Hardware-assisted Security: So Close yet So Far

Ahmad-Reza Sadeghi, Ferdinand Brasser
Technische Universität Darmstadt &
Intel Collaborative Research Institute for Secure Computing
Collaborators

- N. Asokan, Aalto University, Finland
- Luca Davi, Christopher Liebchen, TU Darmstadt, Germany
- Per Larsen, Steven Crane, Andrei Homescu, UCI, USA
- Gene Tsudik, Michael Franz, UCI, USA
- Thorsten Holz, Bochum University, Germany
- Yier Jin, Dean Sullivan, Orlando Arias, UCF, USA
- Patrick Koeberl, Matthias Schunter, Intel Labs
- ARM, Gieseke & Devrient, IBM, Huawei, NXP
**Conclusion**

- Run-time attacks
- Side-channels vulnerable
- Limited access by 3rd party developers
- Afterthought, not scalable

---

**Trusted/Trustworthy Computing**

**PROTECT ALL THE NATIVE CODES**

- Fantastic
- Sad
- Very Sad
- Complicated?
- Total Disaster
Why Hardware-assisted Security?

Software Stack

Operating System

Hardware

Peripheral

CPU

I/O

Memory
Goal: Self-Contained Security

- Establishing Trusted Execution Environment

Software Stack

- App 1
- App 2
- App 3
- App 4

Operating System

Hardware

- Peripherals
- CPU
- Memory
- I/O

• Platform boot integrity
• Secure storage
• Device identification
• Isolated execution
• Device authentication capabilities
Historical Overview

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cambridge CAP</td>
<td>VAX/VMS</td>
<td>Trusted Platform Module (TPM)</td>
<td>PUFs</td>
<td>GP TEE standards</td>
</tr>
<tr>
<td>Simple smart cards</td>
<td>Java security architecture</td>
<td>TI M-Shield</td>
<td>Late launch/TXT</td>
<td>TPM 2.0</td>
</tr>
<tr>
<td>Protection rings</td>
<td></td>
<td></td>
<td>ARM TrustZone</td>
<td>Intel SGX</td>
</tr>
<tr>
<td>Reference monitor</td>
<td>Hardware-assisted secure boot</td>
<td>Mobile hardware security architectures</td>
<td>On-board Credentials</td>
<td>Mobile OS security architectures</td>
</tr>
<tr>
<td>Java Card platform</td>
<td></td>
<td></td>
<td></td>
<td>Mobile Trusted Module (MTM)</td>
</tr>
</tbody>
</table>
Dedicated Security Devices (Smartcards)

Software Stack
- App 1
- App 2
- App 3
- App 4

Operating System

Hardware
- Peripherals
- CPU
- Memory
- I/O

Key icon on I/O
Integrated Security Devices

IBM Integrity Measurement Architecture (IMA)

[Sadeghi et al, ACMSTC 2006]
TPM-Based Trusted Computing

- Remote (binary) attestation is static
  - Does not reflect code’s behavior
    - Property-based Attestation [Stüble et al, NSPW 2004]
  - Does not detect runtime attacks
    - Control FLow Attestation [Davi et al, CCS 2016 & DAC 2017]
ARM TrustZone

- Subsidy Lock
- IMEI Protection

iOS
- Device Encryption
- Touch ID, Apple Pay

Android
- Full-Disk Encryption (FDE)
- Samsung KNOX
  - Secure-I/O, Attestation
  - Real-time Kernel Protection (TIMA)

iOS
- Device Encryption
- Touch ID, Apple Pay

Android
- Full-Disk Encryption (FDE)
- Samsung KNOX
  - Secure-I/O, Attestation
  - Real-time Kernel Protection (TIMA)

IMSI: International Mobile Equipment Identifier

- Onboard credential [NOKIA]
- Mobicore [G&D]

Secure World

- Trustlet 1
- Trustlet 2
- Trustlet 3

Operating System

Hardware
- Peripherals
- CPU
- Memory
- I/O
Attacks on TrustZone

Reflections on trusting TrustZone [Dan Rosenberg, BlackHat US, 2014]

Attacking your Trusted Core [Di Shen, BlackHat US, 2015]

Breaking Android Full Disc Encryption [laginimaineb from Project Zero, 2016]
Breaking Android Full Disk Encryption

[laginimaineb from Project Zero, 2016]

Compromise Media Server System App
Grants Direct Access to QSEE Kernel Driver

Normal World

Android Kernel
Driver
QSEE-COM

App

Media-server

KeyMaster Trustlet
QSEE-HMAC
QSEE Kernel
Message Handler
RWX Section

QSEE: Qualcomm Secure Execution Environment

Hardware
TZ Extension

// inject .code 16
SUB R3,R3,#0x10

// get key buff
MOV R2,#0x0
Loop:
LDR R0,[R3,R2]
STR R0,[R1,R2]
ADD R2,R2,#0x4
MOV R0,#0x30
SUB R0,R2,R0
BLT loop

// ret success
MOV R0,#0
BX LR
Intel Software Guard Extensions (SGX)

- App 1
  - Enclave 1

- App 2
  - Enclave 2

- App 3
  - Enclave 3

- App 4
  - Enclave 4

Software Stack:
- Operating System
- Code-reuse Attacks

Hardware:
- Peripherals
- I/O

Side-Channel Attacks (not in SGX Adv. Model)
Runtime Attacks and Defenses

code injection
exploits
code reuse
SELECTED

1997

- ret2libc
  - Solar Designer

2001

- Advanced ret2libc
  - Nergal

2005

- Borrowed Code Chunk Exploitation
  - Krahmer

2007

- ROP on x86
  - Shacham (CCS)

2008

- ROP on SPARC
  - Buchanan et al (CCS)
- ROP on Atmel AVR
  - Francillon et al (CCS)

2009

- ROP Rootkits
  - Hund et al (USENIX)
- ROP on PowerPC
  - FX Lindner (BlackHat)
- ROP on ARM/iOS
  - Miller et al (BlackHat)
- ROP without Returns
  - Checkoway et al (CCS)

2010

- Practical ROP
  - Zovi (RSA Conference)
- Pwn2Own (iOS/IE)
  - Iozzo et al / Nils

2013

- JIT-ROP
  - Snow et al (IEEE S&P)
- Blind ROP
  - Bittau et al (IEEE S&P)
- Stitching Gadgets
  - Davi et al (USENIX)
- Out-Of-Control
  - Göktas et al (IEEE S&P)
- Flushing Attacks
  - Schuster et al (RAID)
- ROP is Dangerous
  - Carlini et al (USENIX)

2011/2012

- Real-World Exploits
Main Defenses against Code Reuse

1. Code Randomization

2. Control-Flow Integrity (CFI)
<table>
<thead>
<tr>
<th>Year</th>
<th>CFI Defense Literature</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Program Shepherding: Kiriantsy et al. (USENIX Sec.)</td>
</tr>
<tr>
<td></td>
<td>Control-Flow Integrity (CFI): Abadi et al. (CCS 2005)</td>
</tr>
<tr>
<td>2005</td>
<td>XFI: Abadi et al. (OSDI)</td>
</tr>
<tr>
<td></td>
<td>Architectural Support for CFI: Budiu et al. (ASID)</td>
</tr>
<tr>
<td>2006</td>
<td>EMET: Microsoft</td>
</tr>
<tr>
<td></td>
<td>HyperSafe: Wang et al. (IEEE S&amp;P)</td>
</tr>
<tr>
<td>2010</td>
<td>CFI and Data Sandboxing: Zeng et al. (CCS)</td>
</tr>
<tr>
<td></td>
<td>Control-Flow Locking: Bletch et al. (ACSAC)</td>
</tr>
<tr>
<td></td>
<td>ROPdefender: Davi et al. (AsiaCCS)</td>
</tr>
<tr>
<td>2011</td>
<td>Branch Regulation: Kayaalp et al. (ISCA)</td>
</tr>
<tr>
<td></td>
<td>Mobile CFI: Davi et al. (NDSS)</td>
</tr>
<tr>
<td>2012</td>
<td>Control-Flow Restrictor: Pewny et al. (ACSAC)</td>
</tr>
<tr>
<td></td>
<td>kBouncer: Pappas et al. (USENIX Sec.)</td>
</tr>
<tr>
<td></td>
<td>bin-CFI: Zhang et al. (USENIX Sec.)</td>
</tr>
<tr>
<td></td>
<td>CCFIR: Zhang et al. (IEEE S&amp;P)</td>
</tr>
<tr>
<td>2013</td>
<td>ROPecker: Cheng et al. (NDSS)</td>
</tr>
<tr>
<td></td>
<td>Forward-Edge CFI: Tice et al. (USENIX Sec.)</td>
</tr>
<tr>
<td></td>
<td>SAFEDISPATCH: Jang et al. (NDSS)</td>
</tr>
<tr>
<td></td>
<td>Modular CFI: Niu et al. (PLDI)</td>
</tr>
<tr>
<td></td>
<td>RockJIT: Niu et al. (CCS)</td>
</tr>
<tr>
<td></td>
<td>Hardware CFI: Davi et al. (DAC)</td>
</tr>
<tr>
<td>2014</td>
<td>Control-Flow Guard: Microsoft</td>
</tr>
<tr>
<td></td>
<td>Per-input CFI: Jang et al. (NDSS)</td>
</tr>
<tr>
<td></td>
<td>CCFI: Mashtizadeh et al. (CCS)</td>
</tr>
<tr>
<td></td>
<td>PathArmor: Veen et al. (CCS)</td>
</tr>
<tr>
<td></td>
<td>HAFIX: Arias et al. (DAC)</td>
</tr>
<tr>
<td>2015</td>
<td>Vtrust: Zhang et al. (NDSS)</td>
</tr>
<tr>
<td></td>
<td>Protecting Vtables: Bounov et al. (NDSS)</td>
</tr>
<tr>
<td></td>
<td>HAFIX++: Sullivan et al. (DAC)</td>
</tr>
<tr>
<td></td>
<td>CET: Intel</td>
</tr>
<tr>
<td>2016</td>
<td></td>
</tr>
</tbody>
</table>
HAFIX: HW-assisted Flow Integrity

HAFIX: Hardware-Assisted Flow Integrity Extension
Design Automation Conference (DAC 2015)
Orlando Arias, Lucas Davi, Matthias Hanreich, Yier Jin, Patrick Koeberl,
Debayan Paul, Ahmad-Reza Sadeghi, Dean Sullivan
Why CFI Processor Support?

CFI Processor Support based on Instruction set architecture (ISA) extensions

- Dedicated CFI instructions
- Avoids offline training phase
- Instant attack detection
- CFI control state: Binding CFI data to CFI state and instructions
HAFIX++

Strategy Without Tactics:
Policy-Agnostic Hardware-Enhanced Control-Flow Integrity
*Design Automation Conference (DAC 2016)*
Dean Sullivan, Orlando Arias, Lucas Davi, Per Larsen, Ahmad-Reza Sadeghi, Yier Jin
<table>
<thead>
<tr>
<th>Objectives</th>
<th>Security</th>
<th>High performance</th>
<th>Enabling technology</th>
<th>Compatibility to legacy code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward-Edge and Forward-Edge CFI</td>
<td>Stateful, CFI policy agnostic</td>
<td>No burden on developer</td>
<td>No code annotations/changes</td>
<td>CFI and non-CFI code on same platform</td>
</tr>
<tr>
<td>No burden on developer</td>
<td>Hardware protection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>On-Chip Memory for CFI Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No unintended sequences</td>
<td>&lt; 3% overhead</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enabling technology</td>
<td>All applications can use CFI features</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatibility to legacy code</td>
<td>Support of Multitasking</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## HAFIX++ ISA Extensions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfiret</td>
<td>Issue at return site $\rightarrow$ check backward edge</td>
</tr>
<tr>
<td>cfichk</td>
<td>Issued at call/jmp target $\rightarrow$ check forward edge</td>
</tr>
<tr>
<td>cfiprj</td>
<td>Issued at jump site $\rightarrow$ setup jump target</td>
</tr>
<tr>
<td>cfilsr</td>
<td>Issued at call site $\rightarrow$ setup call target</td>
</tr>
<tr>
<td>cfibr</td>
<td>Issued at call site $\rightarrow$ setup backward edge</td>
</tr>
</tbody>
</table>

**Label State**
- Stack (LSS)
- Register (LSR)
Control-flow Enforcement Technology

[Intel 2016]

• Backward edge
  • Shadow stack detects return-address manipulation
  • Shadow stack protected, cannot be accessed by attacker
  • New register \texttt{ssp} for the shadow stack
  • Conventional move instructions cannot be used in shadow stack
  • New instructions to operate on shadow stack

• Forward edge
  • New instruction for indirect call/jump targets: \texttt{branchend}
  • \textit{Any indirect call/jump can target any valid indirect branch target}
  • Could be combined with fine-grained compiler-based CFI (LLVM CFI)
<table>
<thead>
<tr>
<th></th>
<th>BE-Support</th>
<th>FE-Support</th>
<th>Shared library &amp; Multitasking</th>
<th>Granularity</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFI</td>
<td>✖️</td>
<td>✔️</td>
<td>✔️</td>
<td>Coarse</td>
<td>3.75%</td>
</tr>
<tr>
<td>Budiu et al, ASID 2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAFIX</td>
<td>✔️</td>
<td>✖️</td>
<td>✖️</td>
<td>Coarse</td>
<td>2%</td>
</tr>
<tr>
<td>Davi et al., DAC 2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LandHere</td>
<td>✖️</td>
<td>✖️</td>
<td>✔️</td>
<td>Coarse</td>
<td>N/A</td>
</tr>
<tr>
<td><a href="http://langalois.com">http://langalois.com</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HCFI</td>
<td>✔️</td>
<td>✔️</td>
<td>✖️</td>
<td>Fine</td>
<td>1%</td>
</tr>
<tr>
<td>Christoulakis et al., CODASPY 2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel CET</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>Coarse</td>
<td>N/A</td>
</tr>
<tr>
<td>Intel Tech Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAFIX++</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>Fine</td>
<td>1.75%</td>
</tr>
<tr>
<td>Sullivan et al., DAC 2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Architectural dependent optimizations

Hardware-Based Solutions

Intel CET

Intel Tech Review

HAFIX++

Sullivan et al., DAC 2016
Leakage: Use-case SGX
Controlled-Channel Attack on SGX [Xu et al., IEEE S&P’15]

Granularity: page 4K, good for big data structures

EPC: Enclave Page Cache  PT: Page Tables  PF: Page-Fault
Cache Attacks on SGX

CPU caches shared between enclaves and untrusted software enabling cache side-channel attacks

EPC: Enclave Page Cache
Cache Attacks on SGX, Cont.

EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

[Schwarz et al., arXiv:1702.08719]

[Lee et al., arXiv:1611.06952], Branch shadowing

[Moghimi et al., arXiv:1703.06986]

[Brasser et al., arXiv:1702.07521]

[Götzfried et al., EuroSec’17]
## SGX Side-Channel Attacks Comparison

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Observed Cache</th>
<th>Interrupting Victim</th>
<th>Cache Eviction Measurement</th>
<th>Attacker Code</th>
<th>Attacked Victim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lee et al. Branch Shadowing</td>
<td>BTB / LBR</td>
<td>Yes</td>
<td>Execution Timing</td>
<td>OS</td>
<td>RSA &amp; SVM classifier</td>
</tr>
<tr>
<td>Moghimi et al. Prime + Probe</td>
<td>L1(D)</td>
<td>Yes</td>
<td>Access timing</td>
<td>OS</td>
<td>AES</td>
</tr>
<tr>
<td>Götzfried et al. Prime + Probe</td>
<td>L1(D)</td>
<td>No</td>
<td>PCM</td>
<td>OS</td>
<td>AES</td>
</tr>
<tr>
<td>Brasser et al. Prime + Probe</td>
<td>L1(D)</td>
<td>No</td>
<td>PCM</td>
<td>OS</td>
<td>RSA &amp; Genome Sequencing</td>
</tr>
<tr>
<td>Schwarz et al. Prime + Probe</td>
<td>L3</td>
<td>No</td>
<td>Counting Thread</td>
<td>Enclave</td>
<td>AES</td>
</tr>
</tbody>
</table>
Our Attack

PMC: Performance Monitoring Counter (e.g., executed cycles, cache hit/misses, ...)

Core 0

Core n
Attack Use-Cases

- Extracting 2048-bit RSA decryption key

- Extracting genome sequences processed in an enclave
Current Countermeasures

• System level defenses
  • Prevent side-channels requiring frequent interruption of enclaves
  • Randomization

• Application level defenses
  • Side-channel resilient programming (hide accessed memory location)

• Obfuscation techniques
  • ORAM, Flushing

• New Hardware Design
  • Sanctum, Bastille, cache partitioning, etc.
Conclusion

• Hardware-assisted security simply not benefiting users
  • Still target of attacks exploiting vulnerabilities of legacy systems
  • Side channel effect is kind of more drastic than though
  • Current add-on defenses not practical or effective

• Directions
  • New business models
  • Automated use of Trusted Computing solutions
  • Artificial Intelligence in Hardware
  • New fast and dense memory technology