Unboxing the White-Box
Who are we?

- All Principal Security Analyst @Riscure
- Cristofaro Mune
  - Keywords: Software, Reversing, Exploit, Fault Injection…
  - Previous work on Mobile and Embedded Exploitation
- Eloi Sanfelix
  - Keywords: Software security, RE, Exploiting, SCA/FI, CTF
- Job de Haas
  - Keywords: Embedded, Side Channel Analysis, Fault Injection
  - All-round from network pentester to SoC evaluator
What and why…

- White-Box cryptography → Protect keys in untrusted environment
- Increasingly relevant in security solutions
- The idea: Porting Hardware attacks to Software…

…*it works!* Extremely effective approach

- Relevant not only on WBC:
  - Potentially applicable to all Software-based crypto solution
Introduction

Key recovery attacks

Conclusion
Black-Box Security

Encrypt → Decrypt

Observe
Alter
Gray-Box Security

Diagram: A diagram showing the process of encrypting and decrypting data. The process includes steps for observing and altering the data. The diagram includes labels for 'Encrypt', 'Decrypt', 'Observe', and 'Alter'.
Sign of the times…
Sign of the times…
White-Box Security

Encrypt -> Decrypt

Observe
Alter
White-Box Cryptography

- Protection against **key extraction** in the white-box security model
- A technique that allows merging a key into a given crypto algorithm:
  - Described for the first time in 2002 by S. Chow et al.
  - Available for AES and DES
- Lookup tables used for applying mathematical transforms to data
- Remove the distinction between keys and crypto algorithm code.
Software in the White-Box context

- **Input**
- **Output**

- **Business logic**
- **Crypto**

- **I/O**
- **key**

- Can be modified at will
- Direct access
Software Protection

Obfuscation
- Control-flow obfuscation
- Data obfuscation

Anti-analysis and anti-tamper
- Detect debugger/emulator
- Detect hooks and modifications

Device binding
- Bind code to current device

Protected binary

Business logic

I/O

WBC

Focus on this part only

Input

Output
How does WBC work?

Image source: whiteboxcrypto.com
WBC Construction: partial evaluation

```python
def __init__(self, lut, key):
    self._lut = []

    for i in xrange(len(lut)):
        si = lut[i ^ key]
        self._lut.append(si)
```
WBC Construction: encoding

Internal encoding

```
def __init__(self, lut, key, inbij, outbij):
    self._lut = []
    for i in xrange(len(lut)):
        ii = inbij.inv(i)
        si = lut[ii ^ key]
        msi = outbij.apply(si)
        self._lut.append(msi)
```
Example code

```c
void aes128_enc_wb_final(unsigned char in[16], unsigned char out[16])
{
    memcpy(out, in, 16);

    // Let's start the encryption process now
    for (size_t i = 0; i < 9; ++i)
    {
        ShiftRows(out);

        for (size_t j = 0; j < 4; ++j)
        {
            unsigned int aa = Tyboxes[i][j * 4 + 0][out[j * 4 + 0]];
            unsigned int bb = Tyboxes[i][j * 4 + 1][out[j * 4 + 1]];
            unsigned int cc = Tyboxes[i][j * 4 + 2][out[j * 4 + 2]];
            unsigned int dd = Tyboxes[i][j * 4 + 3][out[j * 4 + 3]];

            out[j * 4 + 0] = (T xor ((aa >> 0) & 0xf) & 0xf) & 0xf;
            out[j * 4 + 1] = (T xor ((aa >> 4) & 0xf) & 0xf) & 0xf;
            out[j * 4 + 2] = (T xor ((aa >> 8) & 0xf) & 0xf) & 0xf;
            out[j * 4 + 3] = (T xor ((aa >> 12) & 0xf) & 0xf) & 0xf;
        }
    }

    // Last round which is a bit different
    ShiftRows(out);

    for (size_t j = 0; j < 16; ++j)
    {
        unsigned char x = Tboxes_[j][out[j]];
        out[j] = x;
    }
}
```

Source: https://doar-e.github.io/
Potential attacks on WBC (I)

Data manipulation – Fault Injection (FI)

Modify data here

and observe changes to the output
Potential attacks on WBC (II)

Process manipulation – Fault injection (FI)

Modify code flow here and observe changes to the output
Potential attacks on WBC (III)

Side channel analysis (SCA) / intermediate data analysis

Observe data here

and compare it to expected data here
WBC attack literature

- Attacks for all academic WBC proposals
  - Focus on key extraction
  - Type of transformations assumed known
  - Concrete transformation and key unknown

- In real life…
  - *we do not know much about the design!*

- Not many publicly documented SCA/FI on WBC
  - Implementation-specific DFA paper in 2002 [2]
  - Recent generic DPA-like attack in [3]*

* Authors coined the term Differential Computational Analysis
Key recovery attacks

Introduction

Conclusion
Fault Injection Attacks

…on WBC
Differential Fault Analysis

Correct result

Faulty result

DFA
DFA computation for DES

\[ R_{16} = F(R_{15}, K_{16}) \oplus L_{15} \]
DFA computation for DES

\[ R'_{16} = F(R'_{15}, K_{16}) \oplus L_{15} \]

\[ R_{16} = F(R_{15}, K_{16}) \oplus L_{15} \]

\[ R_{16} \oplus R'_{16} = F(R_{15}, K_{16}) \oplus F(R'_{15}, K_{16}) \]
Divide and conquer

Independent 6-bit sub-keys

Independent 6-bit search
How to port DFA to WBC?
DFA attack process

1. Location of fault injection point

2. Fault injection and ciphertext collection
   - Multiple options available

3. Fault analysis
   - We use our own tools
   - Some AES DFA examples on GitHub
Example target: wbDES

- Binary DES encryption WBC
  - Challenge posted at whiteboxcrypto.com

- DES key hidden within lookups
  - Key value is 0x30 0x32 0x34 0x32 0x34 0x36 0x32 0x36

- We’ll demo all our attacks on this target
STEP 1: Locating the injection point

Red: writes
Green: reads

Target area
STEP 2: Fault injection

1. Select target event within target region

2. Modify data read by that event

```python
def hook_mem_access_fault(uc, access, address, size, value, user_data):
    global output, evtId, fault
    evtId += 1
    pc = uc.reg_read(UC_X86_REG_EIP)

    targetId = user_data[0]
    if access == UC_MEM_READ:
        typ = "r"
        value = u32(uc.mem_read(address, size))
        if should_fault(evtId, targetId, fault, address, size):
            print "FAULTING AT ", targetId
            # Already faulted this time
            fault = False
            # Random bit in this event
            bitfault = 1 << random.randint(0, size*8 -1)
            uc.mem_write(address, pack(value ^ bitfault, size))
```

If event id within target region

Invert a random bit
STEP 3: Analysis

DEMO
## Summary DFA results

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Fault injection</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur (DES)</td>
<td>Unicorn script</td>
<td>Broken in 40 faults</td>
</tr>
<tr>
<td>Hack.lu 2009 (AES)</td>
<td>Debugger script</td>
<td>Broken in 90 faults</td>
</tr>
<tr>
<td>SSTIC 2012 (DES)</td>
<td>Modified lifted code</td>
<td>Broken in 60 faults</td>
</tr>
<tr>
<td>Karroumi (AES)</td>
<td>Modified source code</td>
<td>Broken in 80 faults</td>
</tr>
<tr>
<td>NSC 2013 (encoded AES)</td>
<td>N/A</td>
<td>Not broken – encoding makes DFA not feasible</td>
</tr>
</tbody>
</table>
Side Channel Attacks
...on WBC
What is a DPA attack?

Differential Power Analysis attack

First proposed ~1998 by Paul Kocher to attack on smart cards:

- Measuring power consumption of a crypto execution
- Take multiple measurements for different inputs
- Infer information about the key from the difference of these
Differential trace

Group by known data

Average trace

Subtract

Differential trace
Hypothesis testing
Divide and conquer

Independent 6-bit sub-keys

Independent 6-bit search
Generalization of differential SCA attacks

Take multiple “measurements” of behavior of crypto operations for different data

Predict behavior for sub keys based on the same data and “leakage” model

Apply statistical methods to distinguish the “best” sub key

- Difference of means
- Correlation
- Mutual Information analysis, Linear regression analysis, ...

Find correct guesses for all sub keys to determine key
To our surprise....

It works on White Box Crypto out-of-the-box!!!
SCA attack process

1. Instrument WBC to collect “measurements”
   • Again:

2. Execute WBC with random inputs multiple times

3. Collect “measurement – input/output pairs” in useable form

4. SCA Analysis
STEP1: Capture measurement

- Grab the data using any method that fits your target
  - Instrument execution (e.g., PIN, Valgrind)
  - Capture stack snapshots per crypto round (Hooking, debugger)
  - Use emulators and record (QEMU, Unicorn, PANDA)

- Capture any information during execution that might leak
  - All reads/writes to memory
  - Lower bits of addresses of memory accesses
  - All register contents
STEP2+3: Execute + Collect

- Provide/inject random input data, capture output data
  - Program arguments
  - Use instrumentation from STEP 1

- Store it in a way that allows testing key guesses
  - Store as single bit samples
  - Assure alignment between multiple captures
STEP 4: SCA Analysis

Same target as for DFA: wbDES

Same hidden key: 0x30 0x32 0x34 0x32 0x34 0x36 0x32 0x36

DEMO
## Summary SCA results

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Attacked intermediate</th>
<th>Results</th>
<th>Results NXP [3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur (DES)</td>
<td>Round output</td>
<td>Broken in 75 traces</td>
<td>Broken in 65 traces</td>
</tr>
<tr>
<td>Hack.lu 2009 (AES)</td>
<td>S-Box output</td>
<td>Broken in 16 traces</td>
<td>Broken in 16 traces</td>
</tr>
<tr>
<td>SSTIC 2012 (DES)</td>
<td>Round output</td>
<td>Broken in 16 traces</td>
<td>Broken in 16 traces</td>
</tr>
<tr>
<td>Karroumi (AES)</td>
<td>S-Box and GF(256) inverse</td>
<td>Broken in ~2000 traces</td>
<td>Broken in ~500 traces</td>
</tr>
<tr>
<td>NSC 2013 (encoded AES)</td>
<td>N/A</td>
<td>Not broken</td>
<td>Not broken – encoding makes DPA not feasible</td>
</tr>
</tbody>
</table>
What does it mean?

No detailed knowledge required
- Of WBC implementation
- Where the WBC is processed exactly

No manipulation required

➢ A secret random input/output encoding is the *only* barrier

But:

These random encodings do not work for many real world applications
Is White-Box Crypto dead?
Is WBC broken and useless?

• SCA/Fl on standard WBC very effective:
  • Very limited knowledge required
  • RE skills might be needed
  • Countermeasures and risk mitigation required

• Broken several open-source and commercial WBC
  • Commercial implementations typically require more RE skills

• But…
  • Not regular software crypto → more complex attacks
  • Software protection layers can be a deterrent
  • With renewability it can be good enough
How to make it stronger?

Protected binary

Input

Business logic

Output

WBC

Robustness against advanced SW RE

Robustness against key extraction attacks (SCA, FI, algebraic, …)
But how?

Side Channel Analysis attacks
• Must prevent statistical dependence between intermediates and key
• Typical countermeasures based on randomness difficult in white-box scenario

Differential Fault Analysis attacks
• Double-checks on encoded data → might be bypassed if detected!
• Carry redundant data along computation?
• Break fault models by propagating faults?

Do you have any other ideas?
Thank you!!

eloi@riscure.com
@esanfelix

dehaas@riscure.com

mune@riscure.com
@pulsoid
References


