Bridging Software-Based and Hardware-Based Fault Injection Attacks

Nisrine Jafri

Journée Injection de Fautes
Paris

29-05-2018

Fault injection

- Fault can occur due to:
  - Environment.
  - Attack.
- Fault injection vulnerabilities:
  - Create vulnerabilities in software.

Fault Injection Methods

- Software-Based Fault injection (SBFI):
  - Reproduce at software level the effect of fault.
  - Simulation.
- Hardware-Based Fault injection (HBFI):
  - Disturb the hardware at physical level.
  - Laser, EMP…

SBFI: Pros & Cons

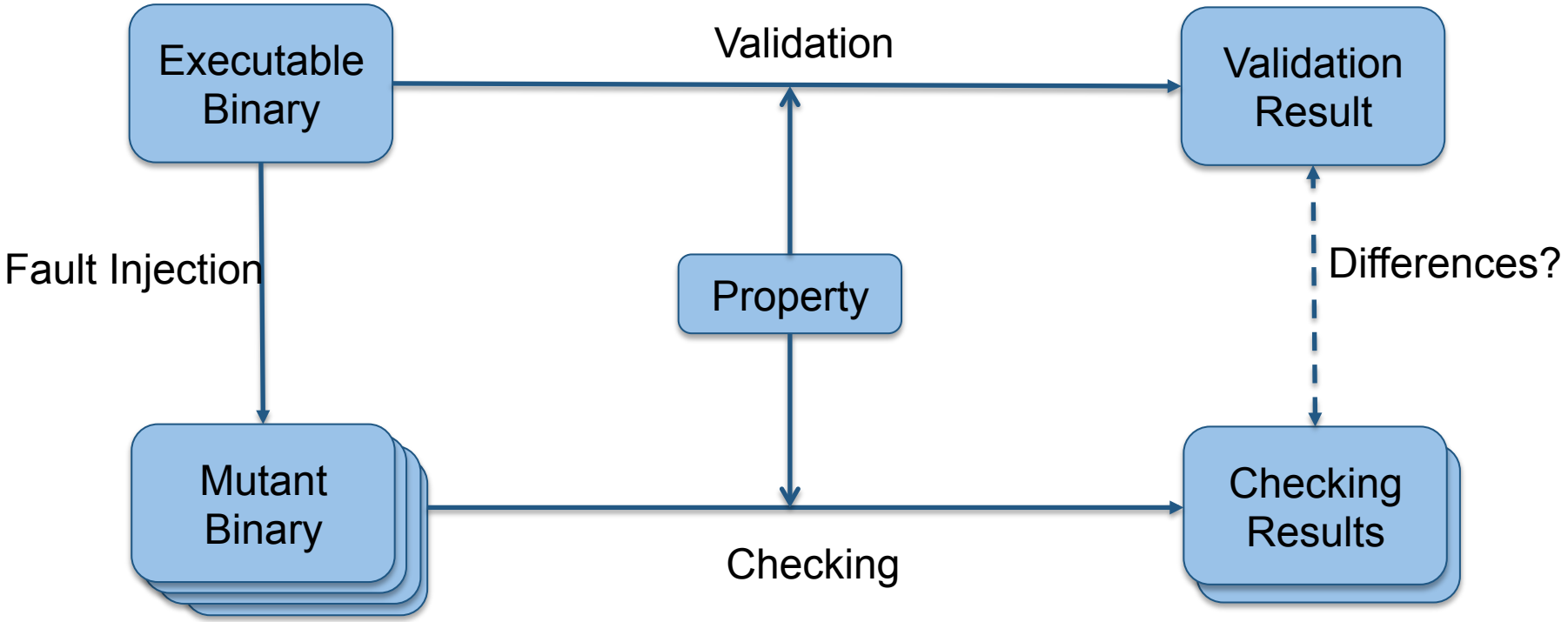
- Advantage :
  - Cheap.
  - Fast.
- Disadvantage :
  - Vulnerabilities not guarantied.

HBFI: Pros & Cons

- Advantage :
  - Real Vulnerabilities.
- Disadvantage :
  - Expensive.
  - Time consuming.

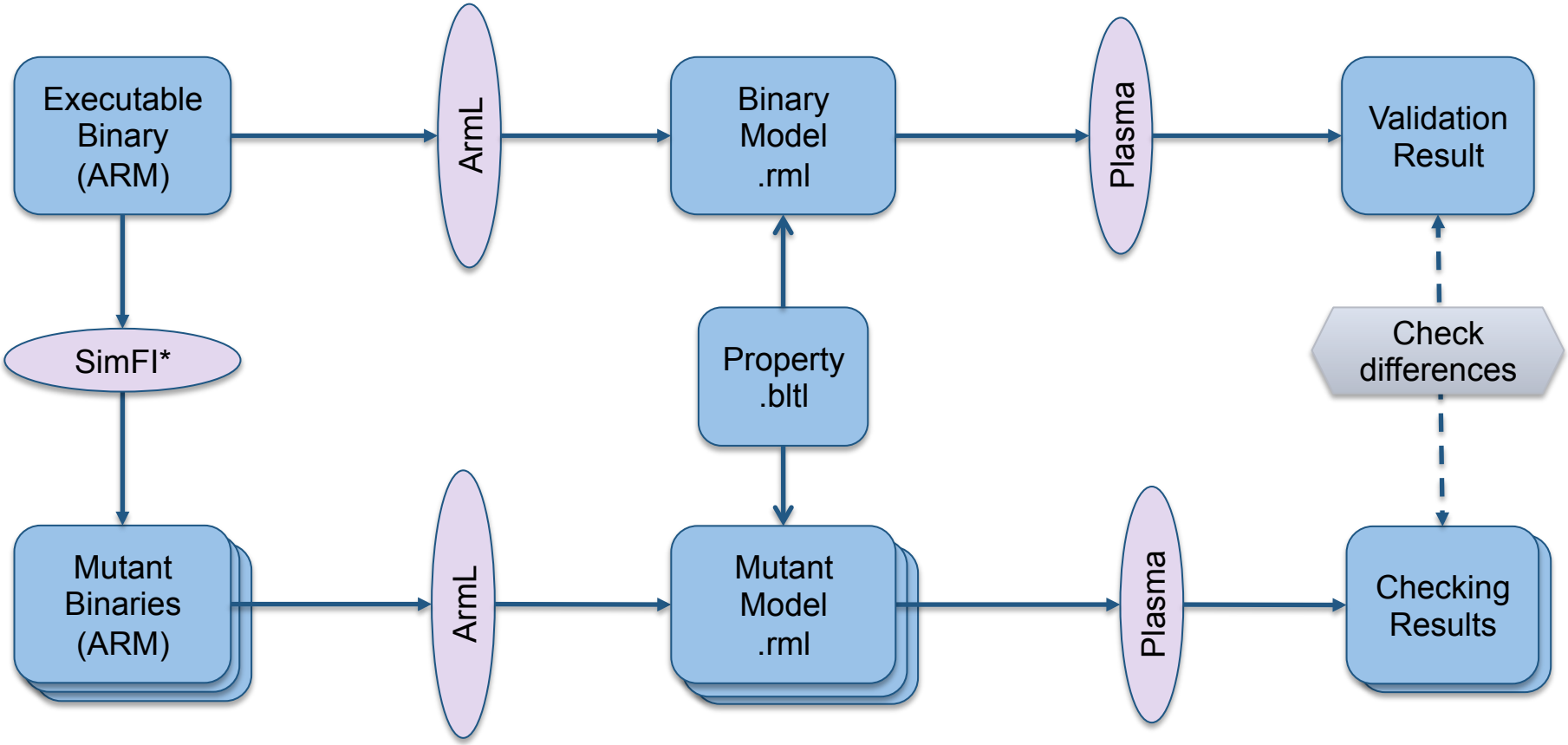
SBFI : Methodology

Automated formal process for detecting fault injection vulnerabilities [1]



[1] Given-Wilson Thomas, Jafri Nisrine, Lanet Jean-Louis, Legay Axel. **An Automated Formal Process for Detecting Fault Injection Vulnerabilities in Binaries and Case Study on PRESENT**. In: :293–300IEEE; 2017.

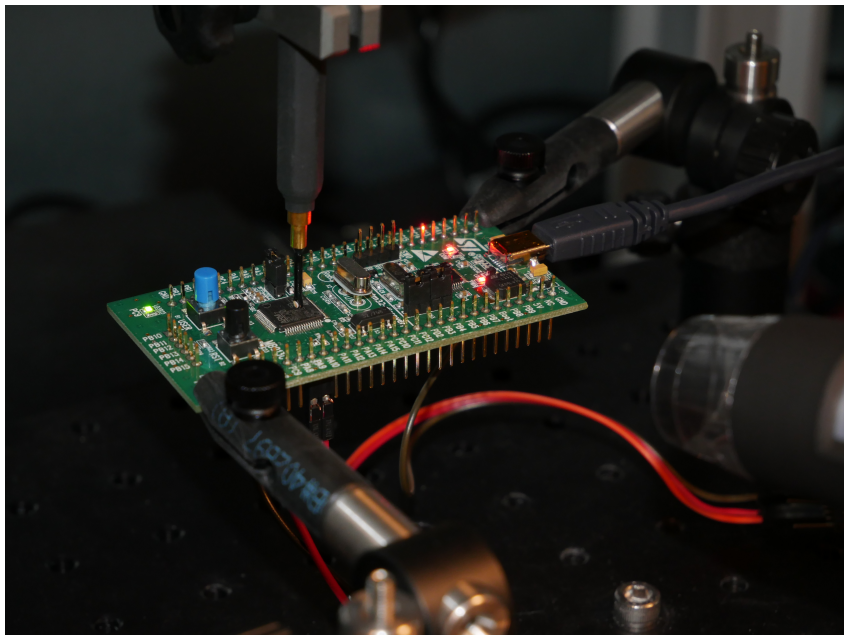
SBFI : Implementation



* Simulation Fault Injection (SimFI) Tool: https://github.com/nisrine/Fault-Injection-Tool

HBFI : Methodology

- Where ?
  - LHS Lab.
- What ?
  - STM32 chip (ARMv7-M)
- How ?
  - EMP.



by Sebanjila Kevin Bukasa

Case Study 1: Control Flow Hijacking

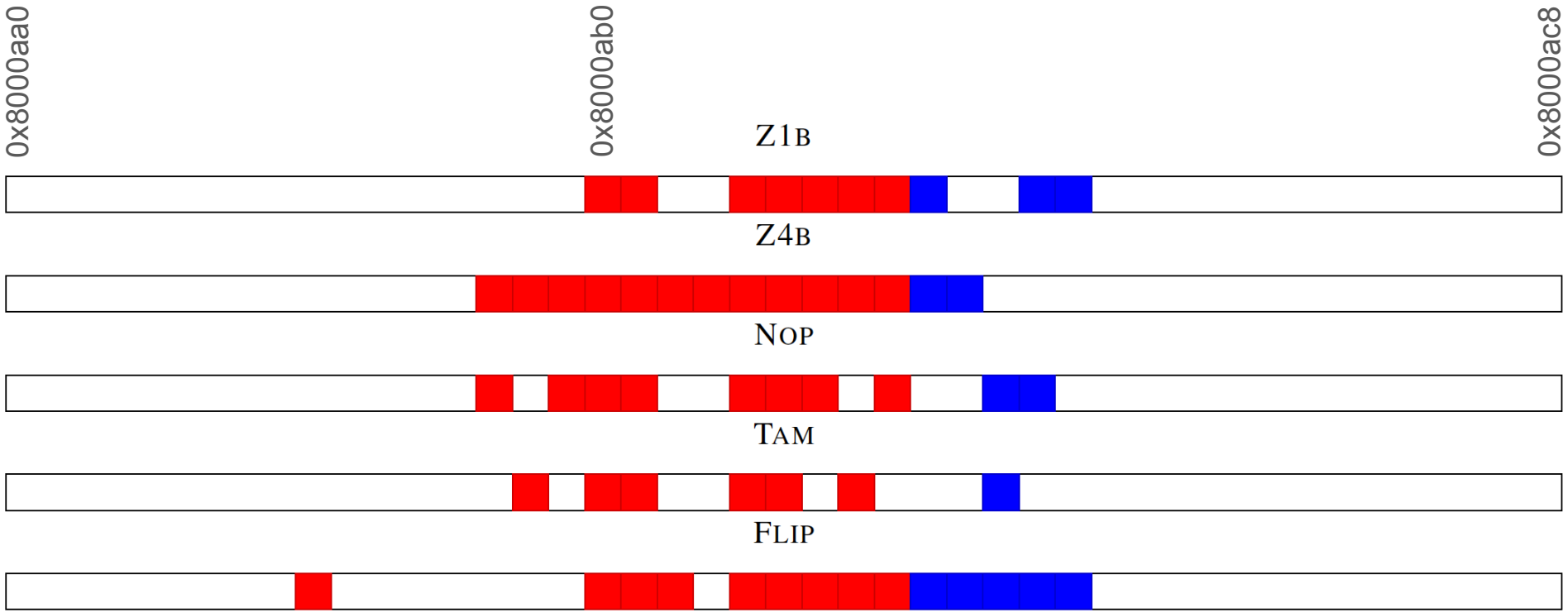
- The control flow hijacking attack consists of modifying the execution flow of a running program.

```
uint32_t test_persistence (void){
   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_SET);
   uint32_t status = 0;
   if (pin_correct==1) {
      status=0xFFFFFFFF;
   } else {
      status=0x55555555;
   }
   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
   return status;
}
```

Case Study 1: Control Flow Hijacking

```
08000aa0 <test_persistence>:
8000aa0: b510 push {r4, lr}
8000aa2: 480a ldr r0, [pc, #40]
8000aa4: 2180 movs r1, #128
8000aa6: 2201 movs r2, #1
8000aa8: f001 f91c bl 8001ce4 <HAL_GPIO_WritePin>
8000aac:   4b08        ldr    r3, [pc, #32]
8000aae:   4807        ldr    r0, [pc, #28]
 8000ab0:  681b        ldr    r3, [r3, #0]
 8000ab2:  2180        movs    r1, #128;  0x80
 8000ab4:  2b01        cmp  r3, #1
 8000ab6:  bf0c        ite    eq
 8000ab8:  f04f 34ff   moveq.w   r4, #4294967295    ; 0xffffffff
 8000abc:  f04f 3455   movne.w   r4, #1431655765    ; 0x55555555
 8000ac0: 2200 movs r2, #0
8000ac2: f001 f90f bl 8001ce4 <HAL_GPIO_WritePin>
8000ac6: 4620 mov r0, r4
8000ac8: bd10 pop {r4, pc}
```

Case Study 1: SBFI Experiment Result



Vulnerability (status=0x55555555)

Incorrect (status !=0xFFFFFFFF && status !=0x5555555)

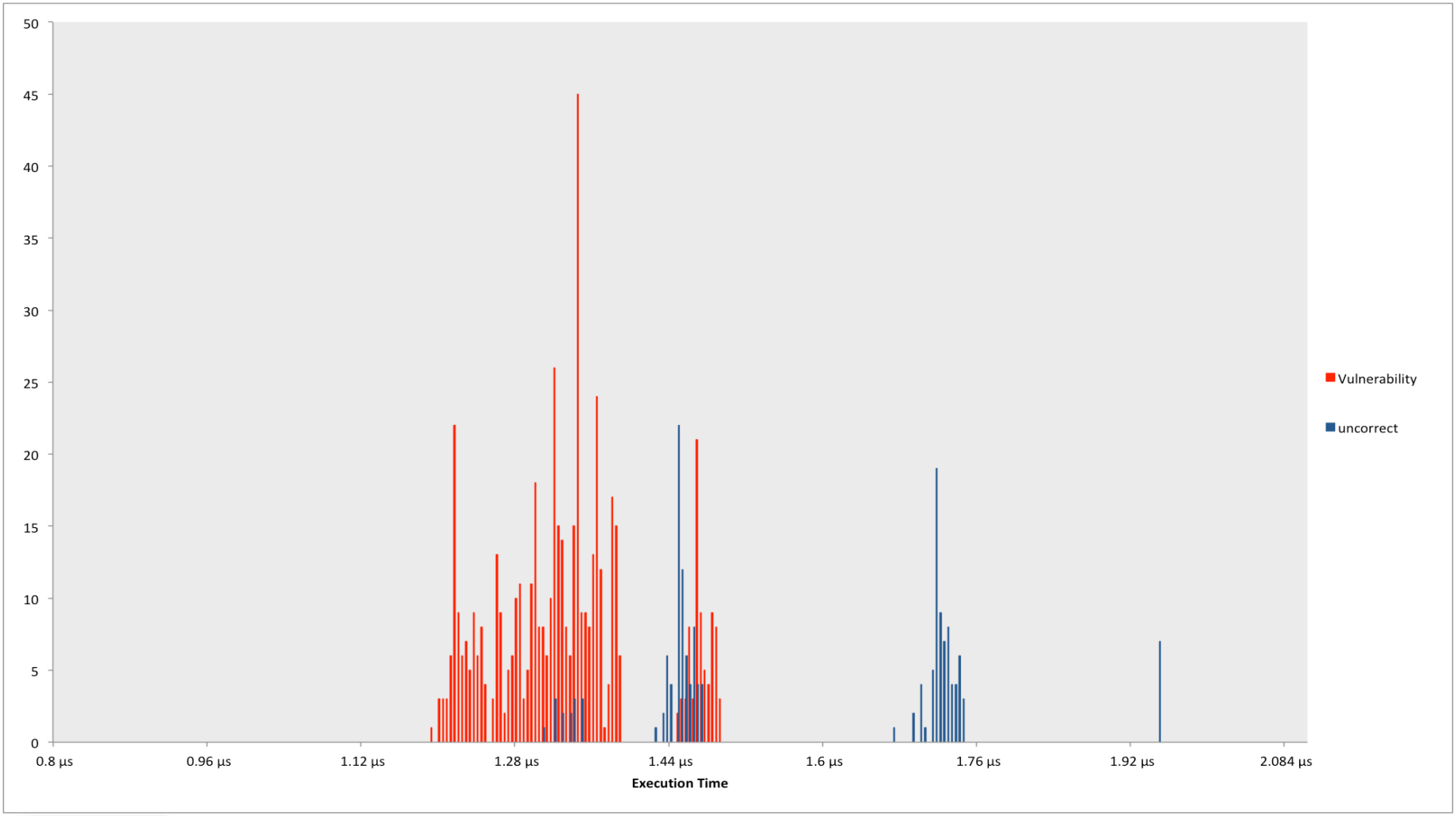
Case Study 1: SBFI Experiment Result

```
08000aa0 <test_persistence>:
8000aa0: b510 push {r4, lr}
8000aa2: 480a ldr r0, [pc, #40]
8000aa4: 2180 movs r1, #128
8000aa6: 2201 movs r2, #1
8000aa8: f001 f91c bl 8001ce4 <HAL_GPIO_WritePin>
8000aac:   4b08        ldr    r3, [pc, #32]
8000aae:   4807        ldr    r0, [pc, #28]
 8000ab0:  681b        ldr    r3, [r3, #0]
 8000ab2:  2180        movs    r1, #128;  0x80
 8000ab4:  2b01        cmp  r3, #1
 8000ab6:  bf0c        ite    eq
 8000ab8:  f04f 34ff   moveq.w   r4, #4294967295    ; 0xffffffff
 8000abc:  f04f 3455   movne.w   r4, #1431655765    ; 0x55555555
 8000ac0: 2200 movs r2, #0
8000ac2: f001 f90f bl 8001ce4 <HAL_GPIO_WritePin>
8000ac6: 4620 mov r0, r4
8000ac8: bd10 pop {r4, pc}
```

- mov r3, r3 (Z1B)
- ldr r3, [r3, #8] (Flip)
- nop (Nop)

- mov r3, #1 (Flip)
- cmp r3, #0 (Z1B)
- nop (Nop)

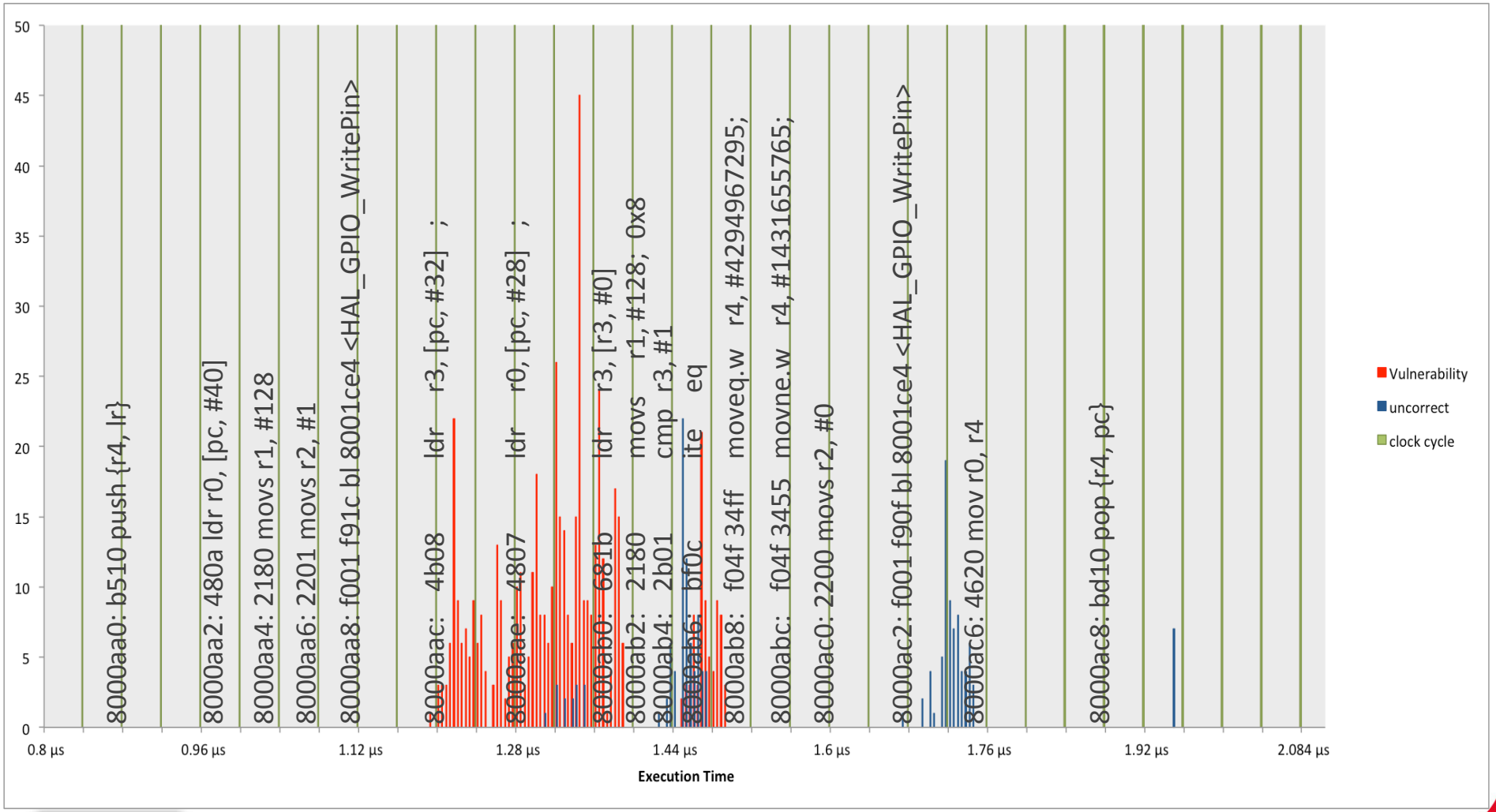
Case Study 1: HBFI Experiment Result



Case Study 1: HBFI Experiment Result

- From the hardware experiments observation and the triggers in the C code:
  - Delay between the the injection and the fault observed effect is 0.08µs to 0.12µs.
- From the ARM Cortex-M3 processor technical reference manual:
  - Instruction executed in order.
  - Number of clock cycle per instruction.
    - Each clock cycle is approximately 0.04µs.

Case Study 1: HBFI Experiment Result



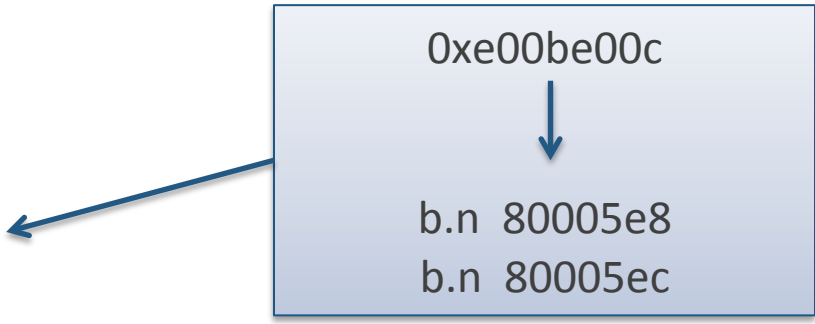
Case Study 2: Backdoor

- The backdoor attack consists in opening a door in the code to access inaccessible code parts.

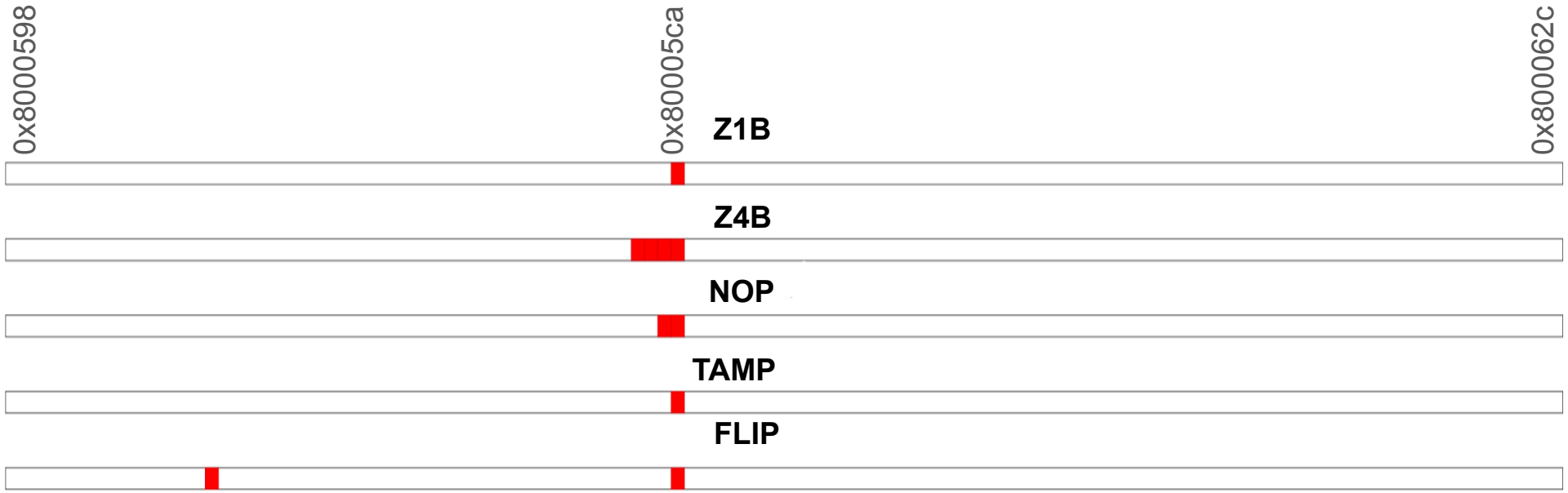
```
void blink_wait(){
   unsigned int wait_for=3758874636;
   unsigned int counter;
   for(counter=0;counter<wait_for;counter+=8000000);
}
void backdoor(void) {
   int i;
   for(i = 0; i < DATA_SIZE; i++){
      ciphertext[i] = key[i];
   }
   HAL_GPIO_WritePin(LED3_GPIO_PORT, LED3_PIN, GPIO_PIN_SET);
}
```

Case Study 2: Backdoor

```
08000598 <blink_wait>:
8000598: b580 push {r7, lr}
800059a: b082 sub sp, #8
...
80005c0: 2003 movs r0, #3
80005c2: f000 f8af bl 8000724 <wait>
80005c6: 3708 adds r7, #8
80005c8: 46bd mov sp, r7
80005ca: bd80 pop {r7, pc}
80005cc: e00be00c .word 0xe00be00c
080005e8 <backdoor>:
 80005e8:  b580        push {r7, lr}
 80005ea:  b082        sub  sp, #8
 80005ec:  af00      add  r7, sp, #0
...
800061a:  f003 f807   bl  800362c <HAL_GPIO_WritePin>
 800061e:  3708        adds r7, #8
 8000620:  46bd        mov  sp, r7
 8000622:  bd80        pop  {r7, pc}
```



Case Study 2: SBFI Experiment Result



Vulnerability (Program Counter inside the backdoor function)

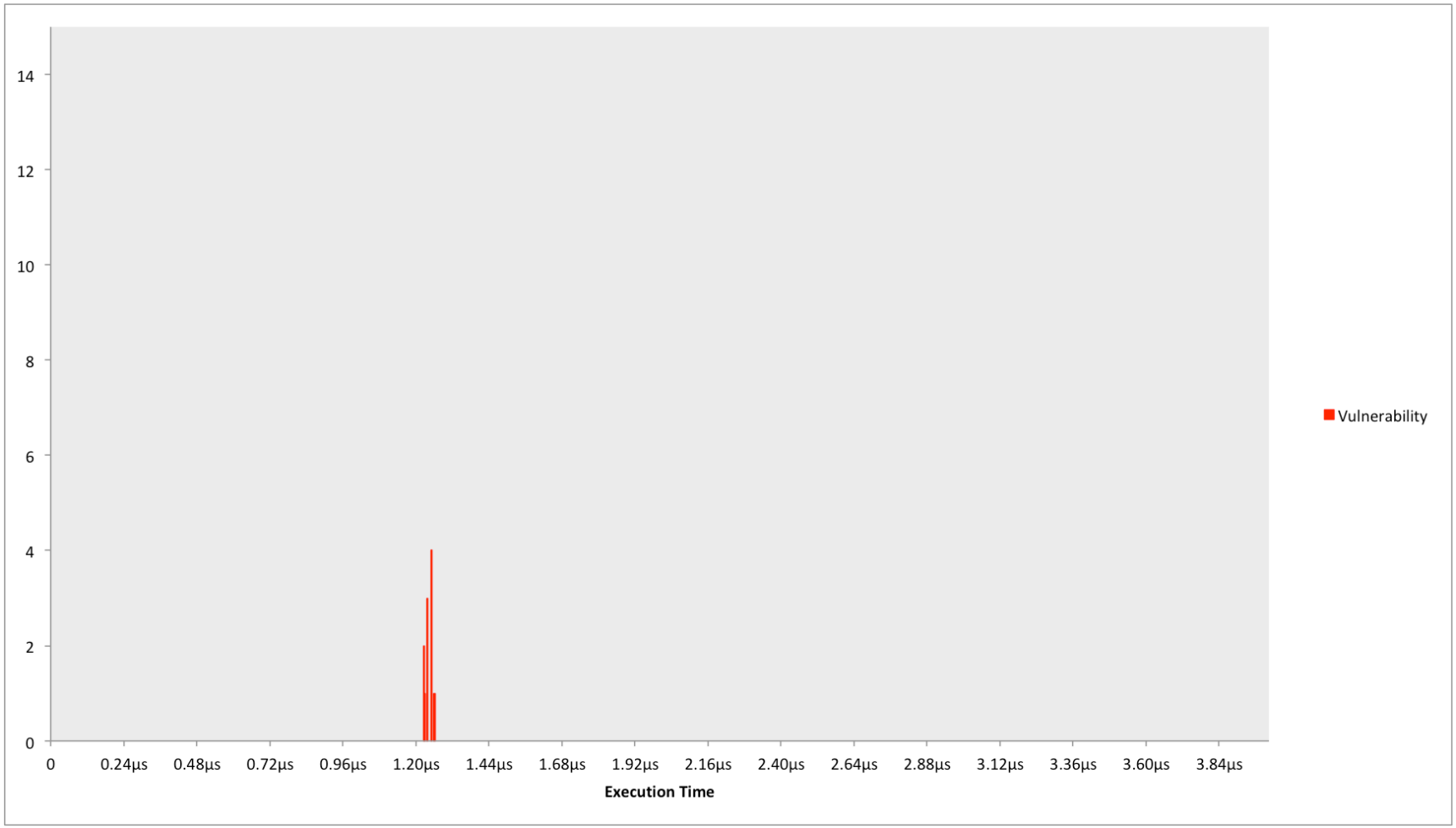
Case Study 2: SBFI Experiment Result

```
08000598 <blink_wait>:
 8000598:  b580 push {r7, lr}
 800059a:  b082 sub sp, #8
...
80005a4:   607b        strr3, [r7, #4]
80005a6:   e005     b.n    80005b4
...
80005c6: 3708 adds r7, #8
80005c8: 46bd mov sp, r7
80005ca: bd80 pop {r7, pc}
80005cc: e00be00c .word 0xe00be00c
080005e8 <backdoor>:
 80005e8:  b580        push {r7, lr}
 80005ea:  b082        sub    sp, #8
 80005ec:  af00      add    r7, sp, #0
...
800061e:   3708        adds r7, #8
 8000620:  46bd        mov  sp, r7
 8000622:  bd80        pop   {r7, pc}
```

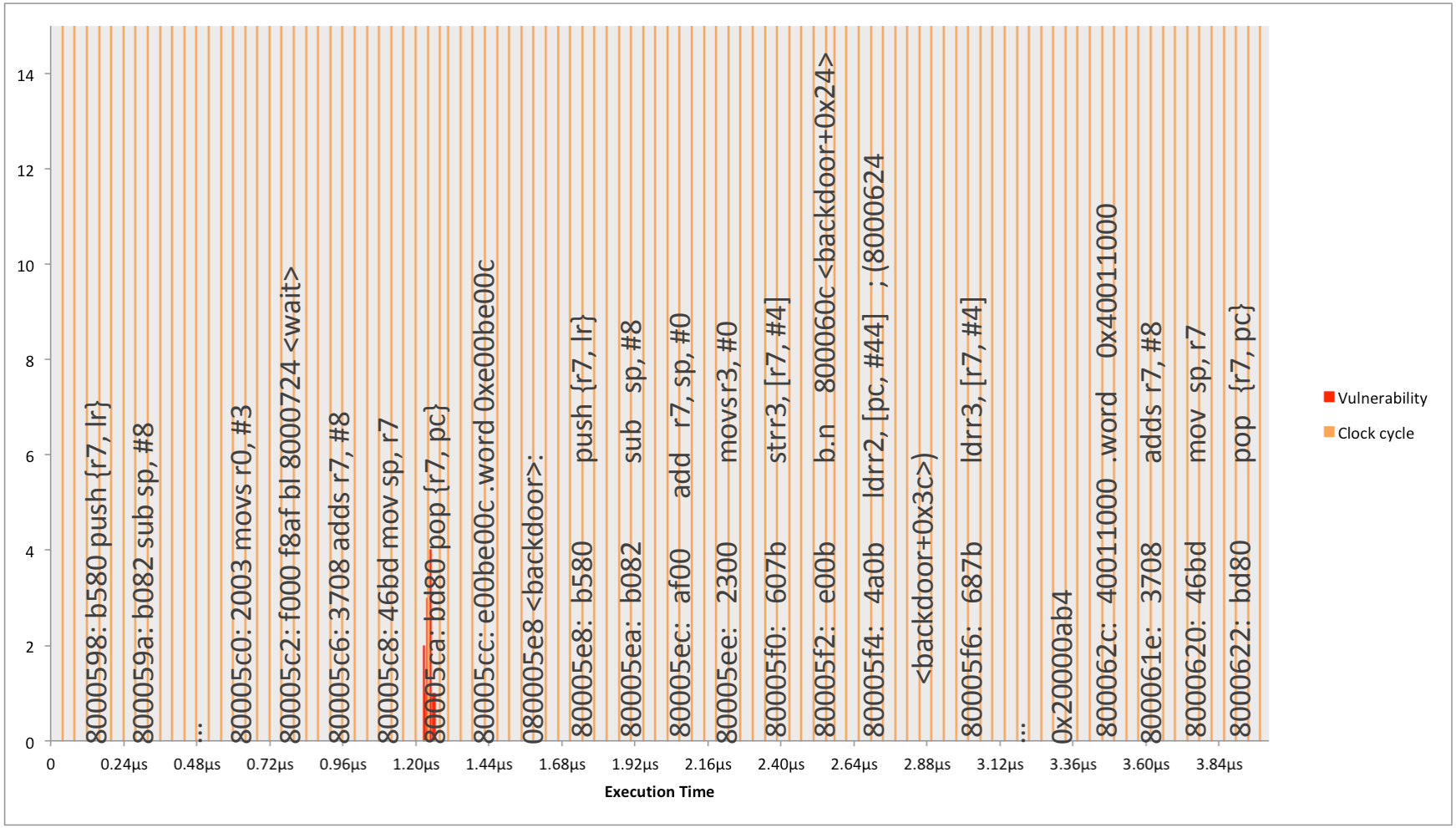
- b.n 80005e8 (Flip)

- sub r5, #128 (Flip)
- mov r0, r0 (Z4B)
- nop (Nop)

Case Study 2: HBFI Experiment Result



Case Study 2: HBFI Experiment Result



Discussion I

- SBFI and HBFI coincide, but it is not exact and not always easy to match.
- SBFI can identify real vulnerabilities, but also un-reproducible ones (in hardware).
- HBFI does not indicate how vulnerability was achieved, or what the actual induced fault was.

Discussion II

- SBFI is clearly not sufficient to claim a vulnerability exists, HBFI is required to verify/exploit.
- SBFI can indicate the most likely place to find a vulnerability with hardware, in practice this can reduce the time/cost/effort required.

Discussion II

- CFH case study :
  - From 1.32µs to 1.4µs.
  - 0.999 probability of fault.
  - Require 10 experiments over 21 of time locations.
  - Reduce required experiments from 117035 to 210.

```
08000aa0 <test_persistence>:
8000aa0: b510 push {r4, lr}
8000aa2: 480a ldr r0, [pc, #40]
8000aa4: 2180 movs r1, #128
8000aa6: 2201 movs r2, #1
8000aa8: f001 f91c bl 8001ce4 <HAL_GPIO_WritePin>
8000aac:  4b08      ldr   r3, [pc, #32]
8000aae:  4807      ldr   r0, [pc, #28]
8000ab0:  681b      ldr   r3, [r3, #0]
8000ab2:  2180      movs  r1, #128;  0x80
8000ab4:  2b01      cmp   r3, #1
8000ab6:  bf0c      ite   eq
8000ab8:  f04f 34ff moveq.w  r4, #4294967295
8000abc:  f04f 3455 movne.w  r4, #1431655765
8000ac0: 2200 movs r2, #0
8000ac2: f001 f90f bl 8001ce4 <HAL_GPIO_WritePin>
8000ac6: 4620 mov r0, r4
8000ac8: bd10 pop {r4, pc}
```

Conclusions

- SBFI is cheap and fast, but the vulnerabilities not guaranteed
- HBFI detect real vulnerabilities, but it is expensive.
- The experimental results of HBFI and SBFI coincide.
- Don't need to know a fault before you start! You can use software to find a potential fault, and hardware to verify it.

Thank You.

Any Questions ?