HARDWARE-ASSISTED ROOTKITS & INSTRUMENTATION:

ARM EDITION
ABOUT

ENDGAME.

- Offense-based approach to security and hunting adversaries
- Research thrusts in malware, threat intel, data science, and exploit prevention
- Matt Spisak (@matspisak)
  - Vulnerability and exploit mitigation research at Endgame
  - Mobile security since Nokia N series (before iPhone)
OUTLINE

- Motivation
- ARM Debug Architecture
- Tracing and Instrumentation
- Rootkits
- TrustZone
- Exploit Mitigations
**DEBUGGING EMBEDDED SYSTEMS IS COMPLICATED**

### Hardware
- JTAG is a gold standard
- Custom dev boards + Virtualization extensions
- JTAG access can be hit/miss
- Destructive
- Expensive

### Software
- Portable, scalable
- existing tools for HLOS like iOS, Android
- Can be tightly coupled to OS
- Often limited to PL0/EL0
- Lots of reinventing wheel

### Emulation
- Scalable and powerful
- Cost-effective
- Sometimes a good option (e.g. CTF)
- Lack support for HW interfaces
- Requires big time investment
SEARCHING FOR ALTERNATIVES

- What's a good general approach?

- Personal philosophy:
  - Always make use of real hardware
  - Lean towards software-based tools

- **GOAL**: find common ARM architectural debug features accessible from software (on COTS devices)
ARM DEBUG ARCHITECTURE
INVASIVE DEBUG

- Debug-modes: Monitor, Halting, or None
  - Software debug events: BKPT, breakpoint, watchpoint, vector trap
  - Halting debug events result in processor entering debug state
- Support driven by DBGEN and SPIDEN authentication signals
  - if DBGEN is low -> BKPT instruction only event supported
  - Authentication signals typically controlled externally
- Without DBGEN, options are limited
NON-INVASIVE DEBUG

- **Trace**: Embedded Trace Buffer (ETB) / CoreSight Program Flow Trace (PFT)
  - PFT/PTM generates traces for waypoints: branch & exception instructions
  - Accessible from external and **software** (coprocessor or memory-mapped)
  - PFT/PTM can be locked (ETMLAR) - only writeable in memory-mapped
  - memory-mapped access is **IMPLEMENTATION DEFINED**
  - Trace drivers in Android kernel check CoreSight fuse status
  - A potential software-based debug feature for COTS devices
NON-INVASIVE DEBUG

- Sample-based Profiling
  - Registers for sampling Program Counter and Context ID
  - No CP14 visibility, optional memory-mapped and external interfaces
- PMU
  - Focus of remainder of talk
NOT THIS PMU.
THIS PMU.

performance counters
PERFORMANCE MONITORING UNIT (PMU)

- Optional extension, but recommended
- Interfaces: CP15 (mandatory), memory-mapped (optional), external (optional)
- Dates back to ARMv6, common in ARM11, Cortex-R, Cortex-A
- 1 cycle counter, up to 31 general counters
- Set of event filters for counting
- Support for interrupts on counter overflow
PERFORMANCE MONITORING UNIT (PMU)

- Provides real-time feedback on system
- Useful for software/hardware engineers
- Diagnose bugs
- Tools:
  - ARM DS-5 Streamline
  - Linux perf / oprofile
**TERMINOLOGY & ABBREVIATIONS**

- PMU - Performance Monitoring Unit
- PMI - Performance Monitoring Interrupt
- PMC - Performance Monitoring Counter

**ARM Exception Vector Table (EVT)**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td></td>
</tr>
<tr>
<td>Undefined Instruction</td>
<td></td>
</tr>
<tr>
<td>SVC</td>
<td>Supervisor Call (e.g. SYSCALL)</td>
</tr>
<tr>
<td>Prefetch Abort</td>
<td>BKPT, or code Page Fault</td>
</tr>
<tr>
<td>Data Abort</td>
<td>Data Page Fault</td>
</tr>
<tr>
<td>IRQ</td>
<td>Interrupts (Normal World)</td>
</tr>
<tr>
<td>FIQ</td>
<td>Fast Interrupts (Secure World)</td>
</tr>
</tbody>
</table>

**PL Levels**

- **PL3/EL3**: SECURE MONITOR
- **PL2/EL2**: HYPervisor
- **PL1/EL1**: KERNEL MODE
- **PL0/EL0**: USER MODE

**Mode Levels**

- **Ring 3**: Least Privileged
- **Ring 0**: Most Privileged

**Architectures**

- ARM
- x86
PMU RELATED WORK

- "Using Hardware Performance Events for Instruction-Level Monitoring on the x86 Architecture", [Vogl, Eckert]
- ROP detection with PMU using mispredicted RET [Wicherski], [Li, Crouse]
- Rootkit detection with performance counters [Wang, Karri]
- Control-flow integrity using BTS [Xia et al]
- All prior art is focused on Intel / x86 architecture
## Sample ARM PMU Events

<table>
<thead>
<tr>
<th>EVENT TYPE</th>
<th>EVENT CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD RETIRED: Load instruction executed</td>
<td>0x06</td>
</tr>
<tr>
<td>ST RETIRED: Store instruction executed</td>
<td>0x07</td>
</tr>
<tr>
<td>INST RETIRED: Instruction executed</td>
<td>0x08</td>
</tr>
<tr>
<td>PC WRITE RETIRED: Software change of PC</td>
<td>0x0C</td>
</tr>
<tr>
<td>BR RETURN RETIRED: Branch Return retired</td>
<td>0x0E</td>
</tr>
<tr>
<td>BR MISP PRED: Branch mispredicted</td>
<td>0x10</td>
</tr>
<tr>
<td>L1I CACHE: Level 1 instruction cache access</td>
<td>0x14</td>
</tr>
</tbody>
</table>
PMU REGISTERS

- **PMCR - Control Register**

- **N**: Number of counters
- **E**: Enable / Disable all counters
- **ARMv6**: \( \text{MRC/MCR} \ p15, 0, <Rd>, c15, c12, 0 \)
- **ARMv7**: \( \text{MRC/MCR} \ p15, 0, <Rd>, c9, c12, 0 \)
PMU REGISTERS - CONFIGURE COUNTERS

- PMCNTENSET - Enable Counter
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c12, 1

- PMCNTENCLR - Disable Counter
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c12, 2

- PMSELR - Counter Selection Register
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c12, 5

Use this register prior to read/write of event type or counter registers
PMU REGISTERS - CONFIGURE COUNTERS

- **PMXEVTYPER** - Counter Event Filter Register
  - Selects event and modes to count
  - ARMv7: `MRC/MCR p15, 0, <Rd>, c9, c13, 1`

- **PMXEVTCNTR** - Event Counter Register
  - ARMv7: `MRC/MCR p15, 0, <Rd>, c9, c13, 2`

---

### EVENT CODE | MODES INCLUDED
---
0x6800000C | Branches in Secure PL1 and HYP
0x6000000C | Branches in Secure PL1
0x9800000C | Branches in Secure PL0 and HYP
0x9000000C | Branches in Secure PL0
0x3800000C | Branches in Secure PL0, PL1, HYP
0x4000000C | Branches in non-secure PL1
0x8000000C | Branches in non-secure PL0
PMU REGISTERS - CONFIGURE COUNTERS

//Enable armv7 PMU Counters
MRC p15, 0, R1, c9, c12, 0
ORR R1, R1, #1
MCR p15, 0, R1, c9, c12, 0

//Set PMC1 to count Instructions Executed
MOV R1, #1
MCR p15, 0, R1, c9, c12, 5 //PMSELR
MOV R1, #0x8
MCR p15, 0, R1, c9, c13, 1 //PMXEVTPYPER

//Initialize PMC1 to -3
MOV R1, #0xFFFFFFFF
MCR p15, 0, R1, c9, c13, 2 //PMXEVTCNTR

//Enable PMC1
MOV R1, #1
MCR p15, 0, R1, c9, c12, 1 //PMCNTENSET
PMU REGISTERS - CONFIGURE INTERRUPTS

- PMINTENSE - Interrupt Enable Register
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c14, 1

- PMINTENCLR - Interrupt Disable Register
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c14, 2

- PMOVSR - Overflow Status Register
- PMOVSET - Overflow Status Set Register
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c12, 3
  - ARMv7: MRC/MCR p15, 0, <Rd>, c9, c14, 3
PMU REGISTERS - CONFIGURE INTERRUPTS

//Enable Interrupts for PMC1 and PMC2
MOV R1, #3
MCR p15, 0, R1, c9, c14, 1 //PMINTENSE

//Read and Clear Overflow on Interrupt
MRC p15, 0, R0, c9, c12, 3 //PMOVSR
MCR p15, 0, R0, c9, c12, 3 //PMOVSR
DO YOU EVEN COUNT?

- **DBGAUTHSTATUS**
  - Lists whether invasive/non-invasive debug are supported in secure and non-secure worlds
  - **ARMv7**: `MRC/MCR p14, 0, <Rd>, c7, c14, 6`

- **ID_DFR0**
  - Lists PMU version supported (if any)
  - **ARMv7**: `MRC/MCR p15, 0, <Rd>, c0, c1, 2`
## The Center for Chips Who Can Count Good

<table>
<thead>
<tr>
<th>Device</th>
<th>Chipset</th>
<th>DBGAUTHSTATUS</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorola Nexus 6</td>
<td>Qualcomm Snapdragon 805 (4x Krait Core)</td>
<td>Non-Invasive Debug (NIDEN) Enabled</td>
<td>PMUv2</td>
</tr>
<tr>
<td>Amazon Fire HD 7&quot;</td>
<td>MediaTek MT8135 (2x Cortex-A15 + 2x Cortex A7)</td>
<td>Non-Invasive Debug (NIDEN) Enabled Secure Non-Invasive Debug (SPNIDEN) Enabled</td>
<td>PMUv2</td>
</tr>
<tr>
<td>Samsung Galaxy Note 2</td>
<td>Samsung Exynos 4412 (4x Cortex-A9)</td>
<td>Non-Invasive Debug (NIDEN) Enabled Invasive Debug (DBGEN) Enabled</td>
<td>PMUv2</td>
</tr>
<tr>
<td>Huawei Ascend P7</td>
<td>HiSilicon Kirin 910T (4x Cortex-A9)</td>
<td>Non-Invasive Debug (NIDEN) Enabled Secure Non-Invasive Debug (SPNIDEN) Enabled</td>
<td>PMUv2</td>
</tr>
<tr>
<td>Multiple</td>
<td>Broadcom BCM4356 WiFi Chip (Cortex R4)</td>
<td>Non-Invasive Debug (NIDEN) Enabled</td>
<td>PMUv1</td>
</tr>
</tbody>
</table>
CASE STUDY: PMU TRACING
APPROACH

- Make the PMU more invasive with frequent PMC-based traps
- CoreSight Program Flow Trace (PFT) captures waypoints (i.e. branches)
- We can come pretty close to PFT Trace using the PMU:
  - Count all branches: predicted and mispredicted
  - Interrupt all the things: set our counter(s) to -1
  - Use our ISR as the instrumentation logic
APPROACH - BRANCH TRACING

PMC1: 0xFFFFFFFF (-1)  Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
</tbody>
</table>

**func:**
- STMFD SP!, {R0-R2,R4-R9,LR}
- MOV R8, R1
- MOV R1, SP
- MOV R2, R2
- LDR R7, [SP]
- CMP R7, #0
- BEQ error

**error:**
- MOV R4, #0xFFFFFFFF7
- ADD SP, SP, #0xC
### CASE STUDY: PMU TRACING

**APPORACH - BRANCH TRACING**

PMC1: 0xFFFFFFFF (-1)  
Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func: 0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>MOV R8, R1</td>
<td></td>
</tr>
<tr>
<td>MOV R1, SP</td>
<td></td>
</tr>
<tr>
<td>MOV R2, R2</td>
<td></td>
</tr>
<tr>
<td>LDR R7, [SP]</td>
<td></td>
</tr>
<tr>
<td>CMP R7, #0</td>
<td></td>
</tr>
<tr>
<td>BEQ error</td>
<td></td>
</tr>
<tr>
<td>error:</td>
<td>MOV R4, #0xFFFFFFFF7</td>
</tr>
<tr>
<td>ADD SP, SP, #0xC</td>
<td></td>
</tr>
</tbody>
</table>

**PMU ISR**
- CAPTURE PC
- CAPTURE REGS
- MEMORY SNAPSHOT
- RESET COUNTER
**APPROACH - BRANCH TRACING**

PMC1: 0xFFFFFFFF (-1)  
Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func:</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>0</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td></td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td></td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td></td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

**PMU ISR**
- Capture PC
- Capture Regs
- Memory Snapshot
- Reset Counter

```asm
func:
error:
```
### Approach - Branch Tracing

**PMC1: 0xFFFFFFFF (-1)**

**Event: 0x0C (All Branches)**

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func:0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td></td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td></td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td></td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

**Error:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVR4, #0xFFFFFFF7</td>
<td></td>
</tr>
<tr>
<td>ADD SP, SP, #0xC</td>
<td></td>
</tr>
</tbody>
</table>
**FUNCTION**

PMC1: 0xFFFFFFFF (-1) Event: 0x0C (All Branches)

### PMC  INSTRUCTION

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func: 0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td></td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td></td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
<tr>
<td>error:</td>
<td>MOV R4, #0xFFFFF7</td>
</tr>
<tr>
<td></td>
<td>ADD SP, SP, #0xC</td>
</tr>
</tbody>
</table>

**PMU ISR**
- CAPTURE PC
- CAPTURE REGS
- MEMORY SNAPSHOT
- RESET COUNTER
## Approach - Branch Tracing

**PMC1**: 0xFFFFFFFF (-1)  
**Event**: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func: 0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td>-1</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td></td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

### Error:
- MOV R4, #0xFFFFFFFF7
- ADD SP, SP, #0xC

---

### PMU ISR
- Capture PC
- Capture REGS
- Memory Snapshot
- Reset Counter
Approach - Branch Tracing

PMU1: 0xFFFFFFFF (-1)  Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func: 0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td>-1</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>-1</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
<tr>
<td></td>
<td>MOV R4, #0xFFFFFFFF7</td>
</tr>
<tr>
<td></td>
<td>ADD SP, SP, #0xC</td>
</tr>
</tbody>
</table>

PMU ISR:
- Capture PC
- Capture Regs
- Memory Snapshot
- Reset Counter
# Case Study: PMU Tracing

## Approach - Branch Tracing

**PMC1:** 0xFFFFFFFF (-1)  
**Event:** 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func:0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td>-1</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>-1</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td>-1</td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

**Error:**

- MOV R4, #0xFFFFFFFC7
- ADD SP, SP, #0xC

**PMU ISR**
- Capture PC
- Capture Regs
- Memory Snapshot
- Reset Counter
APPORACH – BRANCH TRACING

PMC1: 0xFFFFFFFF (-1)   Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func: 0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td>-1</td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td>-1</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>-1</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td>-1</td>
<td>BEQ error</td>
</tr>
<tr>
<td>error: 0</td>
<td>MOV R4, #0xFFFFFFF7</td>
</tr>
<tr>
<td></td>
<td>ADD SP, SP, #0xC</td>
</tr>
</tbody>
</table>

PMU ISR
- Capture PC
- Capture_REGS
- Memory Snapshot
- Reset Counter

PMU ISR
- Capture PC
- Capture_REGS
- Memory Snapshot
- Reset Counter

overflow
## APPROACH – BRANCH TRACING

### PMC1: 0xFFFFFFFF (−1)  
Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>−1</td>
<td>BL func</td>
</tr>
<tr>
<td>0</td>
<td>STMFD SP!, {R0-R2,R4-R9,LR}</td>
</tr>
<tr>
<td>−1</td>
<td>MOV R8, R1</td>
</tr>
<tr>
<td>−1</td>
<td>MOV R1, SP</td>
</tr>
<tr>
<td>−1</td>
<td>MOV R2, R2</td>
</tr>
<tr>
<td>−1</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>−1</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td>−1</td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

### func:

-1  BL func

### error:

0   MOV R4, #0xFFFFFFFF7
-1  ADD SP, SP, #0xC

**PMU ISR**
- Capture PC
- Capture Regs
- Memory Snapshot
- Reset Counter
BUT WHAT ABOUT LINUX PERF?

- We want a custom ISR for instrumentation
- Too tightly coupled to Linux
- Invoking API’s != learning
- But perf source can be useful for understanding PMU interfaces
WHERE'S THE PMU INTERRUPT?
ARM GENERIC INTERRUPT CONTROLLER (GIC) SPECIFICATION

### INTID | Interrupt type | Details
--- | --- | ---
ID0 – ID15 | SGI | These interrupts are local to a CPU interface.
ID16 – ID31 | PPI | Shared peripheral interrupts that the Distributor can route to either a specific PE, or to any one of the PEs in the system that is a participating node, see Participating nodes on page 3-44.
ID32 – ID1019 | SPI | 

- SGI: Software Generated Interrupts
- PPI: Private Peripheral Interrupts
- SPI: Shared Peripheral Interrupts
- ARM GIC spec recommends PMU Overflows to use INTID 23
CASE STUDY: PMU TRACING

CHALLENGE: FINDING PMU INTERRUPTS

- Device Tree Source

  ```
  cpu-pmu {
    compatible = "qcom,krait-pmu";
    qcom,irq-is-percpu;
    interrupts = <1 7 0xf00>;
  };
  ```

- Brute Force

  - Register all unused PPI’s & SPI’s, trigger PMIs, diff /proc/interrupts

- Implementation:

  - Android: request_per_cpu_irq(), request_threaded_irq()
  
  - Embedded firmware: patch IRQ vector handler
# Challenge: Interrupt Shadow

PMC1: 0xFFFFFFFF (-1)  
Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
</tbody>
</table>

`func:`
- LDR R7, [SP]
- CMP R7, #0
- BEQ error

`error:`
- MOV R4, #0xFFFFFFFF7
- ADD SP, SP, #0xC
## CHALLENGE: INTERRUPT SHADOW

**PMC1**: 0xFFFFFFFF (-1)  
**Event**: 0x0C (All Branches)

### PMC

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL</td>
</tr>
<tr>
<td>func:</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>0</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

### overflow

- `BL func`

### error:

- `MOV R4, #0xffffffff`
- `ADD SP, SP, #0xc`
### CHALLENGE: INTERRUPT SHADOW

PMC1: 0xFFFFFFFF (-1)  Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL</td>
</tr>
<tr>
<td>func:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>0</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td></td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

error:

- MOV R4, #0xFFFFFFFF7
- ADD SP, SP, #0xC
**CHALLENGE: INTERRUPT SHADOW**

PMC1: 0xFFFFFFFF (-1)  
Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL  func</td>
</tr>
<tr>
<td>func:</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>0</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td>0</td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

overflow

error:

MOV R4, #0xFFFFFFFF7
ADD SP, SP, #0xC
## Challenge: Interrupt Shadow

PMC1: 0xFFFFFFFF (-1) Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
<tr>
<td>func:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>LDR R7, [SP]</td>
</tr>
<tr>
<td>0</td>
<td>CMP R7, #0</td>
</tr>
<tr>
<td>0</td>
<td>BEQ error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>error:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>MOV R4, #0xFFFFFFFF</td>
</tr>
<tr>
<td>ADD SP, SP, #0xC</td>
</tr>
</tbody>
</table>
**CHALLENGE: INTERRUPT SHADOW**

PMC1: 0xFFFFFFFF (-1)  Event: 0x0C (All Branches)

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL</td>
</tr>
<tr>
<td>func:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>LDR</td>
</tr>
<tr>
<td>0</td>
<td>CMP</td>
</tr>
<tr>
<td>0</td>
<td>BEQ</td>
</tr>
<tr>
<td>error:</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>MOV</td>
</tr>
<tr>
<td>1</td>
<td>ADD</td>
</tr>
</tbody>
</table>

**PMU ISR**
- CAPTURE PC
- CAPTURE REGS
- MEMORY SNAPSHOT
- RESET COUNTER
## CHALLENGE: INTERRUPT SHADOW

### PMC1: 0xFFFFFFFF (-1)  
Event: 0x0C (All Branches)

### PMC

<table>
<thead>
<tr>
<th>PMC</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>BL func</td>
</tr>
</tbody>
</table>

#### func:

| 0    | LDR R7, [SP]         |
| 0    | CMP R7, #0          |
| 0    | BEQ error           |

#### error:

| 1    | MOV R4, #0xFFFFFFFF7 |
| 1    | ADD SP, SP, #0xC     |

**Causes miss of up to 15% covered basic blocks**

**Interrupt Shadow**

Skid = 4 Instructions

**PMU ISR**

- CAPTURE PC
- CAPTURE REGS
- MEMORY SNAPSHOT
- RESET COUNTER
OTHER CHALLENGES

- CPU Hot-Plugging – easy solution for Android: register_hotcpu_notifier()
- Lack of Last Branch Recording feature on ARM
- Complicated kernel mode instrumentation: use sampling period of -2

- Requires small patch to entry-armv.S (or hot patch)
Captures PC at time of interrupt, buffered per core

Visualize coverage and control pmu_server to select threads, mode, and start/stop

pmu_server

pmutrace.ko
CONNECTING THE DOTS

- Use IDA to our advantage
- For each PMU waypoint:
  - Color/count all instructions in Basic Block
  - If only 1 xref from basic block: count/color it
  - If only 1 xref to basic block: count/color it

Example of a perfect PMU branch tracing run
CONNECTING THE DOTS

- Interrupt shadow

- Basic block xref algorithm helps fill in missed blocks

- Fuzzing / code coverage will eventually be interrupted in this block

- Could improve by adding 2nd counter to count instructions between interrupts
DEMO: PMU TRACING

DEVICE REQUIREMENTS:

- ROOTED
- CONFIG_MODULES OPTION (NOT AS COMMON)
- CONFIG_PREEMPT OPTION (COMMON)
- IRQ HANDLER PATCH (PL1/EL1)
ANDROID INSTRUMENTATION. SO WHAT?

- Recall approach is hardware-assisted - not tied to a specific OS
- Less invasive than BKPT tracing
- Supports both user mode and kernel mode instrumentation
- Not limited to branch tracing, other potential instrumentation use-cases
- And these chips can count too:
  - Broadcom WiFi; Intel/Infineon, MediaTek + other ARM Cellular Basebands
  - Apple ARM SoCs
  - PowerPC, MIPS
CASE STUDY: PMU ROOTKITS
PRIOR ART IN ARM ROOTKITS

- Traditional rootkits: modify syscall table or EVT [Phrack Issue 68]
- Suterusu performs hot patching of kernel functions [Coppola]
- Cloaker toggles SCTLR to move EVT [David et al]
- Clock Locking Beats explores using CPU governor for hiding cycles [Thomas]
- TrustZone based rootkit [Roth]
### INSPIRATION

Table C-1 PMU IMPLEMENTATION DEFINED event numbers (continued)

<table>
<thead>
<tr>
<th>Event number</th>
<th>Event mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7F-0x80</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x81</td>
<td>EXC_UNDEF</td>
<td>Exception taken, Undefined Instruction</td>
</tr>
<tr>
<td>0x82</td>
<td>EXC_SVC</td>
<td>Exception taken, Supervisor Call</td>
</tr>
<tr>
<td>0x83</td>
<td>EXC_PABORT</td>
<td>Exception taken, Prefetch Abort</td>
</tr>
<tr>
<td>0x84</td>
<td>EXC_DABORT</td>
<td>Exception taken, Data Abort</td>
</tr>
<tr>
<td>0x85</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x86</td>
<td>EXC_IRQ</td>
<td>Exception taken, IRQ</td>
</tr>
<tr>
<td>0x87</td>
<td>EXC_FIQ</td>
<td>Exception taken, FIQ</td>
</tr>
<tr>
<td>0x88</td>
<td>EXC_SMC</td>
<td>Exception taken, Secure Monitor Call</td>
</tr>
<tr>
<td>0x89</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x8A</td>
<td>EXC_HVC</td>
<td>Exception taken, Hypervisor Call</td>
</tr>
</tbody>
</table>

AH AH AH very interesting....
QUICK NOTE ON ARM LICENSES

- ARM Core License
  - Use core ARM designs

- ARM Architectural license
  - Enables custom cores provided it implements an ARM instruction set
  - Examples: Qualcomm Scorpion/Krait/Kryo, Apple A6/A7/etc.
## Counting the Exception Vector Table

<table>
<thead>
<tr>
<th>EVENT</th>
<th>Cortex-A7</th>
<th>Cortex-A53</th>
<th>Cortex-A57</th>
<th>Cortex-A72</th>
<th>Scorpion</th>
<th>Krait</th>
<th>Kryo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined Instruction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>SVC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Prefetch Abort</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Data Abort</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>IRQ</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>FIQ</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>SMC</td>
<td>*</td>
<td>*</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>HVC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
**DOWN THE RABBIT HOLE**

<table>
<thead>
<tr>
<th>CRn</th>
<th>opc1</th>
<th>CRm</th>
<th>opc2</th>
<th>Reserved for Branch Predictor, Cache and TCM operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>c9</td>
<td>{0-7}</td>
<td>{c0-c2}</td>
<td>{0-7}</td>
<td>Reserved for Branch Predictor, Cache and TCM operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{c5-c8}</td>
<td>{0-7}</td>
<td>Reserved for ARM Performance Monitors Extension</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{c12-c14}</td>
<td>{0-7}</td>
<td>Reserved for IMPLEMENTATION DEFINED performance monitors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c15</td>
<td>{0-7}</td>
<td>Access depends on the operation</td>
</tr>
</tbody>
</table>

- **Read-only**
- **Read/Write**
- **Write-only**

* ARM Architecture Manual ARMv7-A&R

- Chipset vendors with proprietary PMU implementations:
  - Qualcomm
  - Apple
  - Likely others

Covered in earlier slides
# Case Study: PMU Rootkits

## Scorpion
- **Year:** 2008
- **Architecture:** ARMv7
- **Core Count:** 1-2 Cores
- **Devices:**
  - Qualcomm Snapdragon S1/S2/S3
  - BlackBerry Bold 9900
  - Samsung Galaxy S2 (LTE)
  - Nokia Lumia 900
  - HTC Droid Incredible

## Krait
- **Year:** 2012
- **Architecture:** ARMv7
- **Core Count:** 2 or 4 Cores
- **Devices:**
  - Qualcomm Snapdragon S4/400/600/800/805
  - Nexus 4/5/6/7
  - Samsung Galaxy S4/S5
  - HTC One M8
  - LG G3

## Kryo
- **Year:** 2015
- **Architecture:** ARMv8
- **Core Count:** 4 Cores
- **Devices:**
  - Qualcomm Snapdragon 818/820/823
  - LG G5
  - Samsung Galaxy S7
  - HTC 10
  - Xiaomi Mi 5
QUALCOMM KRAIT PMU

- Adds 4 event select registers: 1 for Venum VFP, 3 for other components of CPU
- Krait event encoded using code + group + region => (code << 8 * group)
- ARM event select register (PMXEVTYPER) set to link to Krait region and group

<table>
<thead>
<tr>
<th>Krait Region 0</th>
<th>Krait Region 1</th>
<th>Krait Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRC/MCR p15, 1, &lt;Rd&gt;, c9, c15, 0</td>
<td>MRC/MCR p15, 1, &lt;Rd&gt;, c9, c15, 1</td>
<td>MRC/MCR p15, 1, &lt;Rd&gt;, c9, c15, 2</td>
</tr>
<tr>
<td>Interrupts/Exceptions + other</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>~100 event codes</td>
<td>~128 event codes</td>
<td>~156 event codes</td>
</tr>
<tr>
<td>PMXEVTYPER = 0xCC</td>
<td>group</td>
<td>PMXEVTYPER = 0xD0</td>
</tr>
</tbody>
</table>

Only a few documented in old Scorpion src. Black-box analysis used to determine # of events
QUALCOMM KRAIT PMU

- Configure Krait + ARM PMU to count Prefetch Aborts:
  
  - Krait Event Code: \textcolor{red}{0x0B}  \hspace{1cm} \text{group: 3} \hspace{1cm} \text{Region: 0}

```c
/*Set Krait Region 0 event selection register
To count Prefetch Aborts*/
MRC p15, 0, R1, c9, c15, 0
ORR R1, R1, #0x8b000000
MCR p15, 0, R1, c9, c15, 0

//Set PMXEVTYPER to point to krait region 0
MOV R1, #0xCF
MCR p15, 0, R1, c9, c13, 1
```
PMU-ASSISTED ROOTKITS

- Trap SVC instructions via PMU
- Use ISR to filter system calls, and redirect code execution after servicing PMI
- Avoids patch protection*
- Installation: a few instructions to initialize PMU registers, and then register ISR for PMU interrupts
### Challenge: Delayed Instruction Skid

- PMI serviced at some point after IRQs enabled in `vector_swi`
- 3 cases we must deal with:
  1. **PMI before branch to syscall routine within `vector_swi`**
  2. **PMI at entry point of syscall routine**
  3. **PMI in middle of syscall routine**

#### Case 1: 92.8%

```
vector_swi:
---
MCR   p15, 0, R12, c1, c0, 0
CPSIE i
MOV   R9, SP, LSR#13
MOV   R9, R9, LSL#13
ADR   R8, sys_call_table
LDR   R10, [R9]
STMFD SP!, {R4,R5}
TST   R10, #0xF00
BNE   __sys_trace
CMP   R7, #0x17C
ADR   LR, ret_fast_syscall
LDRCC PC, [R8, R7, LSL#2]
```

#### Case 2: 2.4%

```
sys_read
---
STMFD SP!, {R0-R2,R4-R9,LR}
MOV   R8, R1
MOV   R1, SP
MOV   R9, R2
BL    fget_light
SUBS  R6, R0, #0
.....
```

#### Case 3: 4.7%

```
#define CPSIE_ADDR 0xC01064D0

```c
irq_regs = get_irq_regs(); //get SVC mode regs
pregs = task_pt_regs(current); //get user mode regs
...
if (pregs->ARM_r7 == 0x3) //sys_read
{
    switch (irq_regs->ARM_pc - CPSIE_ADDR) //offset after CPSIE
    {
        //emulate remaining instructions up to LDRCC
        //can skip those involved in resolving syscall routine
        case 0x0:
        case 0x4:
            irq_regs->ARM_r9 = irq_regs->ARM_sp & 0xFFFFE000;
            ...
        case 0x14:
        case 0x18:
        case 0x1C:
        case 0x20:
            irq_regs->ARM_lr = ret_fast_syscall;
        case 0x24:
            irq_regs->ARM_pc = (uint32_t)hook_sysread;
    }
}
```
CASE STUDY: PMU ROOTKITS

CASE 2: SYSCALL ROUTINE ENTRY POINT

- Replace saved PC with address of hook

```c
irq_regs = get_irq_regs();
pregs = task_pt_regs(current);
...
if (pregs->ARM_r7 == 0x3) //sys_read
{
    //Check if PMU interrupted at entry point addr of sys_read
    if (pregs->ARM_pc == orig_sys_read)
    {
        pregss->ARM_pc = (uint32_t)hook_sys_read;
    }
}
```
We will let syscall routine complete

Find address of ret_fast_syscall on the stack and replace with address of trampoline

Trampoline loads LR with ret_fast_syscall, and branches to appropriate post_hook function

post_hook can retrieve original params from saved user mode registers, and modify as necessary
DEMO: PMU ROOTKIT
PROCESS AND FILE HIDING WITH SYS_GETDENTS64 PMU SVC TRAPS
FUN WITH QMI

- Linux rootkits are boring. This is a phone...
- Hook `sys_read` in context of `qmuxd` in order to intercept all QMI comms from modem to Android (using only the PMU)
DEMO: PMU ROOTKIT
INTERCEPTING QMI WITH SYS_READ PMU SVC TRAPS
MOTOROLA NEXUS 6
QCOM APQ8084 (KRAIT) CPU
ANALYSIS AND LIMITATIONS

- PMU trap on SVC instructions adds less than 5% overhead (2-3%)
- Should evade current kernel integrity monitor algorithms
- PMU registers do not persist a core reset
- Any other code at PL1/EL1 or higher can read/write the registers
DETECTION STRATEGIES

- `/proc/interrupts` --> easy to modify and cloak
- Reading PMU registers looking for someone counting SVCs
  - Access to PMU registers can be trapped to HYP mode
  - Not all usage of PMU in this way is malicious...
- `irq_handler_entry/irq_handler_exit` tracepoints
- Validate IRQ handler addresses by iterating radix tree structure
  - PMU Traps on Data & Prefetch Aborts for ShadowWalker?
CASE STUDY: PMU DEFENSE
EXPLOIT DETECTION FROM THE KERNEL

- Trap SVC instructions to perform syscall monitoring
  - Detect ROP behavior (e.g. EMET / ROPGuard checks)
  - Doesn’t increase attack surface to protected user space binaries
  - Much easier to implement than Rootkit since no re-direction required
  - Protect COTS binaries (i.e no source/compiler required)
  - No modifications to kernel image - just need ISR registered
Case Study: PMU Defense

Android CVE’s in Media

- **libstagefright**
  - Information Disclosure: 4
  - Remote Code Execution: 23
  - Elevation of Privilege: 3
  - Denial of Service: 1

- **mediaserver**
  - Information Disclosure: 10
  - Remote Code Execution: 18
  - Elevation of Privilege: 25
  - Denial of Service: 4

*Aug 2015 - Jun 2016*
DEMO: PMU DEFENSE
BLOCKING STAGEFRIGHT ROP CHAIN FROM THE KERNEL

CVE-2015-3864
POC’s courtesy Mark Brand, Google & NorthBit’s Metaphor

LG NEXUS 5
QCOM MSM8974 (KRAIT) CPU
FUTURE WORK

- Port instrumentation approach to basebands
- Analyze Apple hardware for PMU features and explore iOS kernel tracing
ACKNOWLEDGEMENTS

- Cody Pierce, Endgame
- Eric Miller, Endgame
- Jamie Butler, Endgame
- Several others at Endgame
- Researchers that paved the way for PMU assisted security research
QUESTIONS?
OR FEEDBACK

mspisak at endgame.com
@matspisak
ENDGAME.


