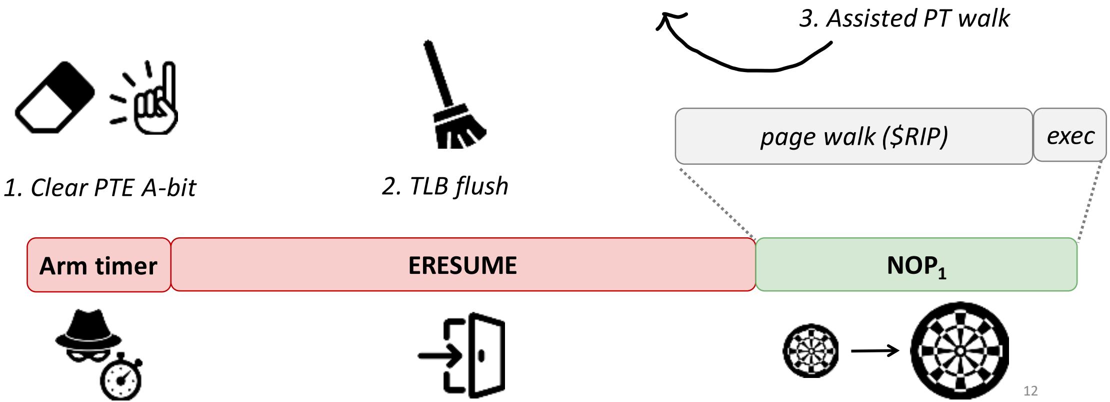
→ Probabilistic: SGX-Step remains relevant!

Root-causing SGX-Step: Aiming the timer interrupt

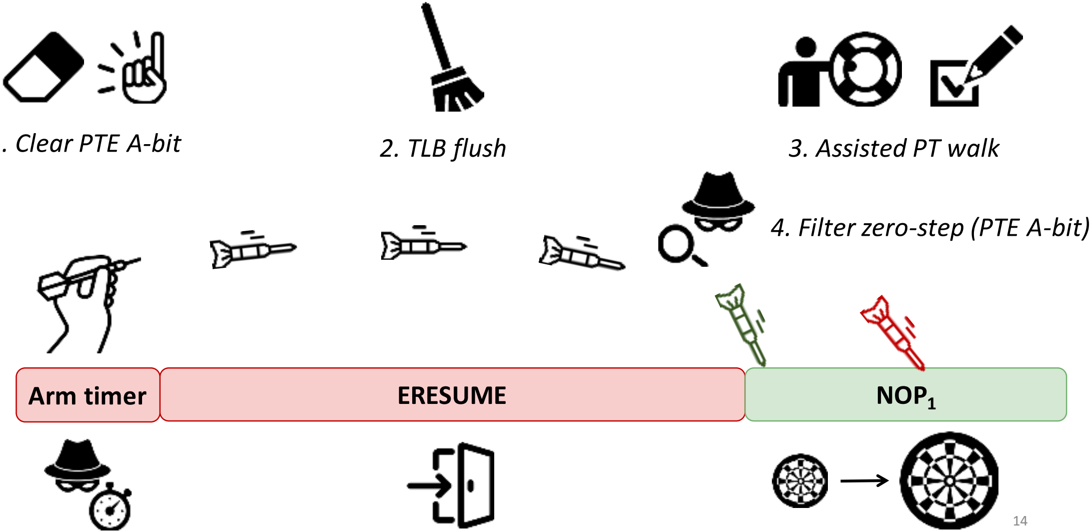


Root-causing SGX-Step: Microcode assists to the rescue!

PTE A-bit	Mean (cycles)	Stddev (cycles)
A=1	27	30
A=0	666	55



Root-causing SGX-Step: Microcode assists to the rescue!



1. Clear PTE A-bit

2. TLB flush

3. Assisted PT walk

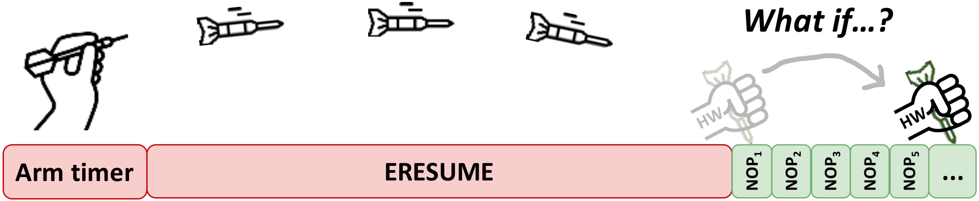
4. Filter zero-step (PTE A-bit)

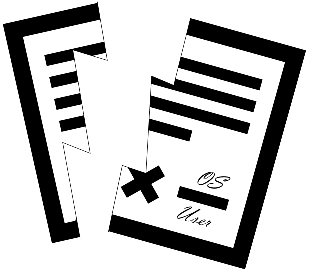
Arm timer

ERESUME

NOP$_1$

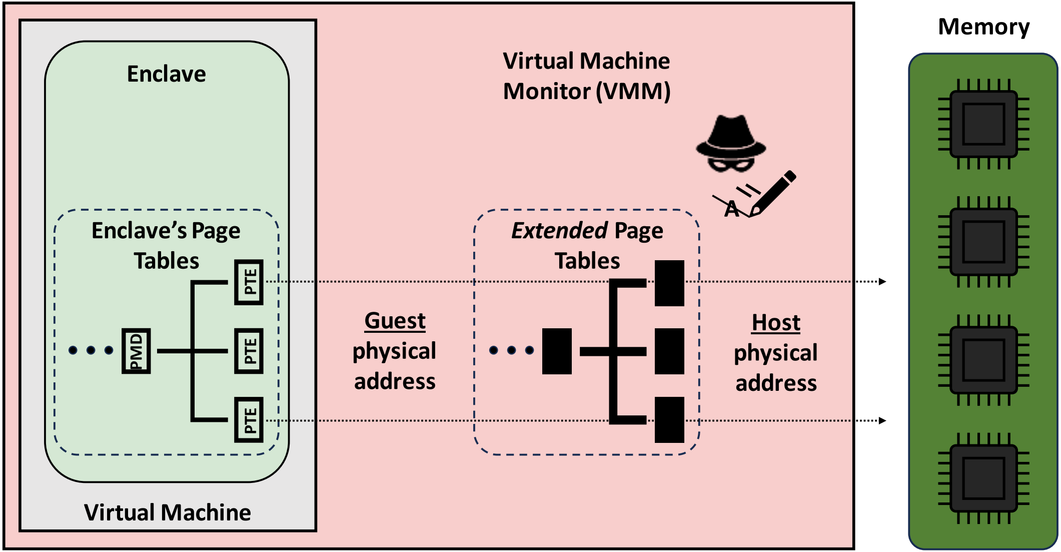
Ideas that were rejected (2)



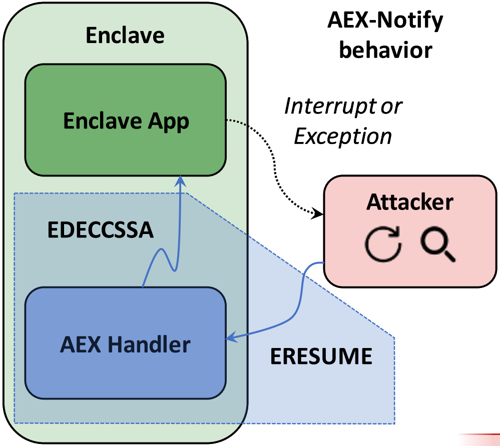


Highly complex

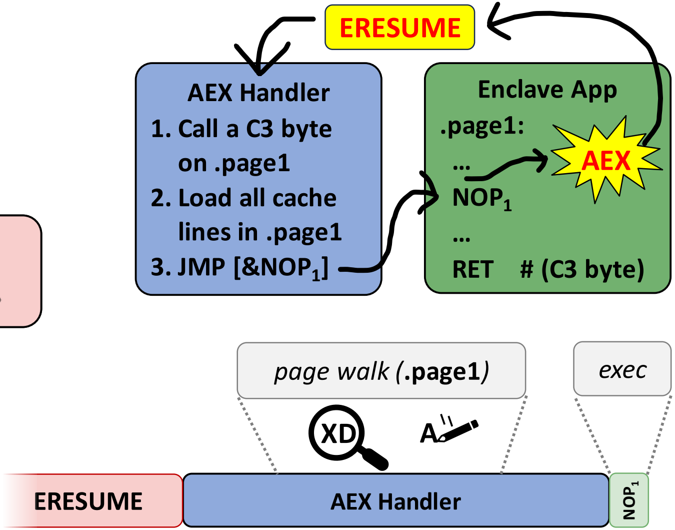
Ideas that were rejected (3)



AEX-Notify solution overview



Legend: AEX-Notify ISA Extension



AEX Handler

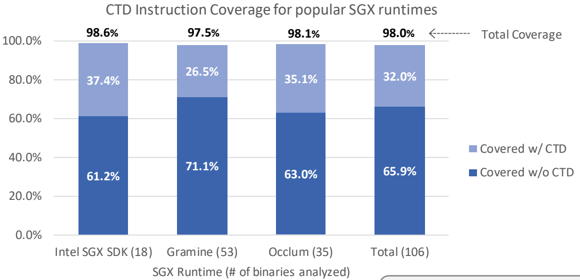
1. Call a C3 byte on .page1
2. Load all cache lines in .page1
3. JMP [&NOP_1]

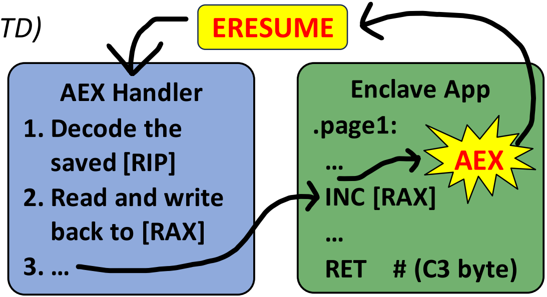
Enclave App

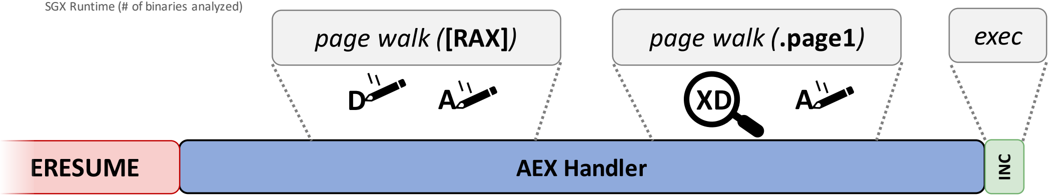
```
.page1:
  …
  NOP₁
  …
  RET    # (C3 byte)
```

AEX-Notify solution overview

We implemented a fast, constant-time decoder (CTD)







TECHNIQUES AND TECHNOLOGIES TO ADDRESS MALICIOUS SINGLE-STEPPING AND ZERO-STEPPING OF TRUSTED EXECUTION ENVIRONMENTS

TECHNICAL FIELD

[0001] The disclosure relates generally to electronics, and, more specifically, an embodiment of the disclosure relates to techniques and technologies to address malicious single-stepping and zero-stepping of trusted execution environments (TEEs).

BACKGROUND

[0002] Trusted Execution Environments (TEEs), such as Intel® Software Guard Extensions (Intel® SGX), are susceptible to methods that induce interrupts or exceptions to maliciously single-step (e.g. SGX-Step) or zero-step instruction processing in the TEE (e.g. Microscope replay attack, PLATYPUS power side-channel attack). During single-stepping or zero-stepping, a malicious hypervisor or operating system (OS) may be able to increase the granularity of side channel information which can be collected during the TEE processing. Analyzing side channel information is a method that can be used to infer information, such as instruction flows and data, about the TEE. Thus, there is value in techniques that can mitigate these attack techniques, specifically single-stepping and zero-stepping of TEEs.

side-channel attack) and then resumes execution of the code from the enclave according to embodiments of the disclosure.

[0011] FIG. **8** illustrates a method of handling an asynchronous exit of the execution of code from an enclave that utilizes an enclave enter instruction, an enclave exit instruction, and an enclave resume instruction that invokes a handler to handle an operating system signal caused by the asynchronous exit and then resumes execution of the code from the enclave according to embodiments of the disclosure.

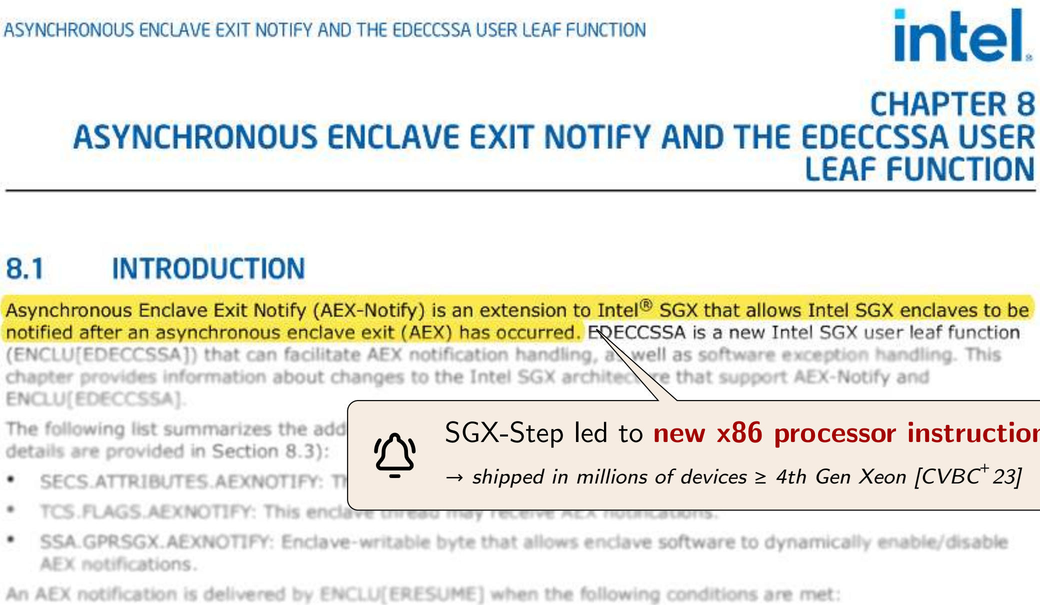
[0012] FIG. **9** illustrates a method of handling an exception with an enclave that comprises a field to indicate a set of one or more exceptions to suppress, and when execution of the code in the enclave encounters the exception, a handler is invoked without delivering the exception to an operating system according to embodiments of the disclosure.

[0013] FIG. **10** illustrates a hardware processor coupled to storage that includes one or more enclave instructions (e.g., an enclave resume (ERESUME) instruction) according to embodiments of the disclosure.

[0014] FIG. **11** is a flow diagram illustrating operations of a method for processing an "ERESUME" instruction according to embodiments of the disclosure.

[0015] FIG. **12** is a flow diagram illustrating operations of another method for processing an "ERESUME" instruction according to embodiments of the disclosure.

[0016] FIG. **13**A is a block diagram illustrating a generic



ASYNCHRONOUS ENCLAVE EXIT NOTIFY AND THE EDECCSSA USER LEAF FUNCTION

intel.

CHAPTER 8
ASYNCHRONOUS ENCLAVE EXIT NOTIFY AND THE EDECCSSA USER LEAF FUNCTION

8.1 INTRODUCTION

Asynchronous Enclave Exit Notify (AEX-Notify) is an extension to Intel® SGX that allows Intel SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. EDECCSSA is a new Intel SGX user leaf function (ENCLU[EDECCSSA]) that can facilitate AEX notification handling, as well as software exception handling. This chapter provides information about changes to the Intel SGX architecture that support AEX-Notify and ENCLU[EDECCSSA].

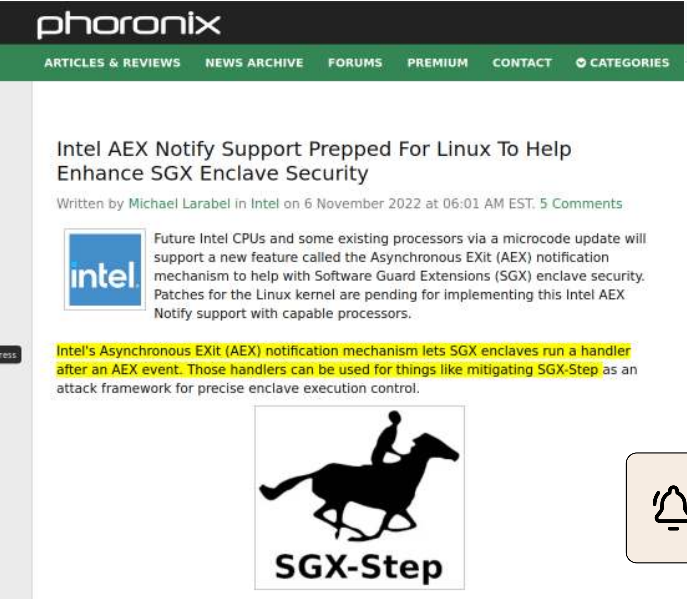
The following list summarizes the add
details are provided in Section 8.3):

- SECS.ATTRIBUTES.AEXNOTIFY: Th
- TCS.FLAGS.AEXNOTIFY: This enclave thread may receive AEX notifications.
- SSA.GPRSGX.AEXNOTIFY: Enclave-writable byte that allows enclave software to dynamically enable/disable AEX notifications.

An AEX notification is delivered by ENCLU[ERESUME] when the following conditions are met:

SGX-Step led to **new x86 processor instructions!**

→ *shipped in millions of devices ≥ 4th Gen Xeon [CVBC+23]*

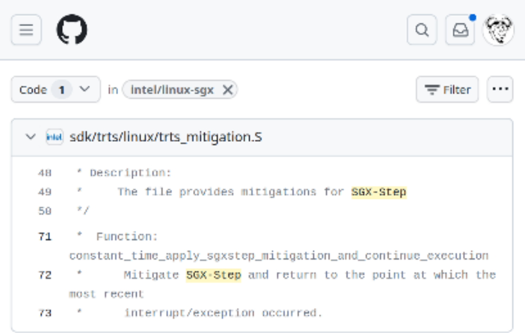


Intel AEX Notify Support Prepped For Linux To Help Enhance SGX Enclave Security

Written by Michael Larabel in Intel on 6 November 2022 at 06:01 AM EST. 5 Comments

Future Intel CPUs and some existing processors via a microcode update will support a new feature called the Asynchronous EXit (AEX) notification mechanism to help with Software Guard Extensions (SGX) enclave security. Patches for the Linux kernel are pending for implementing this Intel AEX Notify support with capable processors.

Intel's Asynchronous EXit (AEX) notification mechanism lets SGX enclaves run a handler after an AEX event. Those handlers can be used for things like mitigating SGX-Step as an attack framework for precise enclave execution control.



sdk/trts/linux/trts_mitigation.S

```
 * Description:
 *     The file provides mitigations for SGX-Step
 */
 * Function: constant_time_apply_sgxstep_mitigation_and_continue_execution
 *     Mitigate SGX-Step and return to the point at which the most recent
 *     interrupt/exception occurred.
```

SGX-Step led to **changes in major OSs and enclave SDKs**

Beyond SGX-Step: Derived Frameworks for Emerging TEEs

SGX-Step has inspired similar single-stepping frameworks for alternative TEEs

→ e.g., **AMD SEV, intel TDX, arm TrustZone**

Independent testimonies on SGX-Step's impact

- *"In the hope that the framework inspires a similar community as SGX-Step, we dubbed it SEV-Step." [WWRE23]*
- *"Leveraging SGX-Step type attack to compromise Intel TDX, which is coined as TDX-Step [. . .] Working exploit well within the timeline but also collaborated closely with the Intel TDX architecture team to review and refine the mitigation for the vulnerability." [Int23]*

“Embedded-systems security is, for lack of a better word, a mess.”

– John Viega & Hugh Thompson (S&P’12)

Sancus: Lightweight trusted computing for the IoT

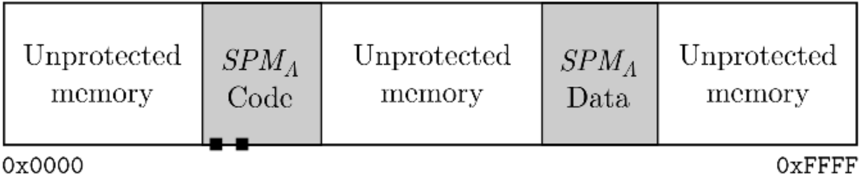
+

+

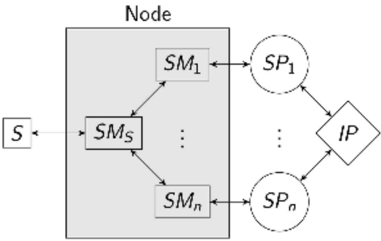
OpenMSP430 CPU extensions
for isolation + attestation

LLVM compiler pass

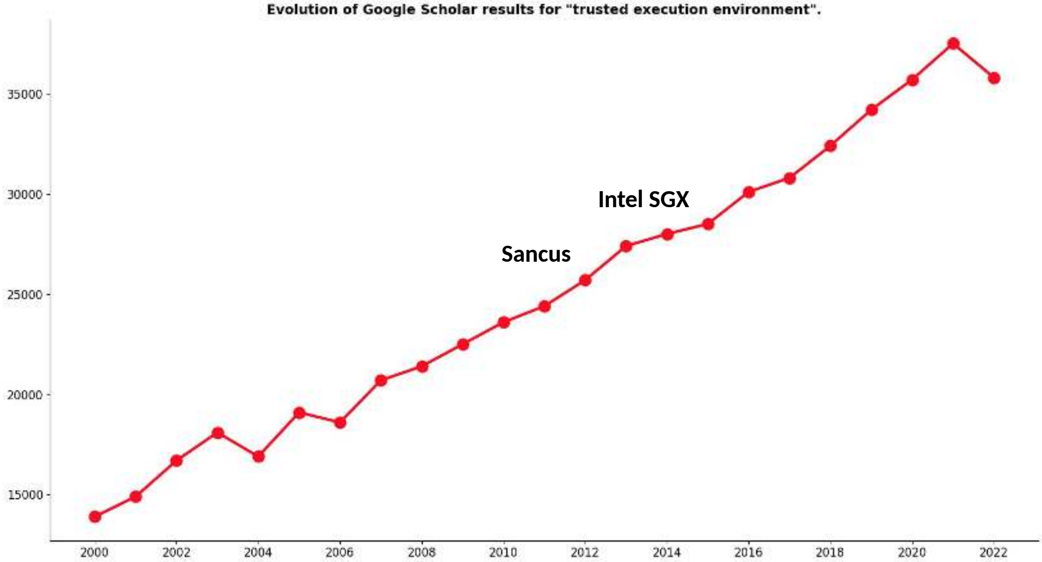
Support software
“operating system”

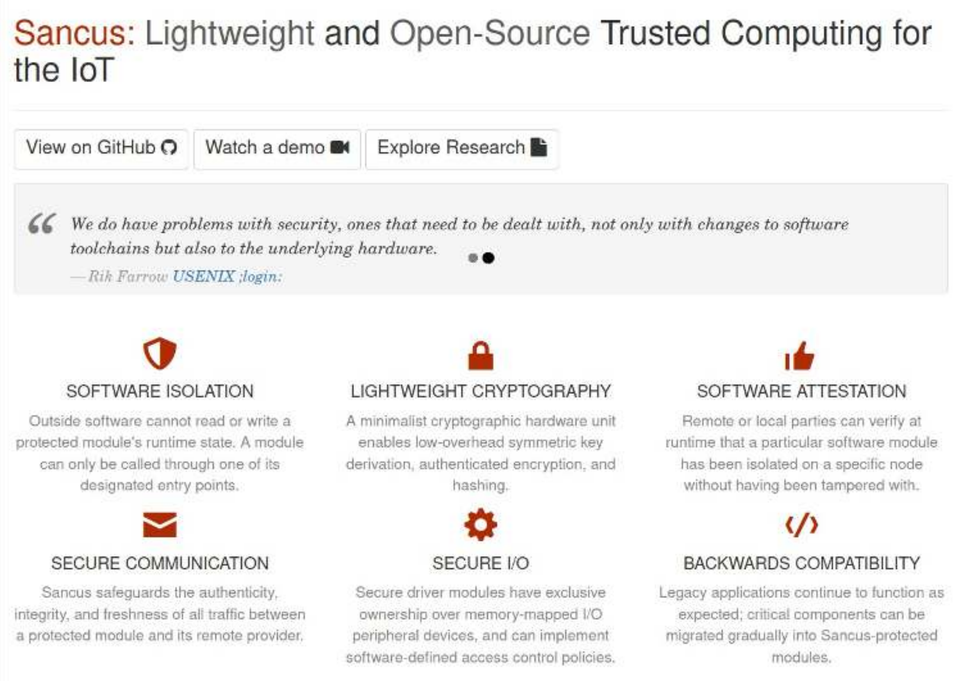


VERIFIED
DistriNet
VERIFIED


Node
SM_1
SP_1
S
SM_S
⋮
⋮
IP
SM_n
SP_n

The bigger picture: The rise of trusted execution





Sancus: Lightweight and Open-Source Trusted Computing for the IoT

> *We do have problems with security, ones that need to be dealt with, not only with changes to software toolchains but also to the underlying hardware.*
>
> *— Rik Farrow USENIX ;login:*

SOFTWARE ISOLATION

Outside software cannot read or write a protected module's runtime state. A module can only be called through one of its designated entry points.

LIGHTWEIGHT CRYPTOGRAPHY

A minimalist cryptographic hardware unit enables low-overhead symmetric key derivation, authenticated encryption, and hashing.

SOFTWARE ATTESTATION

Remote or local parties can verify at runtime that a particular software module has been isolated on a specific node without having been tampered with.

SECURE COMMUNICATION

Sancus safeguards the authenticity, integrity, and freshness of all traffic between a protected module and its remote provider.

SECURE I/O

Secure driver modules have exclusive ownership over memory-mapped I/O peripheral devices, and can implement software-defined access control policies.

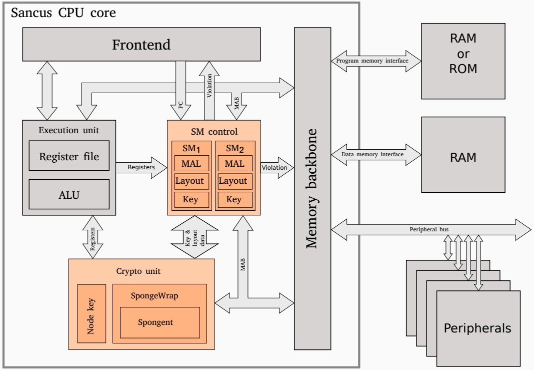
BACKWARDS COMPATIBILITY

Legacy applications continue to function as expected; critical components can be migrated gradually into Sancus-protected modules.

Sancus: A Low-Cost Security Architecture for IoT devices

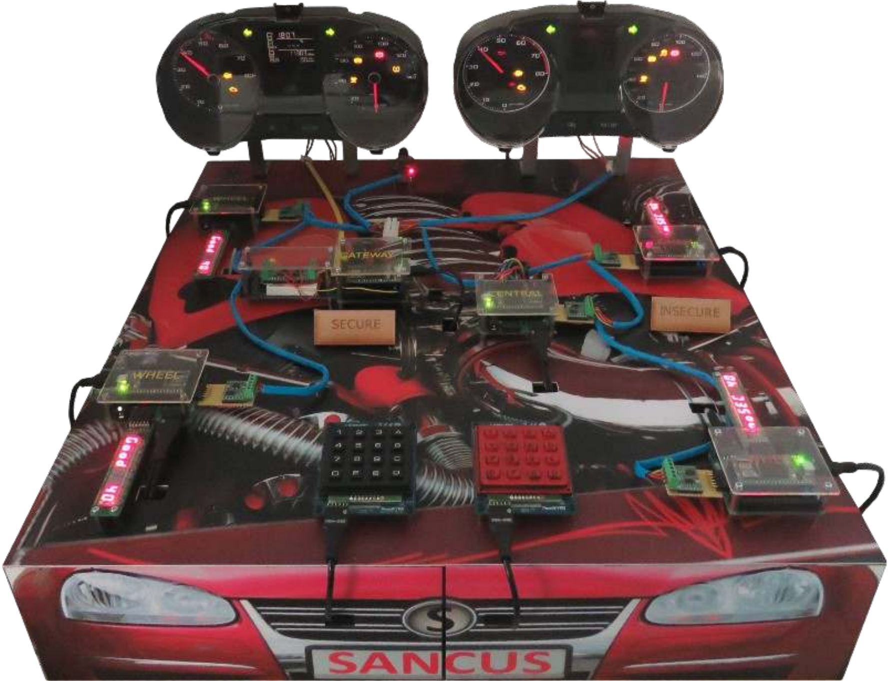
Extends openMSP430 with strong security primitives

- Software Component Isolation
- Cryptography & Attestation
- Secure I/O through isolation of MMIO ranges
- Efficient, Modular, ≤ 2 kLUTs
- Cryptographic key hierarchy for software attestation
- Isolated components are typically very small (< 1kLOC)

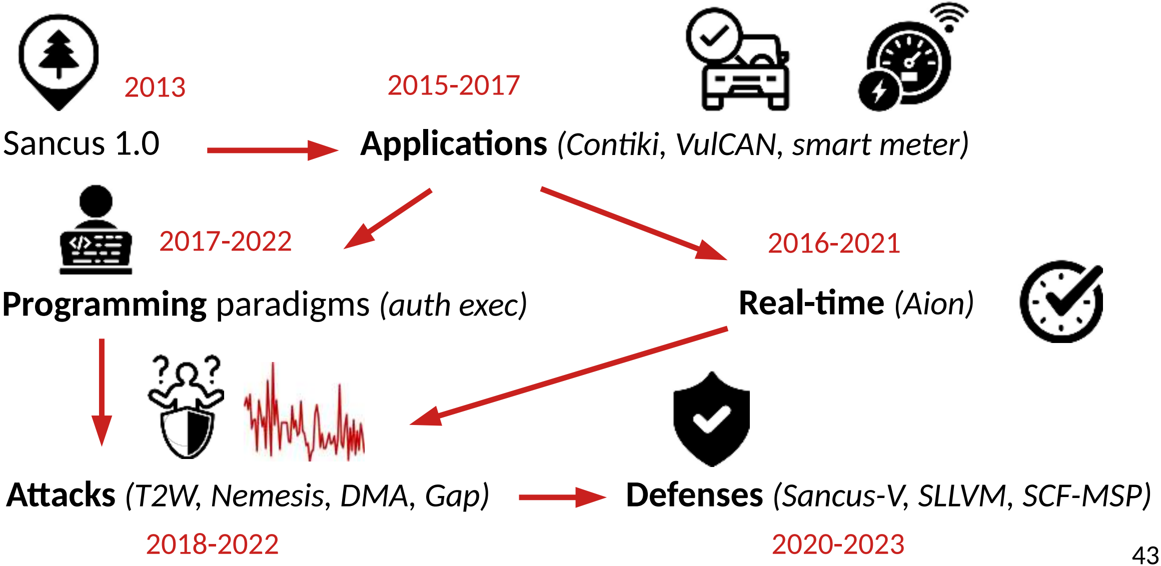


Noorman et al. Sancus 2.0: A Low-Cost Security Architecture for IoT devices. TOPS, 2017

Sancus is open source: `https://distrinet.cs.kuleuven.be/software/sancus/`


GATEWAY
CENTRAL
SECURE
INSECURE
WHEEL
Good 10
Good 40
SANCUS

Sancus is dead, long live Sancus!



2013

Sancus 1.0

2015-2017

Applications *(Contiki, VulCAN, smart meter)*

2017-2022

Programming paradigms *(auth exec)*

2016-2021

Real-time *(Aion)*

Attacks *(T2W, Nemesis, DMA, Gap)*

2018-2022

Defenses *(Sancus-V, SLLVM, SCF-MSP)*

2020-2023

Aion: Strong Availability Guarantees for Enclaves

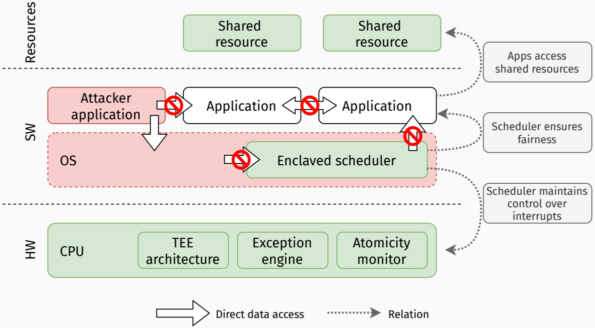
Sancus as a Starting Point

Trusted Software

- Protected Scheduler controls interrupts and scheduling decisions

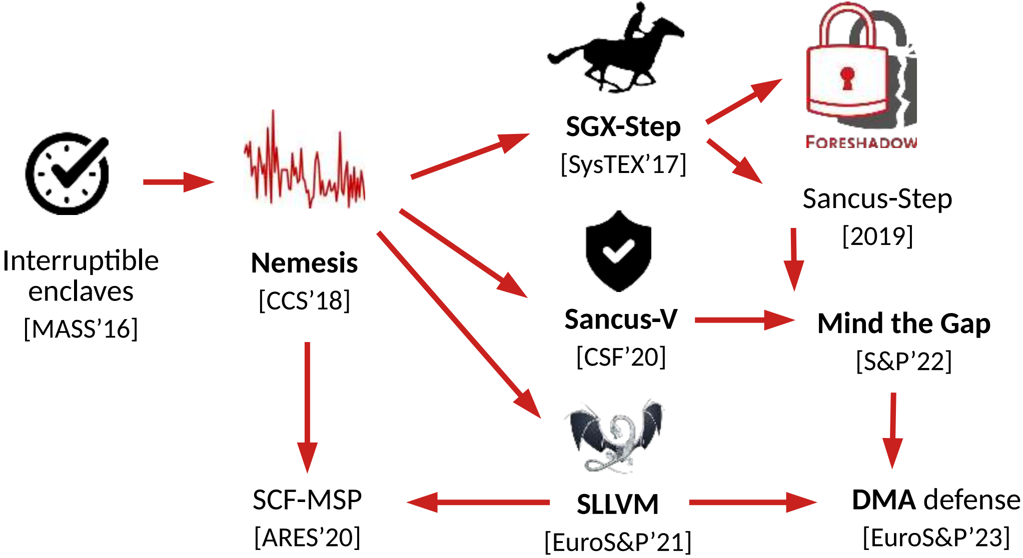
Hardware Extensions

- Exception Engine facilitates interruption of (protected) threads
- Atomicity Monitor provides control over interrupts to scheduler, guarantees bounded critical sections



Alder et al., Aion: Enabling Open Systems through Strong Availability Guarantees for Enclaves. CCS, 2021

Sancus attack research: The gift that keeps giving



Interruptible enclaves [MASS’16]

Nemesis [CCS’18]

SGX-Step [SysTEX’17]

Sancus-Step [2019]

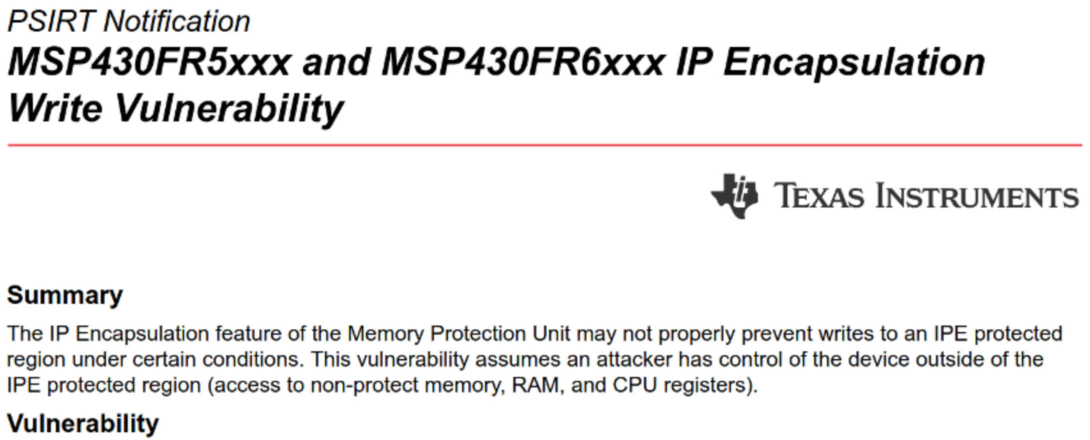
Sancus-V [CSF’20]

Mind the Gap [S&P’22]

SCF-MSP [ARES’20]

SLLVM [EuroS&P’21]

DMA defense [EuroS&P’23]



PSIRT Notification

MSP430FR5xxx and MSP430FR6xxx IP Encapsulation Write Vulnerability

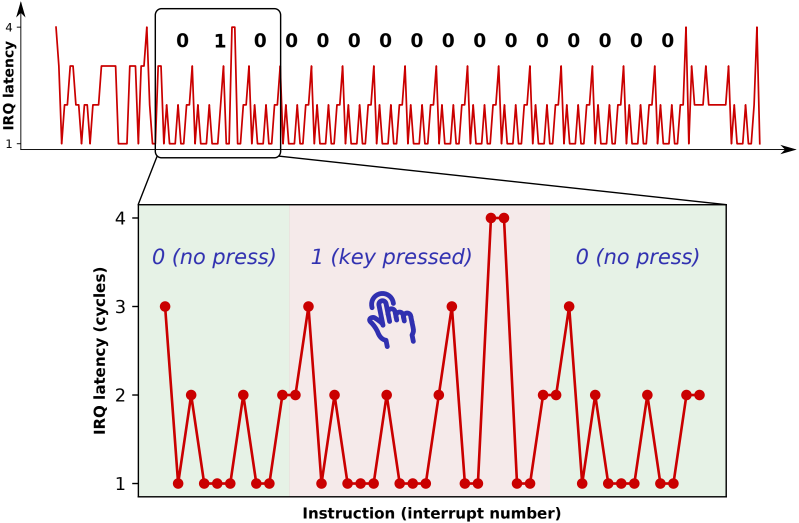
TEXAS INSTRUMENTS

Summary

The IP Encapsulation feature of the Memory Protection Unit may not properly prevent writes to an IPE protected region under certain conditions. This vulnerability assumes an attacker has control of the device outside of the IPE protected region (access to non-protect memory, RAM, and CPU registers).

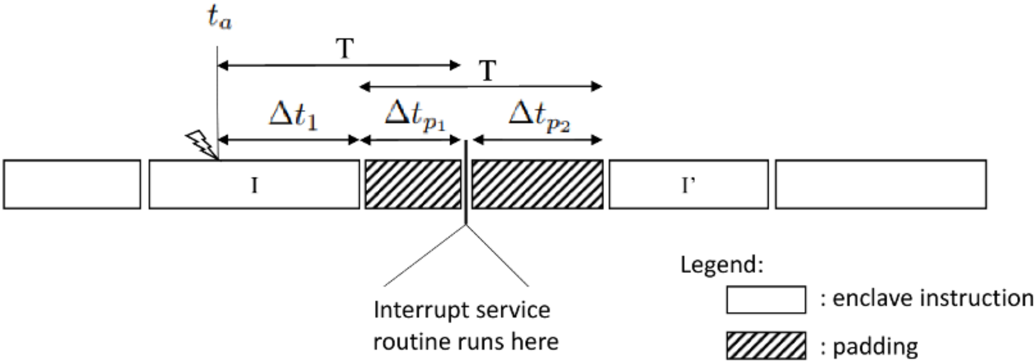
Vulnerability

Nemesis on embedded microprocessors (openMSP430+Sancus)



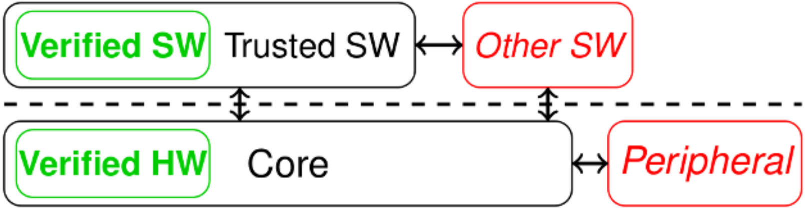
https://github.com/sancus-tee

Nemesis hardware defense: Padding interrupt latency



- 🗋 Busi et al. “Provably Secure Isolation for Interruptible Enclaved Execution on Small Microprocessors”, CSF 2020.

Nemesis hardware defense: Studying limitations of formal systems



Verified SW

Trusted SW

Other SW

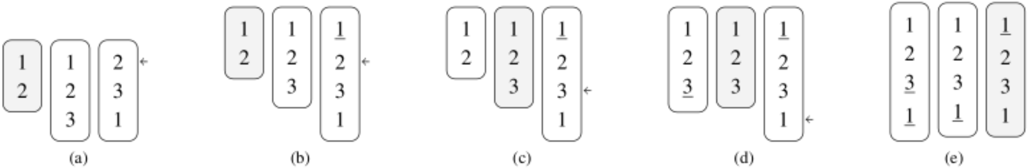
Verified HW

Core

Peripheral

- 🗋 Busi et al. “Provably Secure Isolation for Interruptible Enclaved Execution on Small Microprocessors”, CSF 2020.
- 🗋 Bognar et al. “Mind the Gap: Studying the Insecurity of Provably Secure Embedded Trusted Execution Architectures”, S&P 2022.

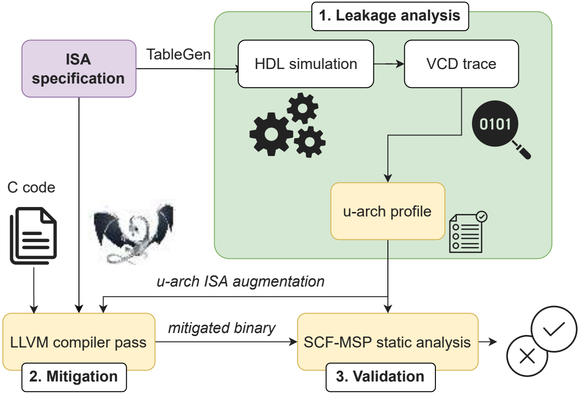
Nemesis software defenses: Balancing vulnerable branches



(a) (b) (c) (d) (e)

- 🗋 Winderix et al. “Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks”, EuroS&P 2021.
- 🗋 Pouyanrad et al. “SCFMSP: Static Detection of Side Channels in MSP430 Programs”, ARES 2020.
- 🗋 Salehi et al. “NemesisGuard: Mitigating Interrupt Latency Side Channel Attacks with Static Binary Rewriting”, Computer Networks 2022.

Nemesis software defenses: Principled ISA augmentation



- 🗋 Winderix et al. "Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks", EuroS&P 2021.
- 🗋 Pouyanrad et al. "SCFMSP: Static Detection of Side Channels in MSP430 Programs", ARES 2020.
- 🗋 Salehi et al. "NemesisGuard: Mitigating Interrupt Latency Side Channel Attacks with Static Binary Rewriting", Computer Networks 2022.
- 🗋 Bognar et al. "MicroProfiler: Principled Side-Channel Mitigation through Microarchitectural Profiling", EuroS&P 2023.

Outlook: Future and ongoing research directions

1. **Universal attack primitives:** Intel TDX, AMD SEV, ARM?

   → Adversary capabilities, hardware vs. software monitor, automation, etc.

2. **Hardware extensions** for next-gen TEEs: MSP430-Sancus, RISC-V

   → Provable security & limitations, availability, SMAP-like restrictions, etc.

3. **Transparent shielding:** Enclave runtime, compiler

   → Fuzzing, formal verification of the enclave interface

   → Compile-time hardening for *incremental* side-channel resistance

4. Towards **transient safety:** Redefining the hardware-software contract

   → Efficient containment of Spectre (long term) vs. LVI (short term)