Who are we?

Thomas Roth  @stacksmashing
Josh Datko  @cryptotx
Dmitry Nedospasov  @nedos

cryptotronix  levedown security
Special thanks: Colin O’Flynn

- Hardware blockchain expert
- CircuitCellar Author
- Builder of the ChipWhisperer
Takeaways

- Glitching is easier than you think
- You need to know very little about hardware (... or software) to attack a chip
- It can become part of your Software- and Hardware Development Lifecycle
Why are we here?

- Almost never see chips as part of the threat model
- Lot's of people think glitching/fault injection is magic
- We want to make glitching more accessible
Why is this getting important?

- Java methodology: Hack once break in everywhere
- Secure IoT devices are essential for a lot of use-cases
- The "secure" chips in them are not as secure as you would want them to be
What is fault injection?
What is fault injection?
Glitching / Fault injection

- Introduce faults into a chip
  - Cut the power for a very short amount of time
  - Change the period of the clock signal
  - Inject electro magnetic shocks (aka point a tazer at your chip)
Voltage glitching

- Cut the power to the chip for a **very** short amount of time
- At a **very** precisely timed moment
- To cause undefined behaviour
Glitching / Fault injection

Figure 26-2. Maximum Frequency vs. \( V_{CC} \), ATmega48P/88P/168P

- Safe Operating Area

- 20 MHz
- 10 MHz
- 4 MHz

- 1.8V
- 2.7V
- 4.5V
- 5.5V
Glitching / Fault injection

Figure 26-2. Maximum Frequency vs. $V_{CC}$, ATmega48P/88P/168P

But what happens here? 😐

Safe Operating Area
Glitching / Fault injection
Glitching

1. Wait for trigger event
2. Wait $delay
3. Glitch $duration
4. Check result
Glitching

1. Trigger on device boot
2. Wait until in bootloader
3. Glitch firmware validation check
4. Boot modified firmware
Voltage glitching: Flash reads

- Reading flash takes a lot of power
- ...We can interrupt that power
- The result gets undefined/garbled
Voltage glitching: Flash reads

Power consumption

cryptotronix

leveldown security
Voltage glitching: Flash reads
Voltage glitching: Flash reads

Power consumption
Voltage glitching: RAM reads

```cpp
bool firmware_is_valid = validate_firmware();
if(firmware_is_valid)
    boot();
```
Voltage glitching: RAM reads

```c
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
    boot();
```
Voltage glitching: RAM reads

```c
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
    boot();
```
Voltage glitching: RAM reads

```c
bool firmware_is_valid = validate_firmware();
if(firmware_is_valid)
    boot();
```
Voltage glitching: RAM reads

```c
bool firmware_is_valid = validate_firmware();
if(firmware_is_valid)
    boot();
```

If we can flip a single bit in `firmware_is_valid` we can boot!
Voltage glitching: RAM reads

```c
bool firmware_is_valid = validate_firmware();
if (firmware_is_valid) {
    boot();
}
```
How do you actually do this?
Three steps to success

- 1. Prepare the device
- 2. Setup a testing firmware
- 3. Hook it up & glitch!
Part 1: Preparing the device
Power domains

- Chips run on multiple voltages, i.e.:
  - Input/Outputs at 3.3V
  - CPU Core at 0.7-1.2V
  - WiFi at 1.3
Power domains

5.3.1.4 Regulator configuration examples

The voltage regulators can be configured in several ways, depending on the selected supply voltage mode (normal/high) and the regulator type option (LDO or DC/DC).

Four configuration examples are illustrated in images below.

Figure 13: Normal voltage mode, LDO only
Power domains

Microcontroller

- CPU Core: 0.7-1.2V
- WiFi: 1.3V
- GPIO: 3.3V

Device prep
Test firmware
Glitching
Power domains

Microcontroller

- Core regulator
- RF regulator
- CPU Core 0.7-1.2V
- WiFi 1.3V
- GPIO 3.3V
Power domains

Microcontroller

- Core regulator
- CPU Core (0.7-1.2V)
- RF regulator
- WiFi (1.3V)
- GPIO (3.3V)

3.3V

GND
Power domains

- 3.3V
- Core regulator
- CPU Core 0.7-1.2V
- GND
Power domains

Microcontroller

- 3.3V
- Core regulator
- CPU Core (0.7-1.2V)
- External capacitor
- GND
Power domains

Without bypass capacitors

cryptotronix

leveldown security
Power domains

With bypass capacitors
Power domains

Microcontroller

3.3V

Core regulator

Gives direct access to core power supply

CPU Core

0.7-1.2V

GND

cryptotronix

leveldown security
Power domains

Microcontroller

- 3.3V
- Core regulator
- CPU Core (0.7-1.2V)
- GND

Device prep
Test firmware
Glitching
Power domains

Microcontroller

- 3.3V
- VDDCORE
- GND

Core regulator

CPU Core
Power domains
Power domains
Removing capacitors: Problem...

- Depending on a lot of factors the chip might not run stable without capacitors
- Solution: Supply VCORE manually!
Power domains

- 3.3V
- VDDCORE
- GND

Microcontroller
- Core regulator
- CPU Core

Device prep
Test firmware
Glitching

cryptotronix
leveledown security
Power domains

- 3.3V
- VDDCORE
- GND
- Glitch here!

Microcontroller
- Core regulator
- CPU Core

Device prep
Test firmware
Glitching

cryptotronix
leveldown security
Removing capacitors
Removing capacitors

cryptotronix

leveldown security
Part 2: Building a test firmware
Test firmware

- Basic code to allow easy configuration of different chip features
- Should provide a good trigger
- Should provide a success indicator
Test firmware

Arduino sketch:

// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
digitalWrite(2, HIGH);

while(1);
### Test firmware

#### Arduino sketch:

```c
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

  // Our success event
digitalWrite(2, HIGH);

while(1);
```

---

**Device prep** **Test firmware** **Glitching**

1. **Boot device**
2. **Configure clocks, BOD & pins**
3. **Set pin to high as trigger**
4. **Read flash address with known value**
5. **If different, indicate success**
Test firmware

Arduino sketch:

```cpp
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

  // Our success event
digitalWrite(2, HIGH);

while(1);
```

Boot device

Configure clocks, BOD & pins

Set pin to high as trigger

Read flash address with known value

If different, indicate success
Test firmware

Arduino sketch:

```cpp
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
digitalWrite(2, HIGH);

while(1);
```

Boot device

Configure clocks, BOD & pins

Set pin to high as trigger

Read flash address with known value

If different, indicate success
Test firmware

Arduino sketch:

// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
digitalWrite(2, HIGH);

while(1);
Test firmware

Arduino sketch:

// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

// Our success event
digitalWrite(2, HIGH);

while(1);
Test firmware

Pin 1

Pin 2

Depends on success
Test firmware

Pin 1

Pin 2

Prepare system, GPIOs, Clocks, etc

Devices prep
Test firmware
Glitching

Depends on success
Test firmware

Pin 1

Pin 2

Trigger event for the glitcher

Depends on success

Device prep

Test firmware

Glitching

cryptotronix

leveldown security
Test firmware

Pin 1

Pin 2

Delay of the glitcher

Depends on success

Device prep
Test firmware
Glitching
Test firmware

Pin 1

Pin 2

Glitch

Depends on success

Device prep  Test firmware  Glitching

cryptotronix  leveldown security
Test firmware

Check for success

Depends on success
Part 3: The glitcher
The chip.fail glitcher

- Digilent Cmod A7 FPGA Board: ca. $70
- MUX PMOD Multiplexer Breakout: ca. $1.80
- DPS3003 Power Supply: ca. $20
- cryptotronix
  - leveledown security
Digilent Cmod A7

- FPGAs provide super precise timing
- In our case, 100'000'000th of a second
- Open-source chip.fail firmware
MAX PMOD

- Breakout for the Maxim MAX4619
- 3-channel analog switch
- Also available as DIP for breadboarding
MAX PMOD

- Breakout for the Maxim MAX4619
- 3-channel analog switch
- Also available as DIP for breadboarding

Want a PCB? Ask us!
MAX PMOD

- MAX4619
- X0, X1, X, A

cryptotronix
leveldown security
MAX PMOD

MAX4619

X0
X1
X
A

Apply power

cryptotronix
leveldown security

Device prep
Test firmware
Glitching
MAX PMOD

![Diagram of MAX4619](image-url)

- **Device prep**
- **Test firmware**
- **Glitching**

**Cryptotronix**

**Leveldown security**
DPS3003/3005/5005

- Cheap, stable power supply
- We use it for suppling VDDCORE
- Can be controlled via UART
Hooking it up

- **FPGA**
  - Trigger in
  - Success in
  - Power out
  - Glitch out

- **MUX Ch1.**
  - GND

- **MUX Ch2.**
  - 3.3

- **Device under testing**
  - Trigger pin
  - Success indicator
  - Supply voltage
  - VDDCORE

- **Power supply Core**
  - GND

- **cryptotronix**

- **leveledown security**

- **Device prep**
- **Test firmware**
- **Glitching**
The glitcher

cryptotronix

leveldown security
Ready to glitch!
FPGA Bitstream

- Provides a serial interface for setting:
  - Power reset pulse length
  - Delay timing
  - Glitch pulse length
Welcome to the chip.fail glitcher

This Jupyter Notebook shows the configuration options and how to use the chip.fail glitcher.

We start by configuring the path to the serial device of the FPGA. Note that the Cmod A7 shows up as two serial ports, you can figure out which one is the right one by trial and error.

```
In [1]: SERIAL_DEVICE = "/dev/tty.usbserial-210328A9PDC51"
```

Next we set up the different parameters for the glitch:

```
POWER_CYCLE_BEFORE_GLITCH
```

Whether the DUT should be power-cycled before the test. Some devices are very slow to start up (for example the ESP32), and as such it makes more sense to try to glitch and endless loop.

```
POWER_CYCLE_PULSE
```

The duration for which the power-cycle pulse should be send, in 100,000,000th of a second
Host control: Example glitcher

cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
    cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if(gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
Host control: Example glitcher

cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
    cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
    time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if(gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
Host control: Example glitcher

```python
success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
    cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
    time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if (gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
```
Host control: Example glitcher

```python
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False

for delay in tnrage(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrage(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if (gpios & 0x01):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
```
Host control: Example glitcher

cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
    cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
    time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if(gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
    cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
    time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if(gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
Host control: Example glitcher

cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
    time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if(gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
Host control: Example glitcher

```python
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if (gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
```
Host control: Example glitcher

```python
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in tnrage(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
for pulse in tnrage(PULSE_FROM, PULSE_TO, leave=False):
    cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
    cmd(device, CMD_GLITCH)
    time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if(gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
```
Host control: Example glitcher

cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)

success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

# Check whether the glitch was successful!
gpios = cmd_read_uint8(device, CMD_READ_GPIO)
if (gpios & 0x1):
    print("*** SUCCESS ***")
    print("Delay: " + str(delay))
    print("Pulse: " + str(pulse))
    success = True
    break
Glitch pulse : 0
Done, if you get here it means everything is working!

In [*]:
```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)
```

```
success = False
for delay in trange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in trange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        # Loop until the status is == 0, aka the glitch is done.
        # This avoids having to manually time the glitch ;)
        while True:
            pass
        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_GET_STATE)
        if(gpios & 0xl):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

0% 0/50000 [00:00-?, ?h/s]

2% 2/99 [00:00-00:05, 18.13h/s]

In [48]:
```
# Show status of IOs
print(format(cmd_read_uint8(device, CMD_READ_GPIO), '#010b'))
```

0b11111110
Testing real targets
Testing real targets

- Should be commonly used in IoT devices
- Should be modern chips, not very outdated ones
- Should be from different vendors
Testing real targets

- Nordic Semiconductor nRF52840
- Espressif Systems ESP32
- Microchip/Atmel SAM L11 (Secure microcontroller)
- STMicroelectronics STM32F2
Testing real targets: Goals

- Configure the chips in real-world conditions
- Identify whether a chip is susceptible to glitching attacks
- Test the chips in-situ
Nordic Semiconductors nRF52840
Nordic Semiconductors nRF52840

- Multi-protocol 2.4GHz SoC
- Commonly used in smaller IoT sensors
- Target: Makerdiary Micro Dev Kit USB Dongle
nRF52840: Target preparation
nRF52840: Target preparation
nRF52840: Target preparation
nRF52840: Test firmware

- Simple test firmware that enables a pin as trigger and reads flash
- Will be released as part of the full chip.fail release
Glitching results

- First success after 1.5h
- Able to glitch application code
- Stable at ~100 attempts for a successful glitch
Espressif Systems ESP32
Espressif Systems ESP32

- Xtensa Architecture, 160 - 240MHz
- Bluetooth & WiFi
- Very commonly found in new IoT products
The values of C14, L4 and C15 vary with the actual selection of a PCB board.
ESP32: Target preparation

The values of C14, L4 and C15 vary with the actual selection of a PCB board.
ESP32: Target preparation
ESP32: Target preparation
ESP32: Test firmware

- Simple test firmware written in Arduino
- Allows setting different CPU speeds etc
- Very nice for testing

```c
#define TRIGGER_PIN 22
#define GLITCH_PIN 23
#define VALUE 0x3FF44000

void setup()
{
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(GLITCH_PIN, OUTPUT);
  digitalWrite(GLITCH_PIN, LOW);

  uint32_t address = 0x40088400;
  digitalWrite(GLITCH_PIN, LOW);
  while(1) {
    digitalWrite(TRIGGER_PIN, LOW);
    delayMicroseconds(20);
    digitalWrite(TRIGGER_PIN, HIGH);
    delayMicroseconds(5);
    if((*uint32_t*)address != VALUE) {
      digitalWrite(GLITCH_PIN, HIGH);
      while(1);
    }
  }
}

void loop() {}
```

cryptotronix  leveldown security
ESP32: Setup
ESP32: Glitching

- Successful after 3 hours
  - Different clock-speeds, SPI speeds etc
- Had to adjust test firmware to run in a loop - startup too slow
- Stable within ~10’000 attempts
World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology
World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry’s Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology
World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology
World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology
How secure is it?

6.3 Power-On Reset and Brown-Out Detectors

The SAM L10/L11 embed three features to monitor, warn and reset the device:

- A Power-on Reset (POR) on $V_{DD}$ (VDDANA and VDDIO):
  - Monitoring is always activated, including during device startup or during any sleep modes.
  - Having $V_{DD}$ below a fixed threshold voltage will reset the whole device.

  **Note:** Refer to 46.11.2 Power-On Reset (POR) Characteristics for the rising and falling threshold voltages.

- A Brown-out Detector (BOD33) on $V_{DD}$ (VDDANA and VDDIO):
  - The BOD33 can monitor VDD continuously (continuous mode) or periodically (sampled mode) with a programmable sample frequency in active mode as in any sleep modes.
  - A programmable threshold loaded from the NVM User Row is used to trigger an interrupt and/or reset the whole device.

  **Note:** BOD12 is calibrated in production and its calibration parameters are stored in the NVM User Row. These data must not be changed to ensure correct device behavior.
Can we glitch it?
Glitching the SAM L11

- Super susceptible to voltage glitching
- Success after literally 5 minutes
- Bypass of the secure reference boot loader - disclosure in progress
Just HOW easy is it to glitch?
The 5$ Glitcher...
Glitching the SAM L11.. For $5
STM32F2
The million dollar microcontroller
Poof goes your crypto…

WALLET.FAIL

Poof goes your crypto…
Previous work

- STM32F0: JTAG bug + RDP2 -> RDP1 Downgrade
  "Shedding too much Light on a Microcontroller’s Firmware Protection"

- STM32F1/STM32F3: Shaping the Glitch

- How to apply it all? Read:
  "Verifying Code Readout Protection Claims"
  http://circuitcellar.com/cc-blog/verifying-code-readout-protection-claims/
STM32F2

- General purpose microcontroller
- ARM Cortex-M3
- Used particularly often in Bitcoin/Crypto wallets...
STM32F2

STM32 32-bit ARM Cortex MCUs

STM32 CubeMX
- Configuration and initialization tool
- Integrated Development Environments (IDE)
- STM Studio Monitoring tool

STM32Cube MCU Packages
- STM32Cube Expansion Packages

STM32 Ecosystem

Software tools

STM32CubeMX
STM Studio

Mainstream
- STM32 F4
- STM32 F7
- STM32 H7

High performance
- STM32 F2
- STM32 F0

Legend:
- Cortex-M0+/M0
- Cortex-M3
- Cortex-M4
- Cortex-M33
- Cortex-M7

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed euismod, nisl at consectetur tempus, nisl nisl euismod nisl, non vestibulum tellus lacus in turpis.

cryptotronix leveldown security
# STM32 Read-out Protection (RDP)

<table>
<thead>
<tr>
<th>RDP</th>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDP0</td>
<td>0xAA</td>
<td>Full Access</td>
</tr>
<tr>
<td>RDP1</td>
<td>NOT 0xAA or 0xCC</td>
<td>Ability to read RAM + Regs, no flash access, no single stepping</td>
</tr>
<tr>
<td>RDP2</td>
<td>0xCC</td>
<td>No access</td>
</tr>
</tbody>
</table>
# STM32 Read-out Protection (RDP)

<table>
<thead>
<tr>
<th>RDP</th>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDP0</td>
<td>0xAA</td>
<td>Full Access</td>
</tr>
<tr>
<td>RDP1</td>
<td>NOT 0xAA or 0xCC</td>
<td>Ability to read RAM + Regs, no flash access, no single stepping</td>
</tr>
<tr>
<td>RDP2</td>
<td>0xCC</td>
<td>No access</td>
</tr>
</tbody>
</table>
Dumping the bootrom

- The STM32F2 bootrom is readable
- Dumped using openocd, reverse engineered
- Trying to find the RDP2 (0xCC) check

<table>
<thead>
<tr>
<th>RDP</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDP0</td>
<td>0xAA</td>
</tr>
<tr>
<td>RDP1</td>
<td>NOT 0xAA or 0xCC</td>
</tr>
<tr>
<td>RDP2</td>
<td>0xCC</td>
</tr>
</tbody>
</table>
Dumping the bootrom

- The STM32F2 bootrom is readable
- Dumped using openocd, reverse engineered
- Trying to find the RDP2 (0xCC) check

No check found, all checks only for RDP0 (0xAA)

<table>
<thead>
<tr>
<th>RDP</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDP0</td>
<td>0xAA</td>
</tr>
<tr>
<td>RDP1</td>
<td>NOT 0xAA or 0xCC</td>
</tr>
<tr>
<td>RDP2</td>
<td>0xCC</td>
</tr>
</tbody>
</table>
Let's apply our methodology
Bootrom Glitching

POR

cryptotronix

leveldown security
Bootrom Glitching

POR → Bootrom
Bootrom Glitching

POR → Bootrom → User Code
Bootrom Glitching

POR → Bootrom

RDP is read from internal NVM

Bootrom → User Code

cryptotronix leveldown security
Bootrom Glitching

POR

Bootrom

User Code

RDP is read from internal NVM
STM32F2 Boot process (1.8ms)
STM32F2 Boot process (1.8ms)
STM32F2 Boot process (1.8ms)
STM32F2 Boot process (1.4ms)
Power consumption after reset (200µs)
Power consumption after reset (200µs)
Power consumption after reset (200µs)

- BootROM
- Execution
- Flash/Option
- Byte Reads
Power consumption after reset (200µs)
Glitching the Trezor One

- Power supply
- MAX4619 Analog Switch
- Debugger
- FPGA
- Trezor
Eventually... Success!
Parameters

- After mitigations have been implemented in Trezor we are happy to release our glitching parameters:
  - Delay: ~17900
  - Pulse: 50
- Our Trezor Glitcher is now available on Github!
How stable is the glitch?
Dumping the money!

- RDP2 -> RD1 allows for SRAM access
- Secrets get copied to SRAM. Bitcoin secrets = money.
- See wallet.fail for more details
The STM32F2 Glitcher
Options for defense

- Choose a component with glitch monitors (Brown out detector != glitch protection)
- Use active tamper
- Test your design for susceptibility (on dev kits)
- Write glitch-resistant code
Conclusion

- All chips we looked at were trivially glitchable
- Glitching can be done on the cheap - and on the very cheap
- Just because a chip is glitchable does not equal exploit
Releases on chip.fail

- Verilog for chip.fail glitter
- PCBs (schematic/gerbers) for MUX PMOD and Arty Glitcher
- All the target test firmware
Thanks!

- chip.fail - going live later this week
- You can reach us on
  - cryptotronix.com
  - leveldown.de