Fault Attacks on Embedded Software: Threats, Design, and Mitigation

Patrick Schaumont
Professor
Bradley Department of ECE
Virginia Tech

Acknowledgements
FAME Project Team
https://sites.google.com/view/famechip

Supported through
National Science Foundation
Semiconductor Research Corporation
Objective

The black-box model

Fault Injection

input

(Secure) SW

output output'

correct behavior faulty behavior

Fault Analysis
Objective

The black-box model

The grey-box model

Fault Injection

(Secure) SW

input

output

output’

Fault Analysis

correct behavior

faulty behavior
The black-box model

input

(Secure) SW

output

output’

correct behavior

faulty behavior

Fault Injection

Fault Analysis

The grey-box model

(Fault Injection)

Injection

Manifestation

Propagation

Observation

Exploitation

Fault Analysis

(Secure) SW

Microprocessor

Mem Hierarchy
Objective

The black-box model

Fault Injection

input

(Secure) SW

output

output’

correct behavior

fauxy behavior

Fault Analysis

The grey-box model

Fault Injection

Manifestation

Propagation

Observation

Exploitation

- Make a systematic review of the fault-attack process on embedded software
1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Fault Injection Techniques
4. Manifestation and Propagation in the ISA
5. FAME – A Mitigation Technique for Microprocessors
Attacks on Embedded Software

- Embedded Software assumes execution is correct
- (This presentation) Incorrect execution as starting point for attack
  - Privilege Escalation
  - Information Leakage
Privilege Escalation & Information Leakage

• **Privilege Escalation**
  = Adversarial Control of Critical Decisions
    ```
    if (! access_allowed )
      abort( );
    ```

• **Information Leakage**
  = Disclosure of Secret Data & Dependencies
    ```
    if (key_bit)
      out = f(r1);
    else
      out = f(r0);
    ```
    `key_bit` leaks through `out`
Triggering *Incorrect Execution*

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Attack Target</th>
<th>Security Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/Output Attacker</td>
<td>Input/Output Data</td>
<td>Software Bugs</td>
</tr>
<tr>
<td>Memory Attacker</td>
<td>Application/Task Image</td>
<td>Lack of Mem Isolation</td>
</tr>
<tr>
<td>Hardware Attacker</td>
<td>Instruction</td>
<td>Opcode Modification</td>
</tr>
<tr>
<td></td>
<td>Instruction Execution</td>
<td>Micro-Architecture</td>
</tr>
<tr>
<td></td>
<td>Circuit</td>
<td>Timing, Threshold Levels</td>
</tr>
<tr>
<td></td>
<td>Environment</td>
<td>Operating Conditions</td>
</tr>
</tbody>
</table>

*This talk*
Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Fault Injection Techniques
4. Manifestation and Propagation in the ISA
5. FAME – A Mitigation Technique for Microprocessors
Anatomy of a Fault Attack

1. Fault Attack Design
   - Fault Target and Fault Model
   - Fault Injection Method
   - Fault Exploitation Method

2. Fault Attack Implementation
   - Fault Injection
   - Fault Manifestation
   - Fault Propagation
   - Fault Observation
   - Fault Exploitation
Anatomy of a Fault Attack

Physical Level

Fault Injection

electrical transient
Anatomy of a Fault Attack

- Circuit Level
- Physical Level

Fault Injection

Fault Manifestation

faulty bits

electrical transient
Anatomy of a Fault Attack

Fault Injection

Fault Manifestation

Fault Propagation

Hardware

Micro-Architecture Level

Circuit Level

Physical Level

faulty micro-op

faulty bits

electrical transient

Patrick Schaumont (VT)
Anatomy of a Fault Attack

Fault Observation

Fault Propagation

Fault Manifestation

Fault Injection

int verify(S, P){
    int r;
    if (S == P)
        r = 1;
    else
        r = 0;
    return r
}
Anatomy of a Fault Attack

Fault Injection

Fault Manifestation

Fault Propagation

Fault Observation

Fault Exploitation

Patrick Schaumont (VT)
1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Fault Injection Techniques
4. Manifestation and Propagation in the ISA
5. FAME – A Mitigation Technique for Microprocessors
Fault-injection Control

Hardware-controlled Fault Injection

Fault Injection Hardware

Fault Control \rightarrow \text{Injector}

Timing \rightarrow \text{Physical Stress}

I/O \rightarrow \text{MEM} \rightarrow \text{CPU}

Software-controlled Fault Injection

Software Tasks

\text{CTL/Injection} \rightarrow \text{Victim}

Physical Stress

I/O \rightarrow \text{MEM} \rightarrow \text{CPU}
Timing

Vdd
Temp

clk

logic

nominal clock period

Vdd
Temp

clk

logic

nominal clock period

critical path

+ slack
Artificial Timing Faults

- Overclocking
- Clock Glitching
- Underfeeding
- Voltage Glitching
- Overheating

Vdd
Temp

clk

logic

shortened clock period

- slack

nominal clock period

increased critical path

- slack

Patrick Schaumont (VT)
Faraday’s Law

\[ E = -A \cdot \frac{dB}{dt} \]
Faraday’s Law

\[ E = -A \cdot \frac{dB}{dt} \]
Noise Injection – Laser Faults

Glitches

Single Event Upset

Laser Beam

Photocurrent

Vdd

on

0

off

1

Vss

Laser Beam

Flip

Laser Beam

Patrick Schaumont (VT)
Software-Controlled Faults

- **DVFS Interface (CLKSCREW)**

  ![Diagram of PLL, PMIC, Core1, and Core2 with software-controlled timing violation by modified (V2,f2).]

- **Memory Disturbance**

  ![Diagram of row buffer with leak charge @ repeated word access.]

Patrick Schaumont (VT)
## Fault Injection Portfolio

<table>
<thead>
<tr>
<th>Fault Injection</th>
<th>Spatial Precision</th>
<th>Temporal Precision</th>
<th>Cost</th>
<th>Intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overclocking</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Clock f</td>
</tr>
<tr>
<td>Clock Glitching</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>Glitch Width</td>
</tr>
<tr>
<td>Underfeeding</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Voltage</td>
</tr>
<tr>
<td>Voltage Glitching</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>Glitch V/W</td>
</tr>
<tr>
<td>Overheating</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Temperature</td>
</tr>
<tr>
<td>Light Pulse</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
<td>Pulse W/Enrgy</td>
</tr>
<tr>
<td>Laser Pulse</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Pulse W/Enrgy</td>
</tr>
<tr>
<td>EM Pulse</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>Probe Current</td>
</tr>
<tr>
<td>DVFS Interface</td>
<td>Low</td>
<td>Medium</td>
<td>Zero</td>
<td>V/f</td>
</tr>
<tr>
<td>Memory Disturbance</td>
<td>High</td>
<td>Medium</td>
<td>Zero</td>
<td>Disturbance f</td>
</tr>
</tbody>
</table>
1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Fault Injection Techniques
4. Manifestation and Propagation in the ISA
5. FAME – A Mitigation Technique for Microprocessors
Processor Micro-architecture

Programmer’s Model
- Instruction Semantics & Syntax
- Memory Model
- Interrupt/Exception Interface

Instruction Set Architecture

Micro-Architecture
- Control
- Decode
- Datapath
- Flags
- Data Mem
- RegFile
- Fetch
- Store
- Load
Processor Micro-architecture Faults

**Programmer’s Model**
- Instruction Semantics & Syntax
- Memory Model
- Interrupt/Exception Interface

**Instruction Set Architecture**

**Micro-Architecture**

- Faulty Instruction
- Propagation
- Manifestation
  - Fault Location
  - Fault Effect
  - Fault Duration
  - Fault Size

**Diagram:**

- Instruction Memory
- Control
- Decode
- Datapath
- Data Mem
- RegFile
- Flags
- Fetch
- Store
- Load
- Faulty Instruction
- Manifestation
Processor Micro-architecture Faults

**Micro-architecture Element**

- Instruction-memory
- Instruction-fetch
- Instruction-decode
- Operand-fetch
- Execute
- Store
- Data-memory
- Register File
- Status Flags
### Processor Micro-architecture Faults

**Micro-architecture Element**

- Instruction-memory
- Instruction-fetch
- Instruction-decode
- Operand-fetch
- Execute
- Store
- Data-memory
- Register File
- Status Flags

<table>
<thead>
<tr>
<th>Function</th>
<th>Operand</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different instruction</td>
<td>Different source/dest</td>
<td>Different value</td>
</tr>
</tbody>
</table>
Processor Micro-architecture Faults

**Micro-architecture Element**

- Instruction-memory
- Instruction-fetch
- Instruction-decode
- Operand-fetch
- Execute
- Store
- Data-memory
- Register File
- Status Flags

<table>
<thead>
<tr>
<th>Function</th>
<th>Operand</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different instruction</td>
<td>Different source/dest</td>
<td>Different value</td>
</tr>
</tbody>
</table>

Assume a one-bit fault on `ld [%i1 + 4], %g1`

Resulting fault space includes
- **21 ld variants with different load address**
- 6 ld variants with a different target
- 1 add variant
- 1 store variant
- 1 call variant
- 2 unknown variants (trap)
Processor Micro-architecture Faults

Micro-architecture Element

Instruction-memory
Instruction-fetch
Instruction-decode
Operand-fetch
Execute
Store
Data-memory
Register File
Status Flags

Assume a one-bit fault on \texttt{add \%l2, \%l7, \%g2}

Resulting fault space includes

- 12 \textit{add variants with a different source}
- 9 unknown variants (trap)
- 5 \textit{add variants with a different target}
- 3 \textit{arithmetic variants (sub, addx, addcc)}
- 2 \textit{logical variants (or, and)}
- 1 \textit{ld variant}
Processor Micro-architecture Faults

Micro-architecture Element

Instruction-memory

Instruction-fetch

Instruction-decode

Operand-fetch

Execute

Store

Data-memory

Register File

Status Flags

Assume a one-bit fault on \texttt{be 0x40005924}

Resulting fault space includes

- \textit{23 be variants with a different target}
- \textit{5 branch targets with different condition}
- \textit{2 unknown variants (trap)}
- \textit{1 call variant}
- \textit{1 add variant}
Processor Micro-architecture Faults

**Micro-architecture Element**

- Instruction-memory
- Instruction-fetch
- **Instruction-decode**
  - Operand-fetch
  - Execute
  - Store
  - Data-memory
  - Register File
  - Status Flags

- Modifies the PC, can modify control flow
**Processor Micro-architecture Faults**

**Micro-architecture Element**

Instruction-memory
Instruction-fetch
Instruction-decode

**Operand-fetch**
Execute
Store
Data-memory
Register File
Status Flags

Modifies the value of the source operands

- `ld [r1 + r2], r3`  
  - faulty r3
- `cmp r1, r2`  
  - faulty flags
- `be dest`  
  - no effect
Processor Micro-architecture Faults

**Micro-architecture Element**

Instruction-memory
Instruction-fetch
Instruction-decode
Operand-fetch

**Execute**
Store
Data-memory
Register File
Status Flags

- Modifies the value of the computation

```plaintext
ld [r1 + r2], r3  # faulty r3
cmp r1, r2        # faulty flags
be dest           # faulty jump address
```
### Micro-architecture Element

- Instruction-memory
- Instruction-fetch
- Instruction-decode
- Operand-fetch
- Execute
- Store
- Data-memory
- Register File
- Status Flags

- Fault effects on a microarchitecture are highly nonlinear

+ For a given fault effect, analysis is possible
Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Fault Injection Techniques
4. Manifestation and Propagation in the ISA
5. FAME – A Mitigation Technique for Microprocessors

Intermezzo: Fault Exploitation

- DFA Biased Fault Analysis
- Safe Error Analysis
- Fault-Aided SCA
- Fault-Enabled Logical Attacks
Bit-flip Attack on AES

Last round of the Advanced Encryption Standard

Secret state v
9th round

SubBytes

ShiftRows

AddRoundKey

Ciphertext C
Bit-flip Attack on AES

Fault Model:
Bit-flip on a secret state bit

A bit-flip results in a faulty ciphertext byte
• **Fault Differential**

\[ c = \text{sbox}(v) \oplus k \]
\[ c' = \text{sbox}(v') \oplus k \]

Hence \( \Delta = c \oplus c' = \text{sbox}(v) \oplus \text{sbox}(v') \)

• **Fault Analysis**

Reconstruct \( v \) by analyzing \( \Delta \)

Once we know \( v \), we find the last round-key as:

\[ k = \text{sbox}(v) \oplus c \]

32 bit-flip faults in round 10 disclose entire key
[TM 2010] Single random byte fault at 8th round of AES-128: Key $2^{128} \rightarrow 2^{12}$

[SL+ 2012] Two seq. byte fault at 9th, 10th round of AES-192: Key $2^{128} \rightarrow 1$

Current DFA methods are *optimal* IF the fault model can be realized
Implementations and Actual Faults

- **Cryptographic Algorithm** → **Fault Model**
  - Random Byte
  - Random Bit
  - Chosen Bit
  → **DFA**
  - $C, C', C'', .. \rightarrow K$

- **Implementation** → **Fault Injection**
- **Cryptographic Architecture** → **Fault**
Biased Fault Attacks

- Cryptographic Algorithm
- Implementation
- Cryptographic Architecture

Fault Model
- Random Byte
- Random Bit
- Chosen Bit

Fault
- 1-bit, 2-bit, ..

Fault Bias
- DFA
- C, C', C'', .. → K

Variable Fault Intensity
- FSA [2010]
- NUEVA [2012]
- NUFVA [2013]
- DFIA [2014]
- DERA [2015]
- ...

Patrick Schaumont (VT)
Biased Faults as a Side Channel

Biased Fault Injection

S

SBOX

RK

S

faulty S'

(8-dimensional space)

correct S

C

C'

Patrick Schaumont (VT)
Biased Faults as a Side Channel

Under Correct Key Hypothesis

\[ S^{\text{correct}} \]  
\[ \text{SBOX}^{-1}(C' \oplus R_{K_{\text{hyp}}}) \]

Under Wrong Key Hypothesis

\[ S^{\text{faulty}} \]  
\[ \text{SBOX}^{-1}(C' \oplus R_{K_{\text{hyp}}}) \]
Differential Fault Intensity Analysis

1. Inject Faults at different Fault Intensities
   \[ \text{HW}(S \oplus S') < \varepsilon \]
2. Collect Fault Ciphertext \( C' \)
3. For all Key hypothesis \( RK_{hyp} \) compute
   \[ S_{i,RK} = \text{SBOX}^{-1}(C' \oplus RK_{hyp}) \]
4. Select \( RK \) for which
   \[ RK = \text{ArgMin}(\sum_i \sum_j \text{HD}(S_{i,RK}, S_{j,RK})) \]
DFIA versus DFA

DFA
- makes a precise assumption on the injected fault
- needs a system of equations to resolve key guess

DFIA
- makes an approximate model of the injected fault
- uses max likelihood testing to resolve key guess

DFIA relaxes the fault model requirements and is more suitable when fault injection is hard to control
Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Fault Injection Techniques
4. Manifestation and Propagation in the ISA
5. FAME – A Mitigation Technique for Microprocessors
Mitigating Fault Attacks on Embedded SW

Redundant Execution in SW

Sensors and Checkpoint

@ Fault Detection

Secure Trap Handler

Fault Effect

Fault

Memory

Micro Processor

Crypto Software

Memory

Micro Processor

Detector

Checkpoint
## Mitigating Fault Attacks on Embedded SW

<table>
<thead>
<tr>
<th></th>
<th>Strategy 1: Redundant Execution in SW</th>
<th>Strategy 2: HW Sensors and Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>Verify redundant copies</td>
<td>Dedicated HW sensor (Timing, EM, Voltage, ..)</td>
</tr>
<tr>
<td>Response</td>
<td>Correct fault using redundancy</td>
<td>Recover from checkpoint</td>
</tr>
<tr>
<td>Overhead</td>
<td>Redundant execution</td>
<td>Checkpoint storage</td>
</tr>
<tr>
<td>Risk</td>
<td>Redundant Fault</td>
<td>False pos/neg on sensor</td>
</tr>
</tbody>
</table>
## Mitigating Fault Attacks on Embedded SW

### FAME

Fault-attack Aware Microprocessor Extension

<table>
<thead>
<tr>
<th></th>
<th><strong>Strategy 1:</strong> Redundant Execution in SW</th>
<th><strong>Strategy 2:</strong> HW Sensors and Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>Verify redundant copies</td>
<td>Dedicated HW sensor (Timing, EM, Voltage, ..)</td>
</tr>
<tr>
<td>Response</td>
<td>Correct fault using redundancy</td>
<td>Recover from checkpoint</td>
</tr>
<tr>
<td>Overhead</td>
<td>Redundant execution</td>
<td>Checkpoint storage</td>
</tr>
<tr>
<td>Risk</td>
<td>Redundant Fault</td>
<td>False pos/neg on sensor</td>
</tr>
</tbody>
</table>
FAME Operation [HASP 16]

1. fault injection

2. alarm

3. transfer the control to the trap handler

4. access and restore fault-free checkpoint

Fault Detection Unit (FDU) → Fault Control Unit (FCU) → Fault Response Registers (FRR)

Baseline Processor

FAME Processor

Protected Software

Application Software

Secure Trap Handler (STH)

Fault-attack Aware Microprocessor Extensions

Patrick Schaumont (VT)
All-digital Fault Sensors in FAME

Glitch Timing Sensor

In-situ EM Sensor
Fault Response Registers (FRR) for critical processor state, including PC, PSR and last two pipeline stages
FAME Chip 1 Block Diagram

FAME Core Functionality

FAME ASIC

- I$ (1KB)
- D$ (2KB)
- Reset Management

FAME Core
- LEON3 Core (w FRR)
- Sensor (FDU)
- Recovery (FCU)

Connections:
- APB
- AHB
Download and Debug Software

FAME ASIC

FAME Core
- Debug Support Unit
- LEON3 Core (w FRR)
- Sensor (FDU)
- Recovery (FCU)

Reset Management

AHB

APB

Debug UART2
- Debug UART1

SRAM 64KB
- ROM 1KB

I$ (1KB)
- D$ (2KB)

Download and Debug Software

Debugger
FAME Chip 1 Block Diagram

Fault Injection and Fault Diagnosis

- FAME ASIC
  - FAME Core
    - Debug Support Unit
    - LEON3 Core (w FRR)
    - Observe
    - Trigger
    - Sensor (FDU)
    - Recovery (FCU)
  - I$ (1KB)
  - D$ (2KB)
  - AH$ Management
  - GPIO
  - User UART
  - Debug UART1
  - Debug UART2
  - SRAM 64KB
  - ROM 1KB

Fault injector (FPGA)

Debugger

User I/O

Fault injection controller
FAME Chip 1 Block Diagram

FAME ASIC

- I$ (1KB)
- D$ (2KB)
- FAME Core
  - Debug Support Unit
  - LEON3 Core (w FRR)
  - Observe
  - Trigger
- Sensor (FDU)
- Recovery (FCU)
- Reset Management

Fault injector (FPGA)

Fault injection controller

debugger

GPIO
- User UART
- Interrupt Controller

user I/O

AHB
- Debug UART1
- Debug UART2
- SRAM 64KB
- ROM 1KB

APB

Patrick Schaumont (VT)
FAME Chip 1 Micrograph

- 180nm 6LM TSMC
- 25 mm² die area
- Active area
  - LEON3: 6.217 mm²
  - w FAME: 6.301 mm²
  - w FAME+Diag: 6.364 mm²
- FAME extensions overhead 1.35% (of active area)
- 80 MHz clock
- 54 I/O
  - Clock, reset
  - 8 I/O, 16 Core Power
  - 3x UART
  - 4 GPIO
  - 4 Trigger
  - Sensor alarm monitor
  - Scan and test pins
- 108-pin PGA package
FAME Chip 1 Test PCB

- **Debug/User USB-UART**
- **Power Measurement**
- **Power/ Clock Glitcher**
- **FPGA Interface:** GPIO, Trigger, Scan, Alarm
- **SAKURA-G FPGA w glitch generator**
FAME Chip 1 Test Setup

Glitch Control Software

FAME Application Monitor
FAME Chip 1 Fault Sensor
int ptc = 3; // Pin Try Counter
char devicePIN[5] = "12824";

int VerifyPin(userPIN) {
    ptc--;
    if (ptc > 0)
        if (Cmp(userPIN, devicePIN))
            result = 1;
        else
            result = 0;
        ptc--;
    else
        result = 0;
    return result;
}

call 40001f5c <Cmp()>

nop
mov %0, %g1
cmp %g1, 0
be <else branch>

<if_branch>:
    mov 1, %g1
    stb %g1, [ %fp + -2 ]
b <end_of_Cmp()>

<else_branch>:
    clr %b [ %fp + -2 ]
    ldub [ %fp + -1 ], %g1
    add %g1, -1, %g1
    stb %g1, [ %fp + -1 ]
b <end_of_Cmp()>
int ptc = 3;

int ptc = 3;  // Pin Try Counter
char devicePIN[5] = "12824";

int VerifyPin(userPIN) {
    ptc--;
    if (ptc > 0)
        if (ptc > 0)
            if (Cmp(userPIN, devicePIN))
                result = 1;
                ptc++;
                else result = 0;
                else result = 0;
                else result = 0;
                else result = 0;
            return result;
}
int ptc = 3;

int ptc = 3; //Pin Try Counter
char devicePIN[5] = “12824”;

int VerifyPin(userPIN) {
    ptc--;
    if (ptc > 0)
        if (ptc > 0)
            if (Cmp(userPIN,devicePIN))
                result = 1;
                ptc++;
                else result = 0;
            else result = 0;
        else result = 0;
    else result = 0;
    return result;
}
int ptc = 3; //Pin Try Counter
char devicePIN[5] = "12824";
int noFault = 1;
int VerifyPin(userPIN) {
    if (ptc > 0)
        if (Cmp(userPIN, devicePIN))
            result = noFault;
        else
            result = 0;
        ptc--;
    else
        result = 0;
    return result;
}

SecureTrapHandler() {
    if (ptc > 0)
        ptc--;
    noFault = 0;
}
EMFI on FAME

Clock Tree

Leaves

Clock Tree

Root

FAME

Flip-flop

[DAC2018]
Global Effect of EMFI
Injection at clock tree root

Local Effect of EMFI
Injection at clock tree leaves


Questions?

Patrick Schaumont
schaum@vt.edu