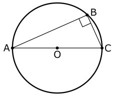
Trust Zone is not enough

Pascal & Muhammad

Dezember 30, Leipzig #34C3

Papers, please !

- Pascal (aka @Pascal_r2)
- Engineer by day
- Researcher by night
  (used to be an associate professor)



- Muhammad Abdul Wahab
- Contact : @Mabdulwahabp
- 3rd year PhD student at IETR, France

Presentation (after my talk !), links, etc :
`https://github.com/pcotret/34c3-trustzone-is-not-enough`

Background (from #34C3 !)

Computer architecture, embedded security. . .

- Alastair, *How can you trust formally verified software ?* (day 1).
- Keegan, *Microarchitectural Attacks on Trusted Execution Environments* (day 1).

FPGA stuff

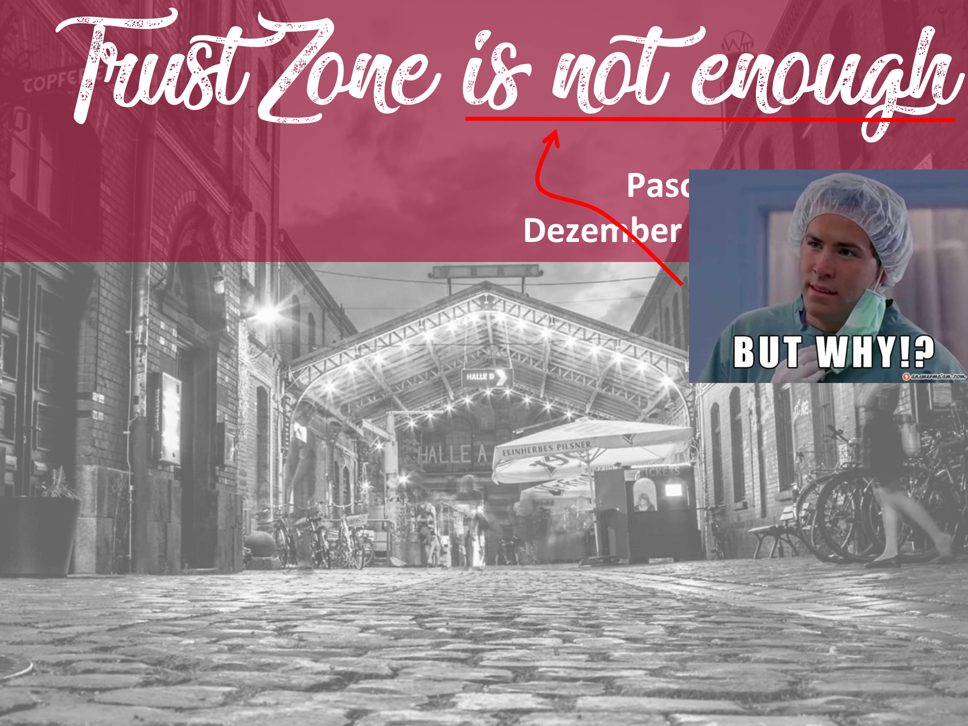
- OpenFPGA assembly.
- Icestorm+Symbiflow tools :
  - `http://www.clifford.at/icestorm/`
  - `https://symbiflow.github.io/`
- Talk on day 2 (FPGA reverse engineering)

Trust Zone is not enough

Pascal & Muhammad

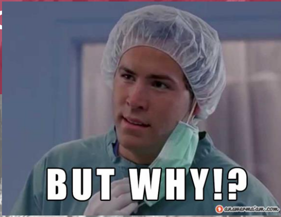
Dezember 30, Leipzig #34C3

Trust Zone is not enough

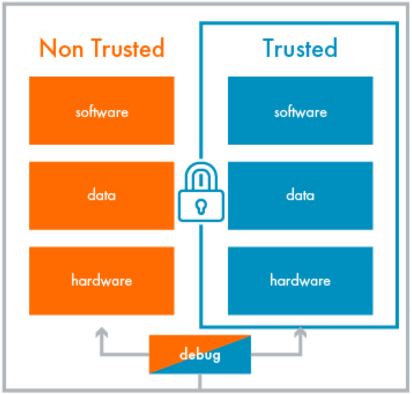


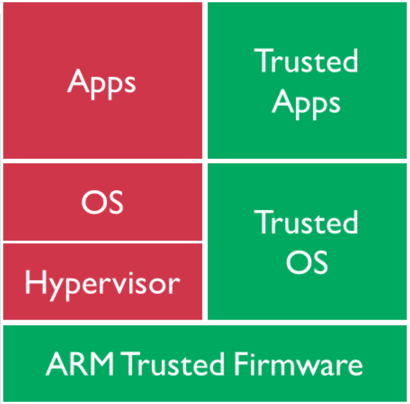
Pasc

Dezember



Why TrustZone is not enough ?



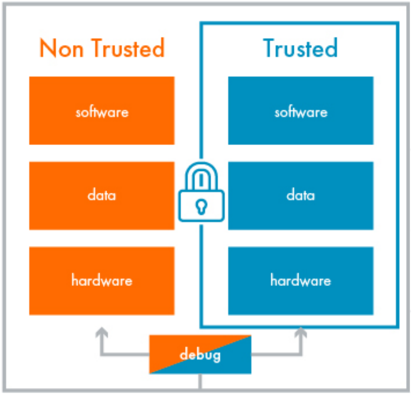


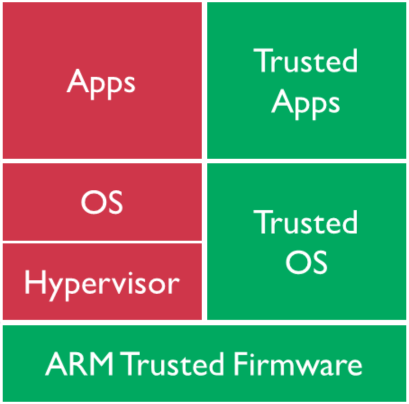
Further reading :

ARM Security Technology, Building a Secure System using TrustZone Technology +

Console Security - Switch, Homebrew on the Horizon (day2 talk)

Why TrustZone is not enough ?





Further reading :

ARM Security Technology, Building a Secure System using TrustZone Technology +

Console Security - Switch, Homebrew on the Horizon (day2 talk)

$\Rightarrow$ This talk is something complementary :)

Outline (normally :p)

Software Security

SoC = Hardcore CPU + FPGA (+ Peripherals)

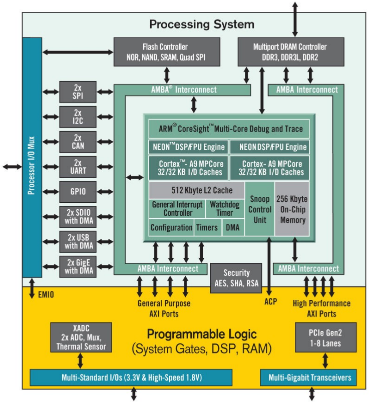


FIGURE – Zynq SoC

Source : Xilinx

SoC = Hardcore CPU + FPGA (+ Peripherals)

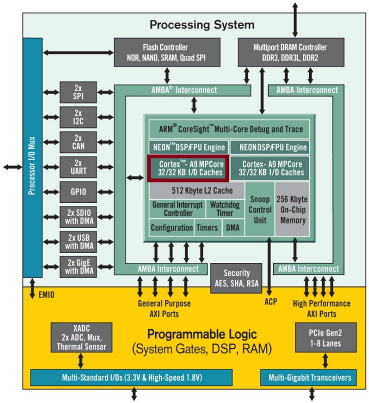

Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

FIGURE – Zynq SoC

Source : Xilinx

Dynamic Information Flow Tracking (DIFT)

Information flow

Information flow is the transfer of information from an information container c_1 to c_2 in a given process P.

$$c_1 \underset{P}{\longrightarrow} c_2$$

Dynamic Information Flow Tracking (DIFT)

Information flow

Information flow is the transfer of information from an information container c_1 to c_2 in a given process P.

$$c_1 \xrightarrow[P]{} c_2$$

Example

```
int a, b, w, x;
a = 11;
b = 5;
w = a * 2;
x = b + 1;
```

DIFT Example : Memory corruption

Attacker overwrites return address and takes control

```
int idx = tainted_input; //stdin (> BUFFER SIZE)
buffer[idx] = x; // buffer overflow
```

set r1 $\leftarrow$ &tainted_input
load r2 $\leftarrow$ M[r1]
add r4 $\leftarrow$ r2 + r3
store M[r4] $\leftarrow$ r5

T	Data
🟩	`r1:&input`
🟩	`r2:idx=input`
🟩	`r3:&buffer`
🟩	`r4:&buffer+idx`
🟩	`r5:x`

T	Data
🟩	Return Address
🟩	int buffer[Size]

DIFT Example : Memory corruption

Attacker overwrites return address and takes control

```
int idx = tainted_input; //stdin (> BUFFER SIZE)
buffer[idx] = x; // buffer overflow
```

set r1 ← &tainted_input
load r2 ← M[r1]
add r4 ← r2 + r3
store M[r4] ← r5

T	Data
🟥	r1:&input
🟩	r2:idx=input
🟩	r3:&buffer
🟩	r4:&buffer+idx
🟩	r5:x

T	Data
🟩	Return Address
🟩	int buffer[Size]

DIFT Example : Memory corruption

Attacker overwrites return address and takes control

```
int idx = tainted_input; //stdin (> BUFFER SIZE)
buffer[idx] = x; // buffer overflow
```

set r1 $\leftarrow$ &tainted_input
load r2 $\leftarrow$ M[r1]
add r4 $\leftarrow$ r2 + r3
store M[r4] $\leftarrow$ r5

T	Data
🟥	r1:&input
🟥	r2:idx=input
🟩	r3:&buffer
🟩	r4:&buffer+idx
🟩	r5:x

T	Data
🟩	Return Address
🟩	int buffer[Size]

DIFT Example : Memory corruption

Attacker overwrites return address and takes control

```
int idx = tainted_input; //stdin (> BUFFER SIZE)
buffer[idx] = x; // buffer overflow
```

set r1 ← &tainted_input
load r2 ← M[r1]
add r4 ← r2 + r3
store M[r4] ← r5

T	Data
🟥	`r1:&input`
🟥	`r2:idx=input`
🟩	`r3:&buffer`
🟥	`r4:&buffer+idx`
🟩	`r5:x`

T	Data
🟩	Return Address
🟩	int buffer[Size]

DIFT Example : Memory corruption

Attacker overwrites return address and takes control

```
int idx = tainted_input; //stdin (> BUFFER SIZE)
buffer[idx] = x; // buffer overflow
```

set r1 ← &tainted_input
load r2 ← M[r1]
add r4 ← r2 + r3
store M[r4] ← r5

T	Data
🟥	r1:&input
🟥	r2:idx=input
🟩	r3:&buffer
🟥	r4:&buffer+idx
🟥	r5:x

T	Data
🟥	Return Address
🟩	int buffer[Size]

DIFT used for DLP (Data Leakage Prevention)

```
char buffer[20]; FILE *fs;
if(geteuid() != 0){ // user
  fs = fopen("welcome", "r"); //public
  if(!fs) exit (1);}
else{ // root
  fs = fopen("passwd", "r"); //secret
  if(!fs) exit(1);}
fread(buffer, 1, sizeof(buffer), fs);
fclose(fs);
printf("Buffer Value: %s \n", buffer);
```

- Compilation ⇒ assembly code
- System calls modified to send tag
- Future : OS integrating support for DIFT

Related work

Different levels

- Application level
  - Java / Android, Javascript, C
- OS level
  - Laminar
  - HiStar
  - kBlare [1]

1. Jacob Zimmermann, Ludovic Mé, and Christophe Bidan. Introducing Reference Flow Control for Detecting Intrusion Symptoms at the OS Level. In : RAID 2002.

Related work

Different levels

- Application level
  - Java / Android, Javascript, C
- OS level
  - Laminar
  - HiStar
  - kBlare[1]
- **Low level**
  - Raksha (Kannan et al.)
  - Flexitaint (Venkataramani et al.)
  - Flexcore (Deng et al.)
  - PAU (Heo et al.)

www.blare-ids.org

1. Jacob Zimmermann, Ludovic Mé, and Christophe Bidan. Introducing Reference Flow Control for Detecting Intrusion Symptoms at the OS Level. In : RAID 2002.

Related work

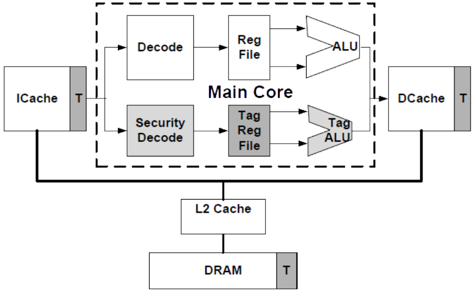


FIGURE – In-core DIFT

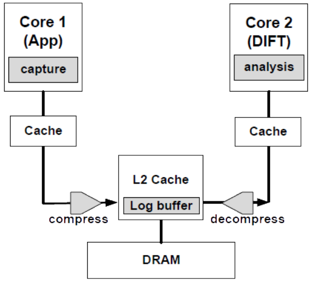


FIGURE – Offloading DIFT

Related work

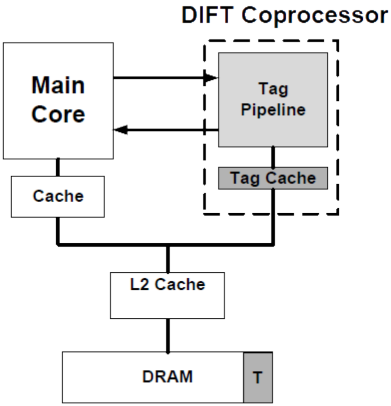


FIGURE – Off-core DIFT (Kannan et al.[2])

2. Hari Kannan, Michael Dalton, and Christos ozyrakis. Decoupling dynamic information flow tracking with a dedicated coprocessor. In : Dependable Systems & Networks, 2009. IEEE. 2009, pp. 105-114.

Related work

		Advantages	Disadvantages
	Software	Flexible security policies Multiple attacks detected	Overhead (from 300% to 3700%)
HW-assisted	**In-core DIFT**	Low overhead (<10%)	Invasive modifications Few security policies
HW-assisted	**Dedicated CPU for DIFT**	Low overhead (<10%) Few modifications to CPU	Wasting resources Energy consumption (x 2)
HW-assisted	**Dedicated DIFT Coprocessor**	Flexible security policies Low overhead (<10%) CPU not modified	Communication between CPU and DIFT Coprocessor

Related work - Limits and Issues

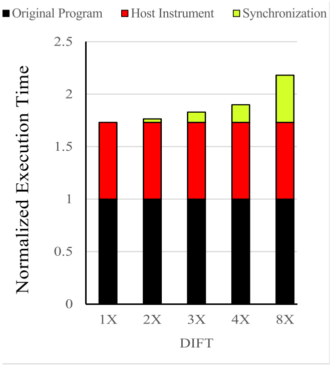


FIGURE – Instrumentation overhead compared to overall DIFT execution time overhead
Source : Heo et al. [3]

"Instrumentation is the transformation of a program into its own measurement tool"
Implementing an LLVM-based Dynamic Binary Instrumentation framework (day2 #34C3)

3. Ingoo Heo et al. Implementing an Application-Specific Instruction-Set Processor for System-Level Dynamic Program Analysis Engines. In : ACM TODAES. 20.4 (2015), p. 53.

Related work

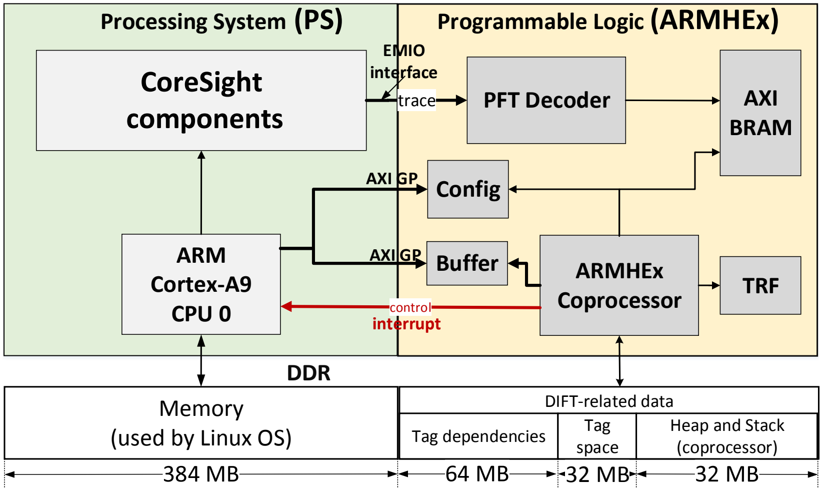
ARMHEx approach

- **Reduce overhead of software instrumentation** as it represents the major portion of overall DIFT execution time overhead
- Lack of **security of DIFT coprocessor**
- **No existing work targets ARM-based SoCs** (related work implementations on softcores)
- **Additional challenges**
  - Limited visibility
  - Frequency gap between CPU and DIFT coprocessor
  - Communication interface, . . .

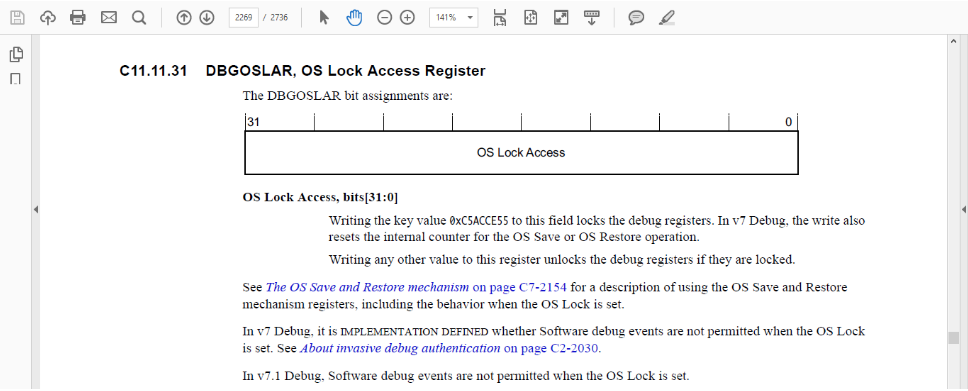
“Black-box testing is fun ...except that it isn’t.”

@plutoo/@derrek/@naehrwert, Console Security - Switch (day2 #34C3)

Overall architecture



ARM-v7 Technical Reference Manual



C11.11.31 DBGOSLAR, OS Lock Access Register

The DBGOSLAR bit assignments are:

31 … 0
OS Lock Access

OS Lock Access, bits[31:0]

Writing the key value `0xC5ACCE55` to this field locks the debug registers. In v7 Debug, the write also resets the internal counter for the OS Save or OS Restore operation.

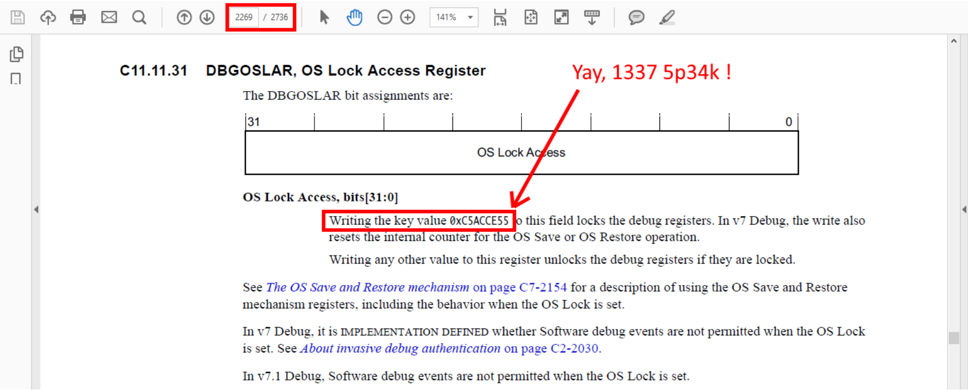
Writing any other value to this register unlocks the debug registers if they are locked.

See *The OS Save and Restore mechanism* on page C7-2154 for a description of using the OS Save and Restore mechanism registers, including the behavior when the OS Lock is set.

In v7 Debug, it is IMPLEMENTATION DEFINED whether Software debug events are not permitted when the OS Lock is set. See *About invasive debug authentication* on page C2-2030.

In v7.1 Debug, Software debug events are not permitted when the OS Lock is set.

ARM-v7 Technical Reference Manual



C11.11.31 DBGOSLAR, OS Lock Access Register

The DBGOSLAR bit assignments are:

OS Lock Access, bits[31:0]

Writing the key value `0xC5ACCE55` to this field locks the debug registers. In v7 Debug, the write also resets the internal counter for the OS Save or OS Restore operation.

Writing any other value to this register unlocks the debug registers if they are locked.

See *The OS Save and Restore mechanism* on page C7-2154 for a description of using the OS Save and Restore mechanism registers, including the behavior when the OS Lock is set.

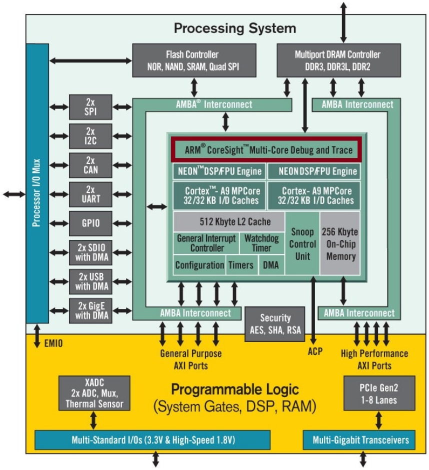
In v7 Debug, it is IMPLEMENTATION DEFINED whether Software debug events are not permitted when the OS Lock is set. See *About invasive debug authentication* on page C2-2030.

In v7.1 Debug, Software debug events are not permitted when the OS Lock is set.

- ARM-v7 TRM : 2736 pages
- ARM-v8 TRM : 6666 pages ⇒ srsly ? ! ?
- ARM-v9 TRM : too many pages (prediction)

Coresight components

A set of IP blocks providing HW-assisted system tracing



Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

FIGURE – ARM Coresight components in Zynq SoC

Source : ARM CoreSight components TRM

Coresight components

A set of IP blocks providing HW-assisted system tracing

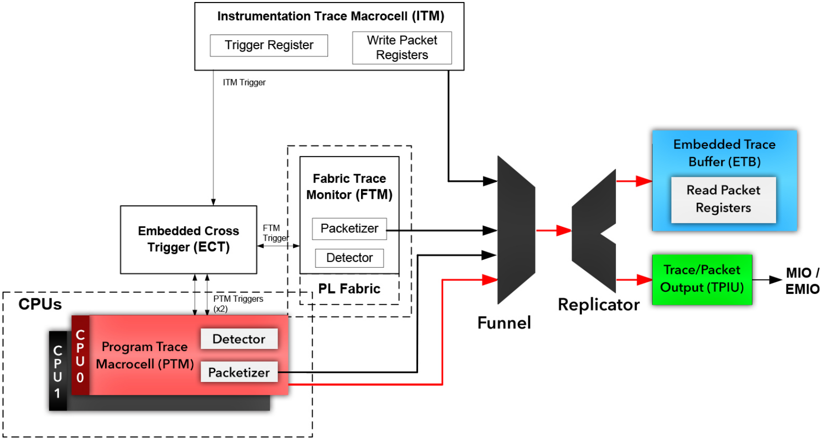


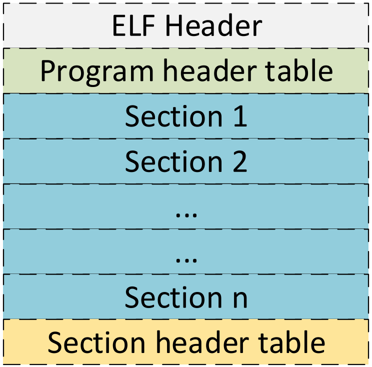
FIGURE – ARM Coresight components in Zynq SoC

Source : ARM CoreSight components TRM

Coresight PTM

Features

- **Trace Filter** (all code or regions of code)



Coresight PTM

Features

- **Trace Filter** (all code or regions of code)
- **Branch Broadcast**[4]

```
(i)   MOV PC, LR
(ii)  ADD R1, R2, R3
(iii) B 0x8084
```

4. Linux driver for PTM patched to support Branch broadcast feature. Link of the commit on the Github page

Coresight PTM

Features

- **Trace Filter** (all code or regions of code)
- **Branch Broadcast**[4]
- Context ID comparator
- CycleAccurate tracing
- Timestamping

```
(i)   MOV PC, LR
(ii)  ADD R1, R2, R3
(iii) B 0x8084
```

4. Linux driver for PTM patched to support Branch broadcast feature. Link of the commit on the Github page

Example Trace

Source code

```
int i;
for (i = 0; i < 10; i++)
```

Example Trace

Source code

```
int i;
for (i = 0; i < 10; i++)
```

Assembly

```
8638 for_loop:
...
b 8654 :
...
866c : bcc 8654
```

Example Trace

Source code

```
int i;
for (i = 0; i < 10; i++)
```

Assembly

```
8638 for_loop:
...
b 8654 :
...
866c : bcc 8654
```

Trace

```
00 00 00 00 00 80 08 38 86 00 00 21
2a 2a 2a 2a 2a 2a 2a 2a 2a 2a 86 01
00 00 00 00 00 00 00 00
```

Example Trace

Source code

```
int i;
for (i = 0; i < 10; i++)
```

Assembly

```
8638 for_loop:
...
b 8654 :
...
866c : bcc 8654
```

Trace

00 00 00 00 00 80 08 38 86 00 00 21 2a 2a 2a 2a 2a 2a 2a 2a 2a 2a 86 01 00 00 00 00 00 00 00 00

Decoded Trace

A-sync
Address 00008638, (I-sync Context 00000000, IB 21)
Address 00008654, Branch Address packet (x 10)

Example Trace

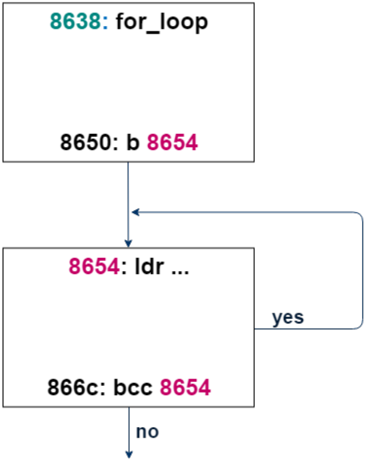


FIGURE – Control Flow Graph

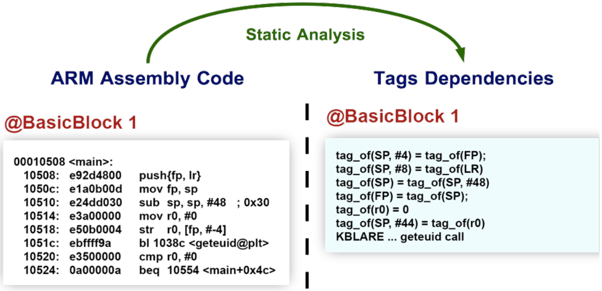
Decoded Trace

A-sync

Address 00008638, (I-sync Context 00000000, IB 21)

Address 00008654, Branch Address packet (x 10)

Static Analysis - Tag dependencies



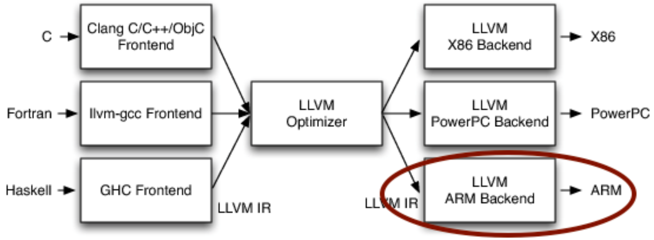
```
00010508 <main>:
  10508:  e92d4800    push{fp, lr}
  1050c:  e1a0b00d    mov fp, sp
  10510:  e24dd030    sub  sp, sp, #48   ; 0x30
  10514:  e3a00000    mov r0, #0
  10518:  e50b0004    str    r0, [fp, #-4]
  1051c:  ebffff9a    bl 1038c <geteuid@plt>
  10520:  e3500000    cmp r0, #0
  10524:  0a00000a    beq  10554 <main+0x4c>
```

```
tag_of(SP, #4) = tag_of(FP);
tag_of(SP, #8) = tag_of(LR)
tag_of(SP) = tag_of(SP, #48)
tag_of(FP) = tag_of(SP);
tag_of(r0) = 0
tag_of(SP, #44) = tag_of(r0)
KBLARE ... geteuid call
```

`ADD R0, R1, R2` $\underline{\texttt{R0}} \leftarrow \underline{\texttt{R1}}$ OR $\underline{\texttt{R2}}$

Static Analysis

- LLVM
- Language-agnostic
- Low-level instructions



Instrumentation

Recover **memory addresses**

Instruction	**Tag dependencies**
ldr r1, [r2, #4]	$\underline{\text{r1}} \leftarrow \underline{\text{ mem (r2 + 4)}}$

Two possible strategies

1. Recover all memory address through instrumentation
2. Recover only register-relative memory address through instrumentation

Instrumentation strategy 1

TABLE – Example tag dependencies instructions

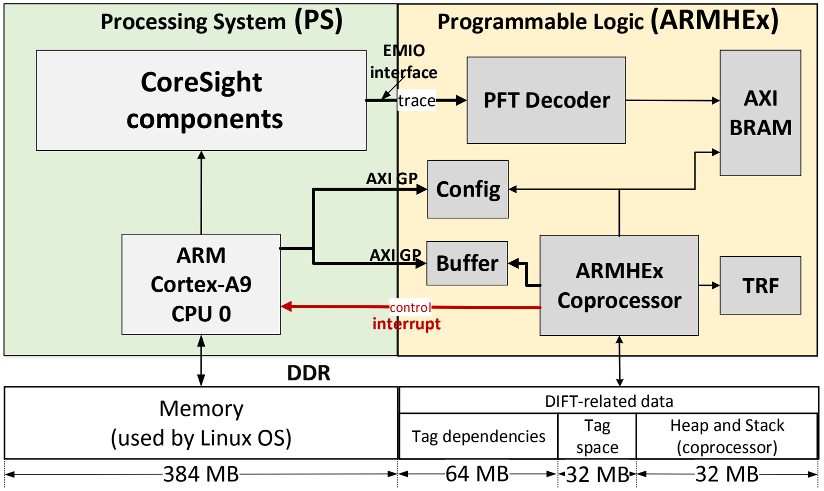
Example Instructions	Tag dependencies	Memory address recovery
`sub r0, r1, r2`	$\underline{\texttt{r0}} = \underline{\texttt{r1}} + \underline{\texttt{r2}}$	
`mov r3, r0`	$\underline{\texttt{r3}} = \underline{\texttt{r0}}$	
`str r1, [PC, #4]`	$\underline{\texttt{@Mem(PC+4)}} = \underline{\texttt{r1}}$	instrumented
`ldr r3, [SP, #-8]`	$\underline{\texttt{r3}} = \underline{\texttt{@Mem(SP-8)}}$	instrumented
`str r1, [r3, r2]`	$\underline{\texttt{@Mem(r3+r2)}} = \underline{\texttt{r1}}$	instrumented

Instrumentation strategy 2

TABLE – Example tag dependencies instructions

Example Instructions	**Tag dependencies**	**Memory address recovery**
`sub r0, r1, r2`	$\underline{\texttt{r0}} = \underline{\texttt{r1}} + \underline{\texttt{r2}}$	
`mov r3, r0`	$\underline{\texttt{r3}} = \underline{\texttt{r0}}$	
`str r1, [PC, #4]`	$\underline{\texttt{@Mem(PC+4)}} = \underline{\texttt{r1}}$	CoreSight PTM
`ldr r3, [SP, #-8]`	$\underline{\texttt{r3}} = \underline{\texttt{@Mem(SP-8)}}$	Static analysis
`str r1, [r3, r2]`	$\underline{\texttt{@Mem(r3+r2)}} = \underline{\texttt{r1}}$	instrumented

Overall architecture

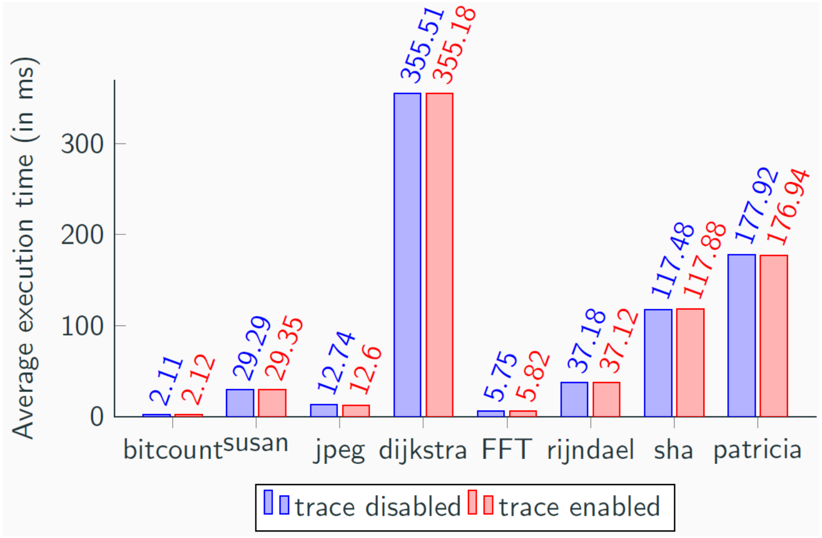


Communication overhead

Goal : Reduce overhead of software instrumentation

- CoreSight PTM
- Static analysis $\rightarrow$ No execution time overhead
- Instrumentation
  - Strategy 1
  - Strategy 2

CoreSight components - Performance overhead



CoreSight components - Performance overhead

- Negligible runtime overhead
  1. PTM non-intrusive (dedicated HW module that works in parallel)
  2. Configuration of CoreSight components (TPIU used [5])
- Communication overhead is only due to instrumentation

5. Linux driver for TPIU has been patched

Instrumentation time overhead

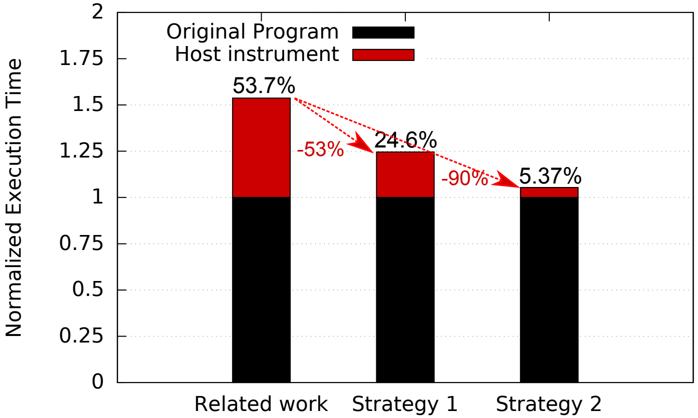


FIGURE – Average execution time of MiBench benchmark for different strategies

Instrumentation time overhead

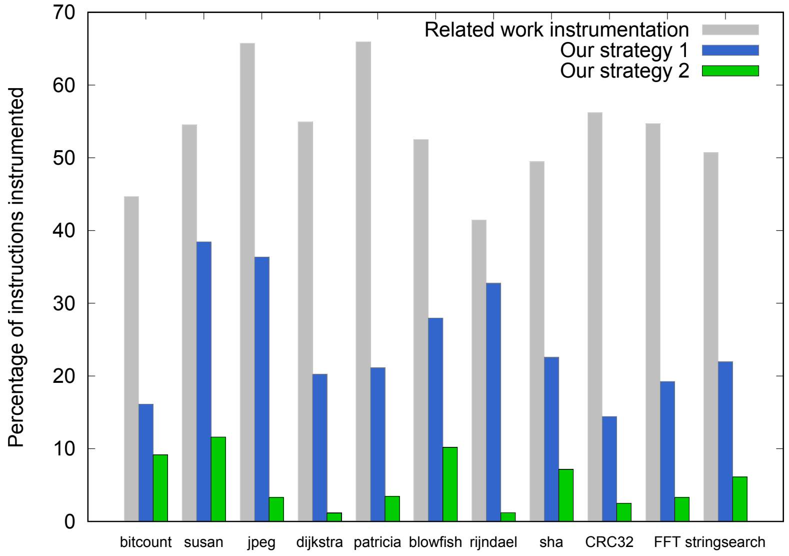
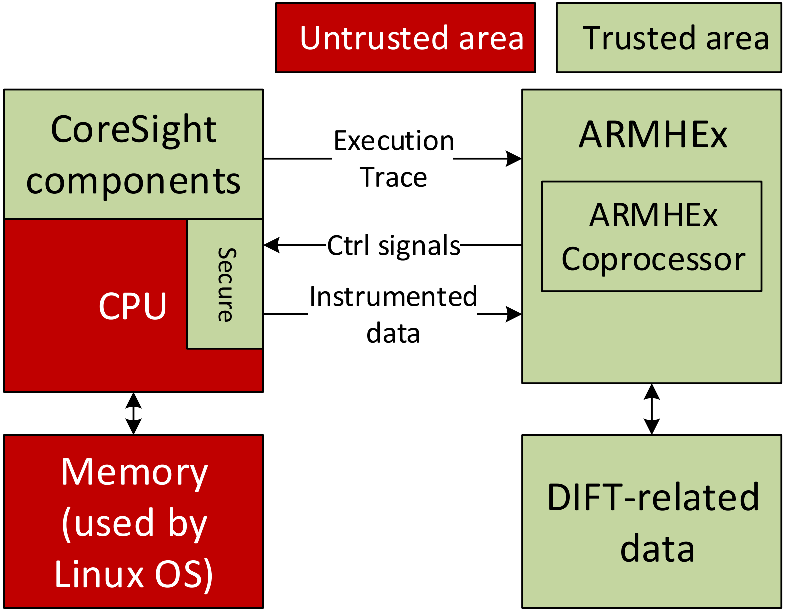
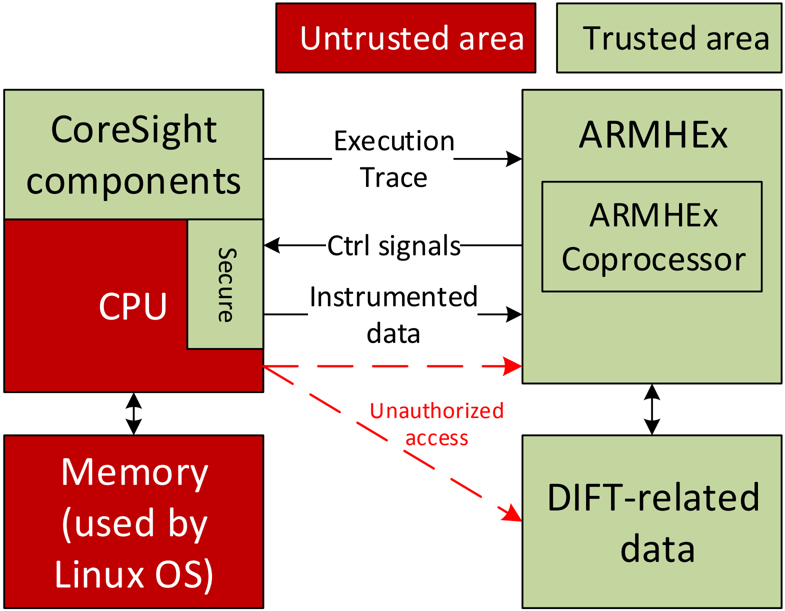


FIGURE – Number of instrumented instructions

DIFT coprocessor security with ARM TrustZone



DIFT coprocessor security with ARM TrustZone



Comparison with related work

TABLE – Performance comparison with related work

Approaches	Kannan	Deng	Heo	ARMHEx
Hardcore portability	No	No	**Yes**	**Yes**
Main CPU	Softcore	Softcore	Softcore	**Hardcore**
Communication overhead	N/A	N/A	60%	**5.4%**
Area overhead	6.4%	14.8%	14.47%	**0.47%**
Area (Gate Counts)	N/A	N/A	256177	**128496**
Power overhead	N/A	**6.3%**	24%	16%
Max frequency	N/A	**256 MHz**	N/A	250 MHz
Isolation	No	No	No	**Yes**

Conclusion

Conclusion

Take away

- CoreSight PTM allows to obtain runtime information (Program Flow)
- Non-intrusive tracing $\rightarrow$ Negligible performance overhead
- Reduced communication time overhead
- Improve software security

Conclusion

Take away

- CoreSight PTM allows to obtain runtime information (Program Flow)
- Non-intrusive tracing → Negligible performance overhead
- Reduced communication time overhead
- Improve software security

Future perspectives

- Combine Low-level and OS-level DIFT
- Extend DIFT on multicore CPU
- Take use of other debug components for security
  - Intel Processor Trace
  - STM (TI)

Trust Zone is not enough

Pascal & Muhammad
Dezember 30, Leipzig #34C3

https://github.com/pcotret/34c3-trustzone-is-not-enough

Many thanks to:
Muhammad Abdul Wahab (IETR, FR)
Mounir Nasr Allah (INRIA CIDRE, FR)
Guillaume Hiet (INRIA CIDRE, FR)
Vianney Lapôtre (UBS, FR)
Guy Gogniat (UBS, FR)