"Les Vieux bRISCards" final report from Rennes

3rd national RISC-V student contest - 2022-2023

Authors :
Majed Abdennadher
Thomas Chevalier
Ulysse Vincenti

Professor :
Guillaume Hiet
Rubén Salvador

Contents

Rennes, April 14, 2023

Abstract : The 5rd national RISC-V student contest is a hackaton sponsored by Thales, the GDR SOC^2 and the CNFM whose goal is to improve the security of the CV32A6 RISC-V soft-core. The task is to defeat ten memory-corruption attacks taken from the RIPE benchmark with minimal performance overhead. The soft-core is geared with Zephyr RTOS, and the contest allows us to modify the CV32A6 core, the OS and/or the compiler. Our solution consists of a forward edge control-flow integrity (FW-CFI) enforcement technique, a shadow stack and a hardened version of memset that protects against heap pointer read or write overflow. Our implementation modifies the CV32A6 core in order to add support for hardware CFI checks and the shadow stack. We also modify the compiler for automatic generation of CFI annotations, as well as the OS and the linked libraries.

The objective of this challenge is to counter a maximum number of attacks in a batch of 10 attacks from the RIPE benchmark ported to RISC-V CVA6-32. Numerous of theses attacks are ways to take control of the control flow. With the control of the flow, the attacker can realise **arbitrary code execution**. It is therefore, of the greatest importance to keep the control of the flow. Multiple countermeasure exist for theses attacks and we based our solution on one of them : *Intel Control-flow Enforcement***intel_cet** (CFE).

Our solution, is a forward edge control-flow integrity (FW-CFI) enforcement technic, a shadow stack(SS) and a memset heap protection. We have tested multiple solution before going for this one. With this implementation we are able to **counter all the attacks given** with no apparent timing overhead.

I Introduction

The national RISC-V student contest is open to all Master 2 students of a French engineering school or University. Each team can count up to 4 members and can be coached by teachers or PhD students. For the last 3 years, this contest has proposed hardware challenges to the students, the first year was focused on the FPGA optimisation of the core, the second year on the power efficiency and finally this year pays attention to the security of the core. A benchmark of 10 attacks is given to the participants, the goal is to prevent the attacks on a Zephyr running upon the CV32A6. The chosen board is a Zybo-z7 20 which embeds an Artix-7 FPGA. The implementation should not increase the size of the FPGA design or the execution time by more than 100%. The teams are judged on the number of attacks they are able to counter and in the case of a tie, on the execution time of a performance benchmark.

The core CV32A6 is an open source 6 stage RISC-V 32 bit processor. It implements I, M and C extensions from the RISC-V instruction set **riscv_isa**. The goal of this core is to run an OS at reasonable speed. The chosen core does not have a MMU or PMP but theses features are available on more recent version of the core.

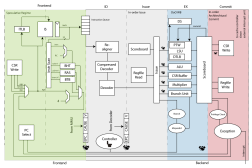

Figure 1: CVA6 core diagram **cva6_doc**

Zephyr is an embedded real-time OS which supports a wide variety of microprocessor/microcontrollers architectures. Zephyr embeds multiple services and features like PMP support, single address space and stack canaries. The OS uses Kconfig and device tree for its configuration and is compiled through cmake with gcc-12.

Runtime Intrusion Prevention Evaluator (Ripe) ripe_attack is a security benchark which contains 850 executable simulated attacks. The goal of this bench is to evaluate solutions against buffer overflows attacks. The benchmark was recently ported to RISC-V and a selection of 10 of theses attacks makes our evaluation benchmark.

I.I Related work

i	approaches	inject param	code ptr	location	function
1	direct	no nop	ret	stack	memcpy
2	direct	no nop	functptrstackvar	stack	memcpy
3	indirect	no nop	functptrstackvar	stack	memcpy
4	direct	data	var leak	heap	sprintf
5	direct	ret2libc	ret	stack	memcpy
6	indirect	ret2libc	funcptrheapvar	heap	memcpy
7	indirect	ret2libc	structfunctptrheap	heap	homebrew
8	indirect	ret2libc	longjumpbugheap	heap	memcpy
9	direct	rop	ret	stack	memcpy
10	direct	rop	structfuncptrheap	heap	memcpy

Table 1: Table presenting the different supposedly simulated attacks on the benchmark

Multiple methods exist for the control flow enforcement. Some of the first ideas that could come to mind would be ASLR. However, this solution is sub optimal because the chosen core is 32 bits. As a side note, the RIPE benchmark takes the form of a single binary that attacks itself and thus defeat any countermeasures that relies on the secrecy of memory locations.

II Implementation

II.I Stack canaries

The stack canaries are a simple tool to implement with Zephyr because it only demands to add 2 lines in the configuration file of the board. Stack canaries

are secret pseudo random value placed on the stack to check the integrity of this memory space. If stack smashing occurs, the canary might be overwritten and the smashing detected.

```
CONFIG_TEST_RANDOM_GENERATOR=y
CONFIG_STACK_CANARIES=y
```

The first line is here to add a pseudo random source and the second one is to add the stack canary.

II.II Simple FW & BW control flow

The development of forward edge and backward edge control flow enforcement has been thought of with 2 custom instructions : the first one as a landing pad (LP) after a call : "*li zero, 2*", this is the forward edge control flow. The second instruction is the LP which would be placed after a call so that when the call returns, the first instruction executed is this instruction : "*li zero, 1*".

The result in assembly is :

```
<z_bss_zero>:
li      zero,2
addi    sp,sp,-16
sw      ra,12(sp)
lui     a0,0x8000b
addi    a2,a0,-336 # 8000aeb0
lui     a5,0x8000b
addi    a5,a5,980 # 8000b3d4
sub     a2,a5,a2
li      a1,0
addi    a0,a0,-336
jal     ra,800076e0 # call <z_early_memset>
li      zero,1
lw      ra,12(sp)
addi    sp,sp,16
ret
```

Listing 1: Example of "z_bss_zero" function compiled with the pads

The instruction "nop" is in reality : "*add zero, 0*", that is why we chose the solution of the index 1 and 2. This technique has been quite efficient. Added to the stack canaries we were able to break :

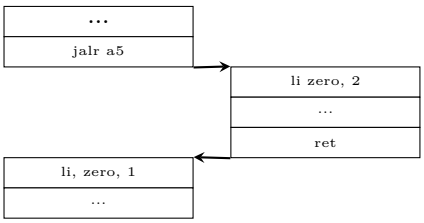

Figure 2: Control Flow Diagram : simple implementation

In a second time, we thought about identifying the function where indirect calls could go. This was achieved by adding some information in the first LP. The first 2 bits of the immediate was kept as it was. The last bit was taken by isVariadic information and the rest of the immediate became the number of arguments.

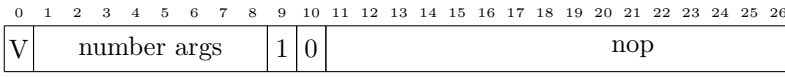

Figure 3: Landing Pad for the FW-CFI

- V : the variadic bit.

But, to check whether or not the call is made with the right number of arguments, the hardware must be aware of this information, this is done by writing this information to a custom CSR.

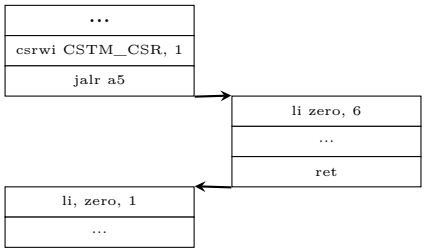

Figure 4: Control Flow Diagram : indirect call

II.III Forward CFI + Shadow Stack

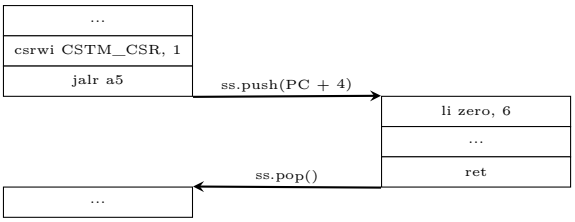

Figure 5: Control Flow Diagram : indirect call with shadow stack

The shadow stack allows to remove the second LP. A shadow stack is a hardware or software security feature which stores the return addresses of a call in a LIFO secured space. During a return, the return address is checked against the shadow stack return address. If the return address in the shadow stack and the return address in the current flow are not the same, their might be an attack occurring.

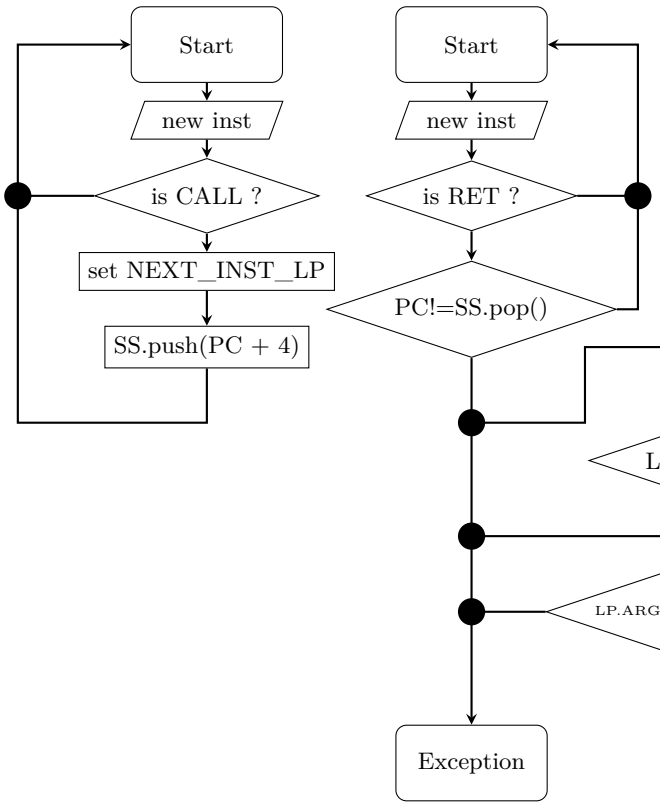

Figure 6: Simplified hardware algorithm implemented in the commit stage

This scheme was simplified by supposing that their is only one commit per clock cycle done. In reality, we can have up to 2 commits done in one clock cycle, which complexify things.

The hardware modifications were done with a atypical debug method. For the first half of the competition we did not have access to the simulator. The ILA components did not seem to want to work, therefore, we developed the first elements with the outputs linked to the 9 controllable LEDs of the board. Once the simulator was up and running we did already have a working solution which only needed a verification by the simulator.

To get better results in the FPGA placement, the SS has a depth of only 100. When the count of calls is greater than 100, the SS is deactivated but the previous return addresses are stored. The SS starts to compare addresses again when counter goes under 100.

The previously shown algorithm is implemented through 2 Finite State Machines (FSM) and 2 modules, the first for the detection and FSM, the second for the LIFO logic.

II.IV Heap "memset" Protection

II.V Optimisation

The first version of the compiled library showed a great lack of optimisation, multiples solutions were applicable to solve that :

1. add an FPU to the core because the performance benchmark is doing floating point operations.
2. reduce the time consumed by the core on nop execution through prematured nop execution. The custom nop could bypass the pipeline from the fetch phase which would remove the timing overhead of the solution.
3. increase the frequency of the core.
4. temper the cycle count CSR.

III Results

III.I Performances

```
*** Booting Zephyr OS build zephyr-v3.2.0-327-
    g869365ab012b  ***
Begining of execution with depth 12, call
    number 50, seed value 63728127.000000
SUCCESS: computed value 868200.000000 -
    duration: 25.317176 sec 632929412 cycles
```

Listing 2: Results of the benchmark on the non-modified hardware, OS and compiler

```
*** Booting Zephyr OS build zephyr-v3.2.0-327-
    g869365ab012b  ***
Begining of execution with depth 12, call
    number 50, seed value 63728127.000000
SUCCESS: computed value 868200.000000 -
    duration: 25.135623 sec 628390584 cycles
```

Listing 3: Results of the benchmark on the FW CFI + shadow stack + secure memset HW, OS and compiler

	number of cycles	t
before modifications	632929412	
stack canaries	633811376	
FW & BW CFI	628152061	
FW CFI + shadow stack	628152061	
FW CFI + shadow stack + secure memset	628390584	

Table 2: Results from the benchmark perf_baseline before and after the modifications

resources	utilisation after modification	utilisation be
LUT	27161 (51.054512%)	25765 (
FF	26056 (24.488722%)	22328
BRAM	71 (50.714283%)	71 (50
clock speed	25MHz	25
perf_baseline size	86828 B	61

Table 3: Table of modification's impact

III.II Attacks prevented

index	stack canaries	FW&BW CFI	FW CFI + shadow	FW CFI +
1	✅	✅	✅	
2	✅	✅	✅	
3	✅	✅	✅	
4	❌	❌	❌	
5	✅	✅	✅	
6	❌	✅	✅	
7	❌	✅	✅	
8	❌	✅	✅	
9	✅	✅	✅	
10	❌	✅	✅	✅

Table 4: Table presenting the different prevented attacks with different approaches

IV Conclusion

V Annexe

```
*** Booting Zephyr OS build ***
RIPE is alive! cv32a6_zybo
RIPE parameters:
technique direct
inject param returnintolibc
code pointer ret
location stack
function memcpy
-----------------
Shellcode instructions:
lui t1, 0x80004
        80004337
addi t1, t1, 0x608
        60830313
jalr t1
        000300e7
-----------------
target_addr == 0x8001124c
buffer == 0x80010cd0
payload size == 1409
bytes to pad: 1404

overflow_ptr: 0x80004630
payload:
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Executing attack...
```

```
*** Booting Zephyr OS build ***
RIPE is alive! cv32a6_zybo
RIPE parameters:
technique indirect
inject param returnintolibc
code pointer funcptrheap
location heap
function memcpy
-----------------
Shellcode instructions:
lui t1, 0x80004
        80004337
addi t1, t1, 0x608
        60830313
jalr t1
        000300e7
-----------------
target_addr == 0x80013370
buffer == 0x80013158
payload size == 541
bytes to pad: 536

overflow_ptr: 0x80013488
payload:
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

bytes to pad: 272

overflow_ptr: 0x80004630
payload:
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

target_addr_aux: 0x80013268

Executing attack...
```

```
*** Booting Zephyr OS build ***
RIPE is alive! cv32a6_zybo
RIPE parameters:
technique direct
inject param rop
code pointer ret
location stack
function memcpy
-----------------
Shellcode instructions:
lui t1, 0x80004
        80004337
addi t1, t1, 0x608
        60830313
jalr t1
        000300e7
-----------------
target_addr == 0x8001124c
buffer == 0x80010cd0
payload size == 1409
bytes to pad: 1404

overflow_ptr: 0x800046cc
payload:
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Executing attack...
```

```
*** Booting Zephyr OS build ***
RIPE is alive! cv32a6_zybo
RIPE parameters:
technique direct
inject param shellcode
code pointer ret
location stack
function memcpy
-----------------
Shellcode instructions:
lui t1, 0x80004
        80004337
addi t1, t1, 0x608
        60830313
jalr t1
        000300e7
-----------------
target_addr == 0x8001124c
buffer == 0x80010cd0
payload size == 1409
bytes to pad: 1392

overflow_ptr: 0x80010cd0
payload: 7C

Executing attack...
```

```
*** Booting Zephyr OS build ***
RIPE is alive! cv32a6_zybo
RIPE parameters:
technique direct
inject param shellcode
code pointer funcptrstackvar
location stack
function memcpy
-----------------
Shellcode instructions:
lui t1, 0x80004
        80004337
addi t1, t1, 0x608
        60830313
jalr t1
        000300e7
-----------------
target_addr == 0x800110fc
buffer == 0x80010cd0
payload size == 1073
bytes to pad: 1056

overflow_ptr: 0x80010cd0
payload: 7C

Executing attack...
```

```
*** Booting Zephyr OS build ***
RIPE is alive! cv32a6_zybo
RIPE parameters:
technique indirect
inject param shellcode
code pointer funcptrstackvar
location stack
function memcpy
-----------------
Shellcode instructions:
lui t1, 0x80004
        80004337
addi t1, t1, 0x608
        60830313
jalr t1
        000300e7
-----------------
target_addr == 0x800110f8
buffer == 0x80010cd0
payload size == 1069
bytes to pad: 1052

overflow_ptr: 0x800110fc
payload: 7C

Executing attack...
```