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

Fault Analysis

(C) 2018 P. Schaumont (VT)
**Objective**

*The black-box model*  \[\Rightarrow\]  *The grey-box model*

---

**Fault Analysis**

- **Fault Injection**
- **(Secure) SW**
- **Input**
  - ?
  - ?
  - ?

**Output**
- Correct behavior
- Faulty behavior
Objective

The black-box model  The grey-box model

Fault Injection

Fault Analysis

(C) 2018 P. Schaumont (VT)
The black-box model

Fault Injection

input

(Secure) SW

Fault Analysis

output

output’
correct behavior faulty behavior

The grey-box model

Fault Injection

Manifestation

Propagation

Observation

Exploitation

• Make a systematic review of the fault-attack process on embedded software
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
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

  ```
  r1
  if (key_bit)
    out = f(r1);
  else
    out = f(r0);

  key_bit leaks through out
  ```
### Triggering Incorrect Execution

#### Attacker vs. Attack Target Security Failure

<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>

_Copyright: © 2018 P. 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
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

Defined by Security (Attack) Objective
Constrained by Implementation
Anatomy of a Fault Attack

Physical Level

Fault Injection

(C) 2018 P. Schaumont (VT)
Anatomy of a Fault Attack

Circuit Level

Physical Level

Fault Manifestation

Fault Injection

(C) 2018 P. Schaumont (VT)
Anatomy of a Fault Attack

Fault Injection

Fault Manifestation

Fault Propagation

(C) 2018 P. Schaumont (VT)
Anatomy of a Fault Attack

Fault Observation

Fault Propagation

Fault Manifestation

Fault Injection

(C) 2018 P. Schaumont (VT)
Anatomy of a Fault Attack

Fault Injection

Fault Manifestation

Fault Propagation

Fault Observation

Fault Exploitation
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

Software-controlled Fault Injection

Fault Injection Hardware

Fault Control → Injector

Timing

Physical Stress

I/O   MEM   CPU

Software Tasks

CTL/Injection   Victim

Physical Stress

I/O   MEM   CPU

(C) 2018 P. Schaumont (VT)
Timing

- **Vdd**
- **Temp**
- **clk**
- **logic**
- **nominal clock period**
- **critical path**
- **+ slack**
Artificial Timing Faults

- Overclocking
- Clock Glitching

- Underfeeding
- Voltage Glitching
- Overheating

- Slack
- Slack

- Shortened clock period
- Nominal clock period

- Increased critical path
- Critical path
Faraday's Law

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

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

Glitches

Single Event Upset

Laser Beam

Photocurrent

Flip
• **DVFS Interface (CLKSCREW)**

![Diagram of PLL, PMIC, and cores](image)

- Programming Interface
- Software controlled timing violation by modified (V2,f2)

• **Memory Disturbance**

![Diagram of memory disturbance](image)

- Software controlled leak charge @ repeated word access

(C) 2018 P. 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 frequency</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**

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

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

Instruction Set Architecture
- Instruction Memory
  - Fetch
- Control
  - Decode
- Datapath
  - Load
  - Store
- Data Mem
- RegFile
  - Flags

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

Micro-Architecture
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

Assume a one-bit fault on \texttt{ld \#i1 + 4, %g1}

Resulting fault space includes
- 21 \texttt{ld} variants with different load address
- 6 \texttt{ld} variants with a different target
- 1 \texttt{add} variant
- 1 \texttt{store} variant
- 1 \texttt{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

### Function-Operand-Immediate

<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 `add %12, %17, %g2`

*Resulting fault space includes*

- 12 add variants with a different source
- 9 unknown variants (trap)
- 5 add variants with a different target
- 3 arithmetic variants (sub, addx, addcc)
- 2 logical variants (or, and)
- 1 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

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

Assume a one-bit fault on **be 0x40005924**

Resulting fault space includes
- **23 be variants with a different target**
- 5 branch targets with different condition
- 2 unknown variants (trap)
- 1 call variant
- 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
- cmp \(r1, r2\)
- be dest

faulty r3
faulty flags
no effect

---

(C) 2018 P. Schaumont (VT)
**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**
- \( \text{ld} \ [r1 + r2], r3 \) faulty r3
- \( \text{cmp} \ r1, r2 \) faulty flags
- \( \text{be} \ \text{dest} \) faulty jump address
- 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

- Cryptanalysis
- Fault-Aided SCA
- Fault-Enabled Logical Attacks

DFA
- Biased Fault Analysis
- Safe Error Analysis
Last round of the Advanced Encryption Standard

(C) 2018 P. Schaumont (VT)
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
Classic Differential Fault Analysis

Cryptographic Algorithm → Fault Model

\[
\text{Random Byte} \\
\text{Random Bit} \\
\text{Chosen Bit}
\]

DFA
\[
C, C', C'', \ldots \rightarrow K
\]

[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 \rightarrow \text{Fault Model} \bigg\{ Random Byte \linebreak Random Bit \linebreak Chosen Bit \bigg\} \rightarrow DFA \rightarrow C, C', C'', .. \rightarrow K

Implementation

Fault Injection

Cryptographic Architecture \rightarrow \text{Fault}
Biased Fault Attacks

- **Cryptographic Algorithm** → **Fault Model**
  - Random Byte
  - Random Bit
  - Chosen Bit
  - **DFA**
    - C, C’, C”, .. → K

- **Implementation**

- **Cryptographic Architecture** → **Fault Injection**
  - Variable Fault Intensity

- **Fault**
  - **Fault Bias**
    - 1-bit, 2-bit, ..

- **FSA [2010]**
- **NUEVA [2012]**
- **NUFVA [2013]**
- **DFIA [2014]**
- **DERA [2015]**
- ...

(C) 2018 P. Schaumont (VT)
Biased Faults as a Side Channel

\[ \text{Biased Fault Injection} \]

\[ S \rightarrow 8 \]

\[ \text{SBOX} \]

\[ \text{RK} \rightarrow C \]

\[ \varepsilon \]

\[ \text{correct S, faulty S'} \]

\[ (8\text{-dimensional space}) \]

\[ C' \]
Biased Faults as a Side Channel

Under Correct Key Hypothesis

Under Wrong Key Hypothesis

\[ S \xrightarrow{8} \text{SBOX} \]

- **Correct S**
- **Faulty S'**

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

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

- **C'**
- **C'**
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\text{hyp} compute
   \[ S_{i,RK} = \text{SBOX}^{-1}(C' \oplus RK\text{hyp}) \]

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**
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

![Diagram showing Redundant Execution and Sensors and Checkpoint](image-url)
## 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><strong>Detection</strong></td>
<td>Verify redundant copies</td>
<td>Dedicated HW sensor (Timing, EM, Voltage, ..)</td>
</tr>
<tr>
<td><strong>Response</strong></td>
<td>Correct fault using redundancy</td>
<td>Recover from checkpoint</td>
</tr>
<tr>
<td><strong>Overhead</strong></td>
<td>Redundant execution</td>
<td>Checkpoint storage</td>
</tr>
<tr>
<td><strong>Risk</strong></td>
<td>Redundant Fault</td>
<td>False pos/neg on sensor</td>
</tr>
</tbody>
</table>

(C) 2018 P. Schaumont (VT)
Mitigating Fault Attacks on Embedded SW

**FAME**
Fault-attack Aware
Microprocessor Extension

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

Baseline Processor
Fault-attack Aware Microprocessor Extensions

Protected Software
Application Software

Secure Trap Handler (STH)

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

FAME Core

LEON3 Core (w FRR)

Sensor (FDU)

Recovery (FCU)

I$ (1KB)

D$ (2KB)

Reset Management

APB

AHB
Download and Debug Software
FAME Chip 1 Block Diagram

User Applications

FAME ASIC

FAME Core

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

Debug Support Unit

Debug UART2
Debug UART1
SRAM 64KB
ROM 1KB

Debugger

User I/O

GPIO
User UART
Interrupt Controller

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

AHB
APB
FAME Chip 1 Block Diagram

Fault Injection and Fault Diagnosis

Fault injection controller

debugger

user I/O

FAME ASIC

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

FAME Core

Debug Support Unit

Observe

Trigger

LEON3 Core (w FRR)

Sensor (FDU)

Recovery (FCU)

Reset Management

Fault injector (FPGA)

AHB

APB

Debug UART2

Debug UART1

SRAM 64KB

ROM 1KB

GPIO

User UART

Interrupt Controller

(C) 2018 P. 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
Secure Trap Handler Development

```c
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>:
CLRB [%fp + -2]
LDUB [%fp + -1], %g1
ADD %g1, -1, %g1
STB %g1, [%fp + -1]
B <end_of_Cmp()>
```

falls through
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))
                if (Cmp(userPIN,devicePIN))
                    result = 1;
                    ptc++;
                    else result = 0;
                    else result = 0;
                    else result = 0;
                    else result = 0;
                return result;
            else result = 0;
        else result = 0;
    else result = 0;
}
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;
}

No redundancy needed:
FAME FRR Hardware Checkpoint prevents fault propagation

No overhead without fault

Secure trap handler enables user-defined fault response
EMFI on FAME

(C) 2018 P. Schaumont (VT)
Global Effect of EMFI
*Injection at clock tree root*

Local Effect of EMFI
*Injection at clock tree leaves*

146 Faulty Flip Flop

24 Faulty Flip Flop

[DAC2018]
References


Questions?

Patrick Schaumont
schaum@vt.edu