DRM obfuscation vs auxiliary attacks

Show me your trace and I'll tell you who you are

REcon 2014

QUARKSLAB
INNOVATIVE SECURITY

Authors

Camille Mougey

- @Quarkslab during the study
- @CEA-DAM now
- Like working on obfuscation, RE, networks, algorithms, Water-Pony, ...

Francis Gabriel

- @Quarkslab
- Enjoy RE, cryptography, DRM analysis, ...

We'll speak about . . .

Reverse engineering

- DRM discovery (R&D)
- Attack methodology

We'll speak about ...

Reverse engineering

- DRM discovery (R&D)
- Attack methodology

Execution trace

- Context evolution collection during runtime
- Collected data management & analysis

We'll speak about . . .

Reverse engineering

- DRM discovery (R&D)
- Attack methodology

Execution trace

- Context evolution collection during runtime
- Collected data management & analysis

Code obfuscation

- What we (try to) fight
- Auxiliary attacks (based on execution trace)

A few words on obfuscation

Purposes

- Code protection (whole or part)
- Make the analysis harder and longer
- Raise RE costs

A few words on obfuscation

Purposes

- Code protection (whole or part)
- Make the analysis harder and longer
- Raise RE costs

Some bad guys

- Code flattening
- Data flow protection
- Junk code
- ...

Binary obfuscation is like an onion . . .

DRM discovery

Network communication

- Packets content lookup
- High entropy data

⇒ Maybe some compression or crypto here :)

DRM discovery

Network communication

- Packets content lookup
- High entropy data

⇒ Maybe some compression or crypto here :)

Application's binary analysis (static and dynamic)

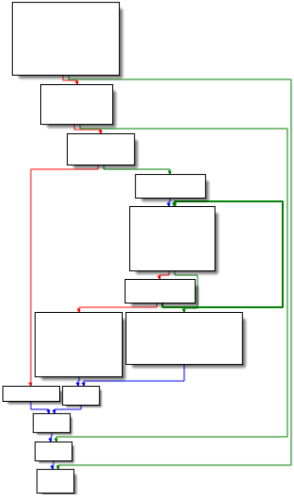
- CFG is flattened
- Instructions in all basic blocks seem obfuscated

Agenda

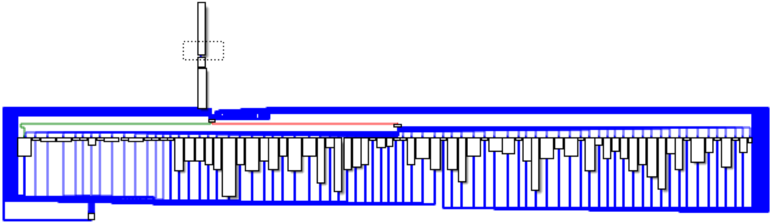
1. First layer: Code flattening
   - Reminder
   - Methods
2. pTra
3. Algorithm reconstruction : RSA-OAEP
4. Rebuilding a cipher function “whiteboxed”: AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

Agenda

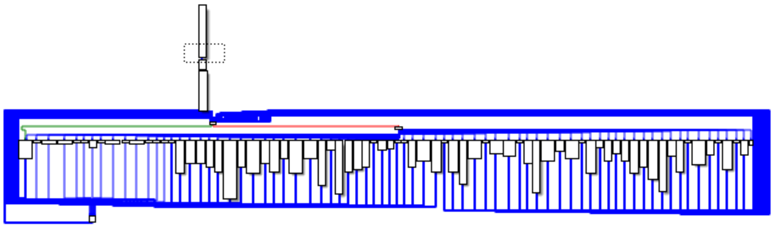
Normal CFG



Flattened CFG



Flattened CFG



How to deal with this kind of protection?

Agenda

Two approaches are possible

Study the protection itself

- Symbolic/Concolic execution of target code
- Advantage: we can reuse know-how on other similar targets

If protection is too complex:

- Lot of resources needed
- Combinatory explosion
- Work in progress...

Two approaches are possible

Study the protection itself

- Symbolic/Concolic execution of target code
- Advantage: we can reuse know-how on other similar targets

If protection is too complex:

- Lot of resources needed
- Combinatory explosion
- Work in progress...

Study only one execution

- Produce an execution trace
- No more CFG but...
- We obtain just one path to analyze
- Advantage: code understanding is easier

What we did

Execution trace approach

1. Context evolution recording
   - registers state
   - executed instructions
   - memory accesses
2. We needed a tool to manage execution trace
3. We needed modules to extract information

What we did

Execution trace approach

1. Context evolution recording
   - registers state
   - executed instructions
   - memory accesses
2. We needed a tool to manage execution trace
3. We needed modules to extract information

Concepts to deal with

- Instrumentation: Execution's data collection
- Database: Efficient trace storage
- Processsing: Relevant information access

What we did

Execution trace approach

1. Context evolution recording
   - registers state
   - executed instructions
   - memory accesses
2. We needed a tool to manage execution trace
3. We needed modules to extract information

Concepts to deal with

- Instrumentation: Execution's data collection
- Database: Efficient trace storage
- Processsing: Relevant information access

That's why we made pTra

Agenda

Agenda

pTra - What we want

Python TRace Analyser

- Execution trace management framework
- Purpose: provide an API for manipulating the trace
- Fully modular, scalable

Constraints

- Architecture independant (re-usability)
- Acceptable response time (usability)

pTra - What we want

Python TRace Analyser

- Execution trace management framework
- Purpose: provide an API for manipulating the trace
- Fully modular, scalable

Constraints

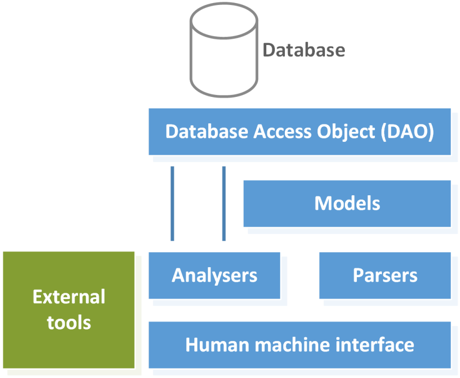
- Architecture independant (re-usability)
- Acceptable response time (usability)

$\Rightarrow$ Generally speaking, be able to quickly implement an idea

Agenda

1. First layer: Code flattening
2. pTra
   - What is this ?
   - A few words on implementation
   - Miasm in 2 slides
3. Algorithm reconstruction : RSA-OAEP
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

Architecture "layered"



Implementation choices

Database

- *MongoDB*
  - Scalable
  - Non relational, a good way to prototype
- A database per trace
  - Avoid inter-trace lock
  - Allow hypothesis on entries

Implementation choices

Database

- *MongoDB*
  - Scalable
  - Non relational, a good way to prototype
- A database per trace
  - Avoid inter-trace lock
  - Allow hypothesis on entries

Getting an execution trace

- Intel PIN
- Miasm sandbox
- IDA, ollydbg, . . .

Implementation choices

Database

- *MongoDB*
  - Scalable
  - Non relational, a good way to prototype
- A database per trace
  - Avoid inter-trace lock
  - Allow hypothesis on entries

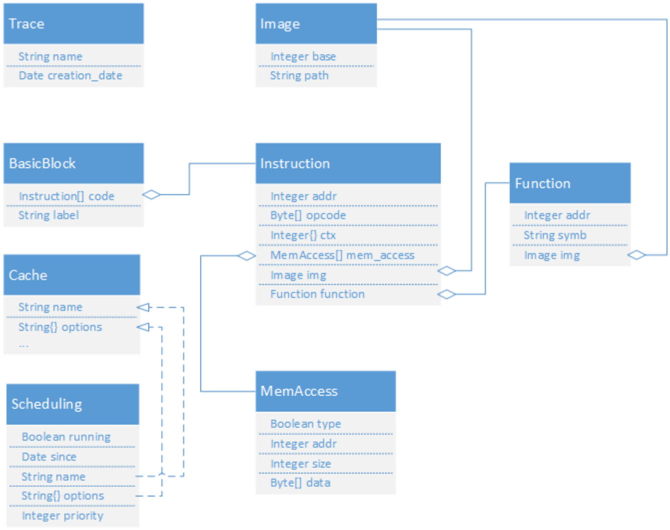
Getting an execution trace

- Intel PIN
- Miasm sandbox
- IDA, ollydbg, . . .

Disassembly engine

- *DiStorm*
- Then Miasm, to be architecture independant . . . and have an IR

Memory model



Detailed information available in [SSTIC 2014 - Actes]

Agenda

1. First layer: Code flattening
2. pTra
   - What is this ?
   - A few words on implementation
   - Miasm in 2 slides
3. Algorithm reconstruction : RSA-OAEP
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

Miasm in 2 slides - 1

Context

- Developed by F. Desclaux
- Miasm v2 released in June 2014
- Available on http://code.google.com/p/miasm

Miasm in 2 slides - 1

Context

- Developed by F. Desclaux
- Miasm v2 released in June 2014
- Available on http://code.google.com/p/miasm

Lego bricks

1. Python
2. Assembly / Disassembly engine "easy-to-write"
3. Intermediate representation RE oriented (8 words)
4. JIT engine (TinyCC, LLVM, Python based)
5. Regression tests :)

Miasm in 2 slides - 2

Features

- Supported architectures
  - x86 {16, 32, 64} bits
  - ARMv7 / Thumb
  - MSP430
  - SH4
  - MIPS32
- Customizable simplification engine
- PE / ELF / shellcode sandboxing
- Common MSDN APIs simulation (or how to rewrite Windows architecture independant)
- ELF / PE binary manipulation thanks to Elfesteem
- Links with STP solver, debuggers, IDA viewer

Miasm in 2 slides - Demonstration

Demo: Shellcode sandboxing (Try & die approach)

Miasm in 2 slides - Demonstration

Demo: ARMv7 execution trace - MD5

Agenda

1. First layer: Code flattening
2. pTra
3. Algorithm reconstruction : RSA-OAEP
   - Introduction
   - Constants detection
   - Dataflow obfuscation
   - Data slicing and functions rebuilding
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

Agenda

1. First layer: Code flattening
2. pTra
3. Algorithm reconstruction : RSA-OAEP
   - Introduction
   - Constants detection
   - Dataflow obfuscation
   - Data slicing and functions rebuilding
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

Algorithm reconstruction - Introduction

What we want to know

- Fully understand an algorithm
- What's inside (encryption, derivations, ...)

$\Rightarrow$ pTra database contains all we need

Algorithm reconstruction - Introduction

What we want to know

- Fully understand an algorithm
- What's inside (encryption, derivations, ...)

$\Rightarrow$ pTra database contains all we need

How to proceed

1. Identify all parts (functions, crypto)
2. Find inputs and outputs of each part
3. Understand links between them

Agenda

1. First layer: Code flattening
2. pTra
3. Algorithm reconstruction : RSA-OAEP
   - Introduction
   - Constants detection
   - Dataflow obfuscation
   - Data slicing and functions rebuilding
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

Constants detection - Theory

What we know

- A cryptographic algorithm can be composed of some "magic" constants
- Hash functions are a good example
- If an algorithm is present, we must find its constants

Constants detection - Theory

What we know

- A cryptographic algorithm can be composed of some "magic" constants
- Hash functions are a good example
- If an algorithm is present, we must find its constants

Where can we find them?

Interesting places:

- Instructions (static analysis)
- Processor's registers
- Memory accesses

$\Rightarrow$ pTra provides a direct access to these elements

Constants detection - Practical

Method

- Add a module to pTra
- Full research in database for known constants
- Avoid false positives
  - Low probability
  - We can group results to detect isolated constants
- Simple, quick and efficient

Constants detection - Practical

Method

- Add a module to pTra
- Full research in database for known constants
- Avoid false positives
  - Low probability
  - We can group results to detect isolated constants
- Simple, quick and efficient

Results

- Mersenne Twister identification (`0x6c078965`)
- SHA-1 identification (`0x67452301`, `0xefcdab89`, `0x98badcfe`, `0x10325476`, `0xc3d2e1f0`)

⇒ Adding SHA-1 primitives knowledge into our call graph (`init`, `update`, `final`)

Agenda

1. First layer: Code flattening
2. pTra
3. Algorithm reconstruction : RSA-OAEP
   - Introduction
   - Constants detection
   - Dataflow obfuscation
   - Data slicing and functions rebuilding
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
6. Bonus

I/O identification - Theory

Purposes

- Unidentified functions:
  - Understanding I/Os can help us to identify them
- Already identified functions:
  - Find where arguments come from
  - Establish the link with other algorithms

⇒ We must find functions input and output

I/O identification - Theory

Purposes

- Unidentified functions:
  - Understanding I/Os can help us to identify them
- Already identified functions:
  - Find where arguments come from
  - Establish the link with other algorithms

⇒ We must find functions input and output

What we know

By studying memory accesses of a function:

- If a data is processed, it will be read
- Results (outputs) will be written

⇒ pTra can help us to find them

I/O identification - Practical

Methods

- To identify outputs:
  - Memory diff
  - (state after) - (state before)
  - We can remove data written and read before the end (temporary data)
- To identify inputs:
  - Data read for the first time by the function
- We can add several heuristics (pointers detection, blocks grouping, entropy computing, ...)

I/O identification - Results

Facts

- Very efficient method to link algorithms parts between them
- We found another protection by looking for I/Os: transformed memory
  - Data in memory never appear in clear format
  - No pattern identified in the code
  - There is a derivation function per memory area

I/O identification - Results

Facts

- Very efficient method to link algorithms parts between them
- We found another protection by looking for I/Os: transformed memory
    - Data in memory never appear in clear format
    - No pattern identified in the code
    - There is a derivation function per memory area

Identified algorithms

- Identified SHA-1 inputs/output verified
- SHA-1 inputs : Certificates ⇒ Cert-chain validation
- RSA-SHA1 signature algorithm is used

⇒ We have to identify RSA function

I/O identification - RSA identification

Main idea

- Destroy modular exponentiation effect of RSA
- Compare execution traces

I/O identification - RSA identification

Main idea

- Destroy modular exponentiation effect of RSA
- Compare execution traces

Steps

1. We know RSA algorithm is used (at least) in cert-chain validation
2. Patch all certificates pub exponents to 1
3. Patch all certificates pub modulus to max value (0xFF..FF)
4. Produce a new execution trace
5. Locate some functions differences (in number of instructions)
6. RSA located ($\pm$50 million instructions)
7. $\Rightarrow$ Add RSA knowledge to the call-graph

Agenda

Data slicing and functions rebuilding

Definitions

- **Data tainting**: find all elements that *depend* on a given one
- **Data slicing**: find all elements *influencing* a given one

Data tainting is forward, and slicing is backward

Data slicing and functions rebuilding

Definitions

- **Data tainting**: find all elements that *depend* on a given one
- **Data slicing**: find all elements *influencing* a given one

Data tainting is forward, and slicing is backward

Data slicing implementation

Using Miasm IR:

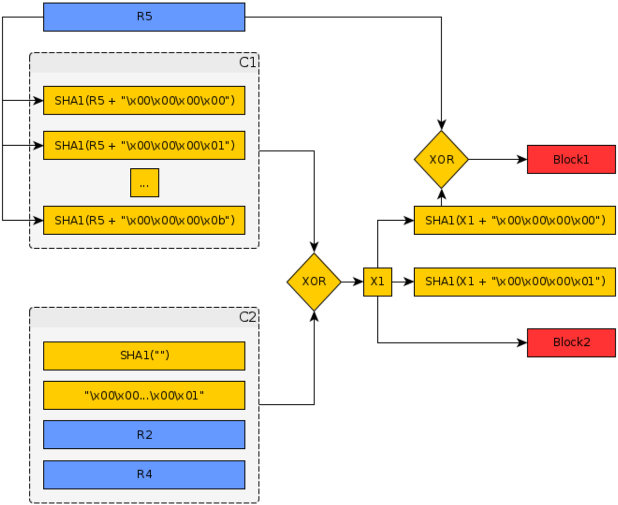
1. Symbolic execution of basic block containing target element
2. We get dependencies of its equation
3. Search for latest writes of each ones
4. And so on.

For data tainting, we proceed almost the same way. We just target elements whose contain the target in their dependencies.

Dependencies graph

Demo: pTra - Slicing as a commercial (with colors)

RSA-OAEP



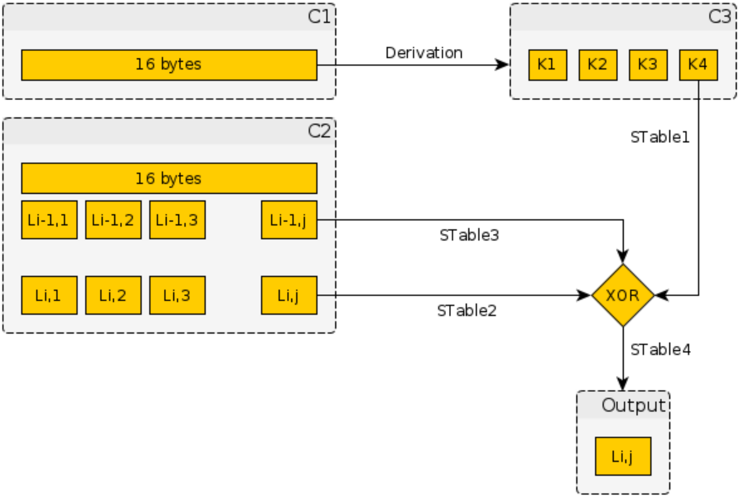
R2, R4, R5 : Random values

Agenda

Agenda

1. First layer: Code flattening
2. pTra
3. Algorithm reconstruction : RSA-OAEP
4. Rebuilding a cipher function "whiteboxed": AES-CBC
   - Some clues
   - Dynamic AES-CBC WhiteBox identification
   - Results
5. Ecofriendly step: Instruction substitution
6. Bonus

Dependencies graph



Equivalence class

Equivalence class statement

Data d1 and d2 are equivalent if and only if their first reads are done by the same instruction. Two instructions are said the same if and only if they share the same address.

Equivalence class

Equivalence class statement

Data d1 and d2 are equivalent if and only if their first reads are done by the same instruction. Two instructions are said the same if and only if they share the same address.

Example

Class:	01	02	03	04	01	02	03	04	01	02	03	04	05
Data:	63	66	F5	F3	76	DC	B1	C1	F6	BC	4D	21	7E

Equivalence class

Equivalence class statement

Data d1 and d2 are equivalent if and only if their first reads are done by the same instruction. Two instructions are said the same if and only if they share the same address.

Example

Class:	01	02	03	04	01	02	03	04	01	02	03	04	05
Data:	63	66	F5	F3	76	DC	B1	C1	F6	BC	4D	21	7E

Grouping

63	66	F5	F3	
76	DC	B1	C1	
F6	BC	4D	21	
				7E

Equivalence class

Applied to dataset

```
1                  16 bytes
+-------------------+
|      Block 1      |
+-------------------+
|      Block 2      |
+-------------------+
|                   |
|     Block 3:      |
|   Group of 16     |
|   bytes block     |
|                   |
+-------------------+
```

Equivalence class

Applied to dataset

```
1                  16 bytes
+--------------------+
|      Block 1       |
+--------------------+
|      Block 2       |
+--------------------+
|                    |
|     Block 3:       |
|   Group of 16      |
|   bytes block      |
|                    |
+--------------------+
```

Applied to output block (reversed way, last write)

```
1  2                16 bytes
+--+
|  | /* 2 bytes blocks */
+--+
   +----------------+
 __|                |
|                   |
|    Bytes group    |
|                   |
|                   |
|           ________|
|           |
+-----------+
            +--------+
 /* Bytes   |        |
 on the     +--------+
 output, but never
 read */
```

Function rebuilding

```
def make_C3(inp):

        C3 = [inp]
        for i in xrange(10):
                tmp = []
                tmp.append(inp[0] ^ table1[(0x100*i)+inp[13]])
                tmp.append(inp[1] ^ table2[inp[14]])
                tmp.append(inp[2] ^ table2[inp[15]])
                tmp.append(inp[3] ^ table2[inp[12]])
                tmp.append(inp[4] ^ tmp[0])
                tmp.append(inp[5] ^ tmp[1])
                tmp.append(inp[6] ^ tmp[2])
                tmp.append(inp[7] ^ tmp[3])
                tmp.append(inp[8] ^ tmp[4])
                tmp.append(inp[9] ^ tmp[5])
                tmp.append(inp[10] ^ tmp[6])
                tmp.append(inp[11] ^ tmp[7])
                tmp.append(inp[12] ^ tmp[8])
                tmp.append(inp[13] ^ tmp[9])
                tmp.append(inp[14] ^ tmp[10])
                tmp.append(inp[15] ^ tmp[11])
                C3.append(tmp)
                inp = tmp

        return C3
```

Comparison between `make_c3` and AES key scheduling

```
def make_C3(self,inp):

    C3 = [inp]
    for i in range(10):
        tmp = []
        tmp.append(inp[0] ^ table1[(0x100*i)+inp[13]])
        tmp.append(inp[1] ^ table2[inp[14]])
        tmp.append(inp[2] ^ table2[inp[15]])
        tmp.append(inp[3] ^ table2[inp[12]])
        tmp.append(inp[4] ^ tmp[0])
        tmp.append(inp[5] ^ tmp[1])
        tmp.append(inp[6] ^ tmp[2])
        tmp.append(inp[7] ^ tmp[3])
        tmp.append(inp[8] ^ tmp[4])
        tmp.append(inp[9] ^ tmp[5])
        tmp.append(inp[10] ^ tmp[6])
        tmp.append(inp[11] ^ tmp[7])
        tmp.append(inp[12] ^ tmp[8])
        tmp.append(inp[13] ^ tmp[9])
        tmp.append(inp[14] ^ tmp[10])
        tmp.append(inp[15] ^ tmp[11])
        C3.append(tmp)
        inp = tmp

    return C3
```

AES Key expansion

```
for size in range(expandedKeySize):

    for k in range(4):
        word[k] = expandedKey[(size - 4) + k]

    if size % sizeKey == 0:

        word = rotate(word)
        for i in range(4):
            word[i] = getSBoxValue(word[i])

        word[0] = word[0] ^ getRconValue(rconIteration)

        rconIteration += 1;

    for m in range(4):
        expandedKey[size] = expandedKey[size - sizeKey] ^ t[m]
        size += 1
```

Agenda

Dynamic AES-CBC WhiteBox identification

Identification

- Try to reproduce intputs/outputs
- ⇒ Results don't match
- ⇒ Encryption steps are completely done on modified states, key in input list
- ⇒ " Dynamic " whitebox

Dynamic AES-CBC WhiteBox identification

Identification

- Try to reproduce intputs/outputs
- $\Rightarrow$ Results don't match
- $\Rightarrow$ Encryption steps are completely done on modified states, key in input list
- $\Rightarrow$ “ Dynamic ” whitebox

Interest in a DRM

- Wasting analysts time
- Hiding inputs and outputs
- Difficulty to reproduce the algorithm on another system (apart ripping it)
- Reverse algorithm is hard to find

Agenda

Results

Attack

1. Homomorphic algorithm (to XOR)
2. Mathematic properties needed
3. A limited set of candidates

⇒ Derivation functions computation

We are finally able to read/alter values encrypted by the algorithm, which is a 128 bits AES-CBC.

Agenda

Agenda

Instruction substitution - Basics

Trivial method

For $x \in [0, 2^{32} - 1]$:

$$f(x) = (16 * x + 16) mod 2^{32}$$

could be rewritten as:

$f(x) = 129441535 - 1793574399 * (1584987567 * (3781768432 * x + 2881946191) - 4282621936)$

Instruction substitution - Basics

Trivial method

For $x \in [0, 2^{32} - 1]$:

$$f(x) = (16 * x + 16) mod 2^{32}$$

could be rewritten as:

$f(x) = 129441535 - 1793574399 * (1584987567 * (3781768432 * x + 2881946191) - 4282621936)$

Simplification

Function simplified by modern compilation passes (particularly constant folding)

Instruction substitution - Advanced

MBA : Mixed Boolean Arithmetic

By mixing logical and arithmetical transformations:

$$(x + y) \equiv ((x \land y) + (x \lor y))$$

$$(x + y) \equiv ((x \oplus y) + 2 \times (x \land y))$$

$$(x \oplus y) - y \equiv (x \land \neg y) - (x \land y)$$

Instruction substitution - Advanced

MBA : Mixed Boolean Arithmetic

By mixing logical and arithmetical transformations:

$$(x + y) \equiv ((x \wedge y) + (x \vee y))$$

$$(x + y) \equiv ((x \oplus y) + 2 \times (x \wedge y))$$

$$(x \oplus y) - y \equiv (x \wedge \neg y) - (x \wedge y)$$

Simplification

- Nothing from compiler passes
- Nothing more from MatLab, Maple, Mathematica or Z3

Instruction substitution - Advanced

MBA : Mixed Boolean Arithmetic

By mixing logical and arithmetical transformations:

$$(x + y) \equiv ((x \wedge y) + (x \vee y))$$

$$(x + y) \equiv ((x \oplus y) + 2 \times (x \wedge y))$$

$$(x \oplus y) - y \equiv (x \wedge \neg y) - (x \wedge y)$$

Simplification

- Nothing from compiler passes
- Nothing more from MatLab, Maple, Mathematica or Z3

Effective simplification

- Once equations are identified, capitalize them thanks to Miasm simplification engine
- By using the generation algorithm of these expressions

MBA generation

Construction

- A matrix A in $\{x, y, x \oplus y, \ldots\}$ base (expressions are represented by their truth table)
- An associated vector v composed of $\{1, \text{-}1\}$ standing for operation between elements
- Equation is valid / generalizable to 2^n iff a linear combination of A's columns is equal to null element

MBA generation

Construction

- A matrix A in $\{x, y, x \oplus y, \dots\}$ base (expressions are represented by their truth table)
- An associated vector v composed of $\{1, -1\}$ standing for operation between elements
- Equation is valid / generalizable to 2^n iff a linear combination of A's columns is equal to null element

Example

$$x + y - (x \oplus y)$$

$$\begin{cases} A = (f_1, f_2, f_3) \\ v = (+1, +1, -1) \end{cases} \quad (1)$$

$$\begin{cases} f_1 = x = (0, 0, 1, 1) \\ f_2 = y = (0, 1, 0, 1) \\ f_3 = x \oplus y = (0, 1, 1, 0) \end{cases} \quad (2)$$

MBA simplification

Example

$$x + \neg x - (x \wedge y) - (x \oplus y) + \neg y$$

MBA simplification

Example

$$x + \neg x - (x \wedge y) - (x \oplus y) + \neg y$$

$$\begin{cases} A = \begin{matrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{matrix} \\ v = (+1, +1, \text{-}1, \text{-}1, +1) \end{cases}$$

MBA simplification

Example

$$x + \neg x - (x \wedge y) - (x \oplus y) + \neg y$$

$$\begin{cases} A & = & \begin{matrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{matrix} \\ v & = & (+1,\ +1,\ \text{-}1,\ \text{-}1,\ +1) \end{cases}$$

Linear combination

$$\begin{matrix} +2 \\ +0 \\ +1 \\ +0 \end{matrix}$$

MBA simplification

Smallest addition to nullify

$$\begin{cases} A & = & \begin{matrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{matrix} \\ v & = & (\text{-}1,\ \text{-}1) \end{cases}$$

Final equation

$$x + \neg x - (x \wedge y) - (x \oplus y) + \neg y - \neg y - \neg(x \vee y) = 0$$

MBA simplification

Smallest addition to nullify

$$\begin{cases} A = \begin{matrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{matrix} \\ v = (\text{-}1, \text{-}1) \end{cases}$$

Final equation

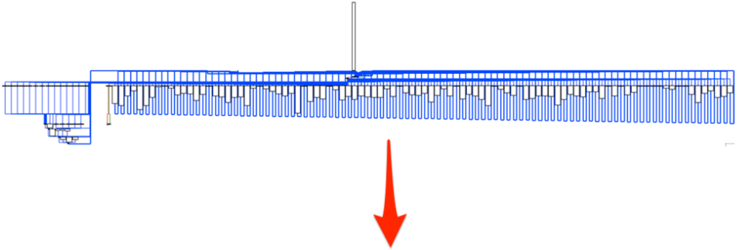
$$x + \neg x - (x \wedge y) - (x \oplus y) + \neg y - \neg y - \neg(x \vee y) = 0$$

$$x + \neg x - (x \wedge y) - (x \oplus y) + \neg y = \neg y + \neg(x \vee y)$$

Agenda

1. First layer: Code flattening
2. pTra
3. Algorithm reconstruction : RSA-OAEP
4. Rebuilding a cipher function "whiteboxed": AES-CBC
5. Ecofriendly step: Instruction substitution
   - Introduction
   - Industrial version
6. Bonus

Transfer equation of the targeted function



```
int f(int x) {
    result = (0xed*(((((((((((((((((((((- ((((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26
0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFF
F)*0x2))+0xFF)&0xFE)+((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*
x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF))*0xE587A503)
0xB717A54D)*0xAD17DB56)+0x60BA9824)&0xFFFFFF46)*0xA57C144B)+((((((- ((((((((0xe5*x + 0xF7)&0xFF + ( 0x0 <<
 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&
0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0x2))+0xFF)&0xFE)+((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*
0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 <<
8)&0xFFFFFFFF))*0xE587A503)+0xB717A54D)*0xE09C02E7)+0xB5ED2776)((((((((((- ((((((((0xe5*x + 0xF7)&0xFF +
( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6
)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0x2))+0xFF)&0xFE)+((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF
)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 <<
 8)&0xFFFFFFFF))*0xE587A503)+0xB717A54D)*0xAD17DB56)+0x60BA9824)&0xFFFFFF46)*0xA57C144B)+(((((((-
((((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF +
( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0x2))+0xFF)&0xFE)+((((((0xe5*x +
0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)
&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF))*0xE587A503) ...
    return result;
}
```

Variable identification, then function resolution: XOR 0x5C

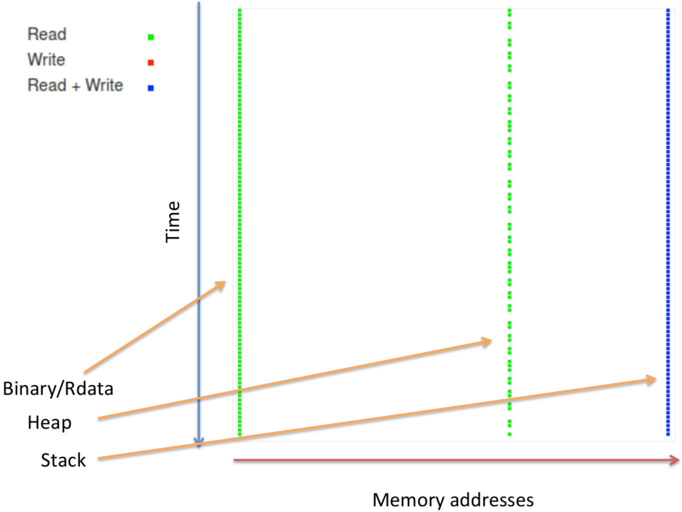
```
int f(int x) {
    x = (0xe5*x + 0xF7) % 0x100;
    v1 = 0x0;
    v2 = 0xFE;
    v0 = (x&0xFF + ( v1 << 8)&0xFFFFFFFF);
    v3 = (((((v0*0xFFFFFE26)+0x55)&v2)+(v0*0xED)+0xD6)&0xFF&0xFF + ( v1 << 8)&0xFFFFFFFF);
    v4 = ((((((- (v3*0x2))+0xFF)&v2)+v3)*0xE587A503)+0xB717A54D);
    v5 = (((((v4*0xAD17DB56)+0x60BA9824)&0xFFFFFF46)*0xA57C144B)+(v4*0xE09C02E7)+0xB5ED2776);
    v7 = ((((v5*0xC463D53A)+0x3C8878AF)&0xCC44B4F4)+(v5*0x1DCE1563)+0xFB99692E);
    v6 = (v7&0x94);
    v8 = ((((v6+v6+(- (v7&0xFF&0xFF + ( v1 << 8)&0xFFFFFFFF)))*0x67000000)+0xD000000) >> 0x18);
    result = ((v8*0xFFFFB22D)+(((v8*0xAE)|0x22)*0xE5)+0xC2)&0xFF & 0xFFFFFFFF;
    result = (0xed*(result-0xF7)) % 0x100;

    return result;
}
```

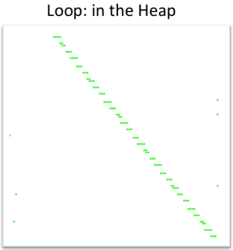
```
int f(int x) {
    return (x & 0xFF) ^ 0x5C;
}
```

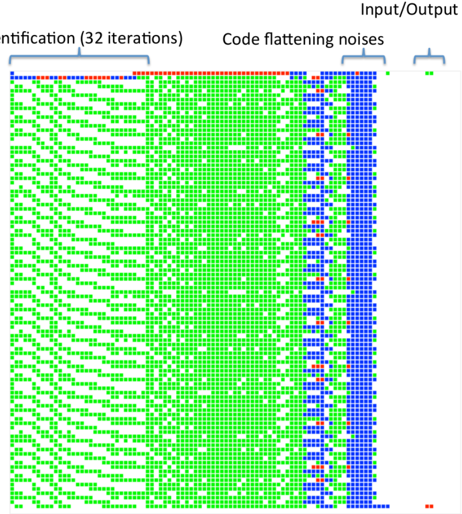
Agenda

Graphing memory accesses over the time



Zoom on stack, loop detection





O-LLVM

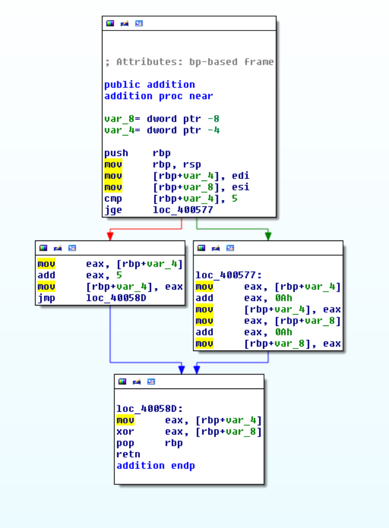
Why O-LLVM?

- Open-source
- Recent project

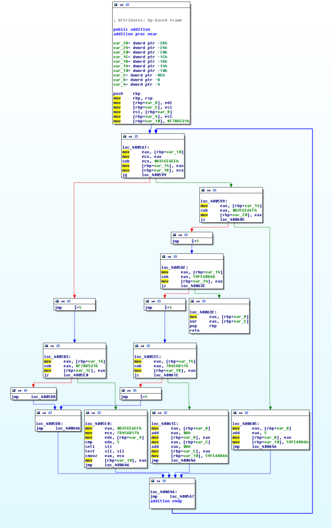
Implemented protections

- Instruction substitution
- Opaque predicates (*Bogus control flow*)
- Code flattening

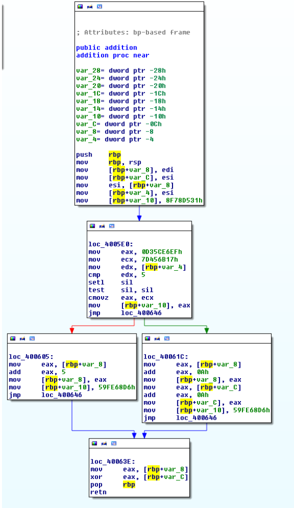
Initial function: addition



After code flattening



CFG rebuilding (using symbolic execution)



So . . .

Conclusion

Approach interests

- Allowed us to analyse state of the art obfuscation mechanisms
- One more method in analyst's toolbox
- Can be used in other cases such as malware analysis, vulnerability research, . . .

Conclusion

Approach interests

- Allowed us to analyse state of the art obfuscation mechanisms
- One more method in analyst's toolbox
- Can be used in other cases such as malware analysis, vulnerability research, . . .

Obfuscation

- More and more used nowadays
- Public initiative O-LLVM, still too young
- Devices, even mobile ones, got enough resources to waste them

Conclusion

Approach interests

- Allowed us to analyse state of the art obfuscation mechanisms
- One more method in analyst's toolbox
- Can be used in other cases such as malware analysis, vulnerability research, ...

Obfuscation

- More and more used nowadays
- Public initiative O-LLVM, still too young
- Devices, even mobile ones, got enough resources to waste them

Our approach isn't better than others; it's just another way to proceed :)

Questions?

QUARKSLAB
INNOVATIVE SECURITY

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com