



This work is distributed
under the CC BY-SA license,
unless stated otherwise.

HardwareSecurity

12. Fault attacks (tutorial)

Ján Jančár

Summer 2026

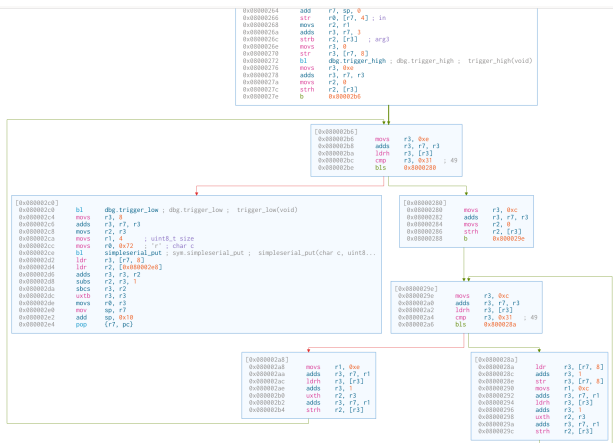
Today

- Targets
- Voltage glitching
- ChipWhisperer Nano

Targets

Loop

```
cnt = 0;
trigger_high();
for(i=0; i<10; i++){
    for(j=0; j<10; j++){
        cnt++;
    }
}
trigger_low();
simpleserial_put("r", 4, (uint8_t*)&cnt);
return cnt == 100;
```

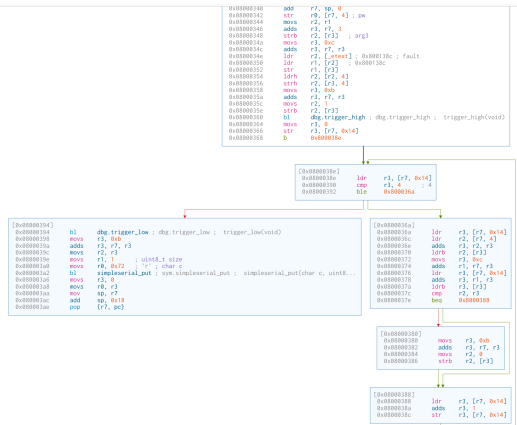


Targets

Password

```
char passwd[] = "fault";
char passok = 1;
int cnt;

trigger_high();
for(cnt = 0; cnt < 5; cnt++){
    if (pw[cnt] != passwd[cnt]){
        passok = 0;
    }
}
trigger_low();
simpleserial_put("r", 1, (uint8_t*)&passok);
return passok;
```



Targets

Comparison

```
uint8_t ok = 5;
trigger_high();
if (*in == 0xA2){
    ok = 1;
} else {
    ok = 0;
}
trigger_low();
simplserial_put("r", 1, (uint8_t*)&ok);
return ok;
```

```
[0x00002ec]
;-- (0x00002ed) aav.0x000002ed:
uint8_t glitch_comparison(uint8_t *in, uint8_t len)
; var int16_t arg3 @ r2
; arg uint8_t *in @ stack + 0x90
; arg uint8_t len @ stack + 0x90
; var uint8_t ok @ stack + 0x90
0x00002ec push {r4, r7, lr}
0x00002ee sub sp, 0x14
0x00002f0 add r7, sp, 0
0x00002f2 str r0, [r7, 4] ; in
0x00002f4 movs r2, r1
0x00002f6 adds r3, r7, 3
0x00002f8 strb r2, [r3] ; arg3
0x00002fa movs r4, 0xf
0x00002fc adds r3, r7, r4
0x00002fe movs r2, 5
0x0000300 strb r2, [r3]
0x0000302 bl dbg.trigger_high ; dbg.trigger_high ; trigger_high(void)
0x0000306 ldr r3, [r7, 4]
0x0000308 ldrb r3, [r3]
0x000030a cmp r3, 0xa2 ; 162
0x000030c bne 0x0000316
```

```
[0x000030e] adds r3, r7, r4
[0x0000310] movs r2, 1
[0x0000312] strb r2, [r3]
[0x0000314] b 0x000031e
```

```
[0x0000316] movs r3, 0xf
[0x0000318] adds r3, r7, r3
[0x000031a] movs r2, 0
[0x000031c] strb r2, [r3]
```

```
[0x000031e] bl dbg.trigger_low ; dbg.trigger_low ; trigger_low(void)
0x0000320 movs r3, 0xf
0x0000322 adds r3, r7, r3
0x0000324 movs r2, r3
0x0000326 movs r1, 1 ; uint8_t size
0x0000328 movs r0, 0x72 ; 'r' ; char c
0x000032a bl simplserial_put ; sym.simplserial_put ; simplserial_put(char c, uint8...
0x000032c r3, 0
0x000032e movs r0, r3
0x0000330 mov sp, r7
0x0000332 add sp, 0x14
0x0000334 pop {r4, r7, pc}
```

Targets

Infinite loop

```
volatile uint8_t a = 0;
```

```
trigger_high();  
trigger_low();
```

```
while(a != 2){  
    ;  
}
```

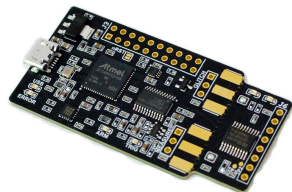
```
// Blink
```

```
00000000 add    r0, r0, #0x10  
00000008 add    r7, sp, #0  
00000010 str    r0, [r7, #4]; in  
0000001c movs    r2, r1  
00000020 adds    r3, r7, #3  
00000024 uhrs    r2, [r3]; arg3  
00000028 movs    r0, #0; unsigned int status  
00000030 dbg_led_ok ; dbg_led_ok ; led_ok(unsigned int status)  
00000034 movs    r0, #0; unsigned int status  
00000038 movs    r0, #0; unsigned int status  
0000003c b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000040 movs    r3, r7, #3  
00000044 adds    r3, r7, #3  
00000048 movs    r2, #0  
0000004c strb   r2, [r3]  
00000050 b1      dbg_trigger_high ; dbg_trigger_high ; trigger_high(void)  
00000054 b1      dbg_trigger_low ; dbg_trigger_low ; trigger_low(void)  
00000058 movs    r8, r8
```

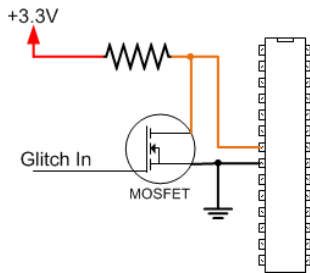
```
[0x00000030]  
00000030 movs    r3, #0xf  
00000034 adds    r3, r7, #3  
00000038 uhrs    r3, [r3]  
0000003c cmp     r3, #2 ; 2  
00000040 bne     0x00000030
```

```
[0x00000030]  
00000030 movs    r0, #1; unsigned int status  
00000034 b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000038 movs    r0, #1; unsigned int status  
0000003c b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000040 movs    r0, #1; unsigned int status  
00000044 b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000048 movs    r0, #1; unsigned int status  
0000004c b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000050 movs    r0, #1; unsigned int status  
00000054 b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000058 movs    r0, #1; unsigned int status  
0000005c b1      dbg_led_error ; dbg_led_error ; led_error(unsigned int status)  
00000060 movs    r0, #0xf7; char c  
00000064 b1      dbg_putch ; dbg_putch ; putch(char c)  
00000068 movs    r0, #0x2; char c  
0000006c b1      dbg_putch ; dbg_putch ; putch(char c)  
00000070 movs    r0, #0x2; char c  
00000074 b1      dbg_putch ; dbg_putch ; putch(char c)  
00000078 movs    r0, #0x4; char c  
0000007c b1      dbg_putch ; dbg_putch ; putch(char c)  
00000080 movs    r0, #0x4; char c  
00000084 b1      dbg_putch ; dbg_putch ; putch(char c)  
00000088 movs    r0, #0x4b; char c  
0000008c b1      dbg_putch ; dbg_putch ; putch(char c)  
00000090 movs    r0, #0x4f; char c  
00000094 b1      dbg_putch ; dbg_putch ; putch(char c)  
00000098 b1      dbg_putch ; dbg_putch ; putch(char c)
```

ChipWhisperer Nano



- Can only do voltage glitching
- Temporarily connects target VCC to ground



ChipWhisperer Nano

Glitch parameters

- `scope.glitch.ext_offset`: The offset of the glitch (from trigger)
- `scope.glitch.repeat`: The length of the glitch

- Serial communication interface
- **Command**
 - Single character command id
 - Payload
 - CRC
- **Response**
 - Single character response id ("r")
 - Payload
 - CRC

ChipWhisperer Nano

Faulting

- Outcomes?
 - Normal behavior (fault did not manifest)
 - Faulty behavior (response is incorrect)
 - Reset (target did not respond at all)
- Too “*little*” fault → normal behavior
- Too “*much*” fault → reset
- **Lots** of trial and error, unfortunately

Tools

- **arm-none-eabi-gcc**: Building and binary examination
- **Rizin** + **Cutter**: Binary analysis

Tasks

Explore

- **1.1:** Explore fault parameter space for **Loop** target
- **1.2:** Determine relationship between glitch offset and sample index
- **1.3:** Explore glitches in different places in the computation

Mitigate

- **2.1:** Implement and test a countermeasure for the **Loop** target

Make notes so we can discuss your solutions together.

Tips and tricks

- Lecture
- [Riscure guidance on SCA/FI](#)
 - FAULT.CONSTANT.CODING
 - FAULT.DETECT
 - FAULT.FLOW
 - FAULT.DOUBLECHECK
 - FAULT.LOOPCHECK
- Hardware Hacking Handbook – Chapter 14
- Check that the compiled code looks right
 - Uses `volatile` variables to avoid optimization

Countermeasures

Example loop

```
cnt1 = 0;
cnt2 = 0
trigger_high();
for(i=0; i<10; i++){
    for(j=0; j<10; j++){
        cnt1++;
    }
}
for(i=0; i<10; i++){
    for(j=0; j<10; j++){
        cnt2++;
    }
}
trigger_low();
if (cnt1 != cnt2) {
    //Fault
}
return cnt1 == 100;
```

Takeaways

- ❓ What are **you** taking away from this tutorial?
- ❓ What surprised you?