Code Deobfuscation: Intertwining Dynamic, Static and Symbolic Approaches

Robin David & Sébastien Bardin
CEA LIST
Who are we?

#Robin David
- PhD Student at CEA LIST

#Sébastien Bardin
- Full-time researcher at CEA LIST

About our lab

Atomic Energy Commission (CEA LIST), Paris Saclay
- Software Safety & Security Lab
  - frama
  - BINSEC
**Context & Goal**

- Analysis of obfuscated binaries and malware (potentially self-modifying)
- Recovering high-level view of the program (e.g., CFG)
- Locating and removing obfuscation if any

**Challenges**

- Static, dynamic and symbolic analyses are not enough used alone
- Scalability, robustness
Our proposal

- A new symbolic method for infeasibility-based obfuscation problems
- A combination of approaches to handle obfuscations impeding different kind of analyses

Achievements

- A set of new tools and algorithms to analyse binaries
- Detection of several obfuscations in packers
- Deobfuscation of the X-Tunnel malware
Long term objectives

- Disassembling highly obfuscated codes is challenging
- Combining static, dynamic and symbolic is promising (accurate and efficient)

Takeaway message
Agenda

Background
1. Disassembling obfuscated codes
2. Dynamic Symbolic Execution

Our proposal
3. Backward-Bounded DSE
4. Analysis combination

Binsec
5. The Binsec platform

Case-studies
6. Packers
7. X-Tunnel
Disassembling obfuscated codes

Getting an exploitable representation of the program
An essential task before in-depth analysis is the CFG disassembly recovery of the program
**Disassembly process**

**Code discovery**
(aka. Decoding opcodes)
- Non-code bytes
- Missing symbols (function addr)
- Instruction overlapping

**CFG reconstruction**
(aka. Building the graph, nodes & edges)
- Indirect control-flow
- Non-returning functions

**CFG partitioning**
(aka. Finding functions, bounds etc)
- Function code sharing
- Non-contiguous function
- Tail calls

*segmentation proposed in Binary Code is Not Easy, Xiaozhu Meng, Barton P. Miller*
Obfuscation

Any means aiming at slowing-down the disassembly and analysis process either for a human or an automated algorithm.
## Obfuscation Diversity

### Control Vs Data

<table>
<thead>
<tr>
<th></th>
<th>Target</th>
<th>Against</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Control</td>
<td>Data</td>
</tr>
<tr>
<td><strong>CFG flattening</strong></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td><strong>Jump encoding</strong></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td><em>(direct → indirect/computed)</em></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td><strong>Opaque predicates</strong></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>VM (Virtual-Machines)</strong></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Polymorphism</strong> <em>(self-modification, resource ciphering)</em></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Call stack tampering</strong></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Anti-debug/Anti-tampering</strong></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Signal/Exception</strong></td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

and so many others....
Opaque predicates

**Definition:** Predicate always evaluating to true (resp. false). (but for which this property is difficult to deduce)

**Can be based on:**
- Arithmetic
- Data-structure
- Pointer
- Concurrency
- Environment

**Corollary,** the dead branch allows to:
- Grow the code (artificially)
- Drown the genuine code

Example:
```assembly
mov   eax, ds:X
mov   ecx, ds:Y
imul  ecx, ecx
imul  ecx, 7
sub   ecx, 1
imul  eax, eax
cmp   ecx, eax
jz    <dead_addr>
```

eg: $7y^2 - 1 \neq x^2$
(for any value of $x, y$ in modular arithmetic)
**Call stack tampering**

**Definition:** Alter the standard compilation scheme of call and ret instructions

**Corollary:**
- Real ret target hidden, and returnsite potentially not code
- Impede the recovery of control flow edges
- Impede the high-level function recovery

In addition, able to characterize the tampering with alignment and multiplicity.

Need to handle the tail call optimization.
Deobfuscation

- Revert the transformation (often impossible)
- Simplify the code to facilitate later analyses
## Disassembly

### Notations
- **Correct**: only genuine (executable) instructions are disassembled
- **Complete**: all genuine instructions are disassembled

### Standard approaches

<table>
<thead>
<tr>
<th></th>
<th>static</th>
<th>dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>robust <em>(obfuscation)</em></td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>correct</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>complete</td>
<td>⚫</td>
<td>⚫</td>
</tr>
</tbody>
</table>
Disassembly

Notations

- **Correct**: only genuine (executable) instructions are disassembled
- **Complete**: all genuine instructions are disassembled

Standard approaches

- Static disassembly

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>robust (obfuscation)</td>
<td>⚪</td>
<td>⚫</td>
</tr>
<tr>
<td>correct</td>
<td>⚪</td>
<td>⚫</td>
</tr>
<tr>
<td>complete</td>
<td>⚪</td>
<td>⚫</td>
</tr>
</tbody>
</table>

**Dynamic jump**
Disassembly

Notations

- **Correct**: only genuine (executable) instructions are disassembled
- **Complete**: all genuine instructions are disassembled

Standard approaches

- Static disassembly
- Dynamic disassembly

<table>
<thead>
<tr>
<th></th>
<th>static</th>
<th>dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>robust (obfuscation)</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>correct</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>complete</td>
<td>⚫</td>
<td>⚫</td>
</tr>
</tbody>
</table>

- Dynamic jump: input dependent
2 Dynamic Symbolic Execution

a.k.a Concolic Execution
Symbolic Execution: mean of executing a program using symbolic values (logical symbols) rather than actual values (bitvectors) in order to obtain in-out relationship of a path.

How to reach “OK”?

Source Code (C)
```c
int f(int a, int b) {
    if (a < 10) {
        if (a > b) {
            printf("OK");
        }
    }
}
```

Formula:
```
a < 10 ∧ a > b
```

Solution:
```
a=5, b=1
```
(using SMT solvers)
Symbolic Execution: mean of executing a program using symbolic values (logical symbols) rather than actual values (bitvectors) in order to obtain in-out relationship of a path.

How to reach “OK”?

Source Code (C)
```c
int f(int a, int b) {
    if (a < 10) {
        if (a > b) {
            printf("OK");
        }
    }
}
```

Dynamic Symbolic Execution: works on a dynamic path

Formula:
```
a < 10 \land a > b
```

Solution:
```
a=5, b=1
```
(using SMT solvers)
Why use DSE?

- Obfuscation alters the syntax but keeps the semantic
- DSE finds new paths
DSE on a switch

Source Code (C)
enum E = {A, B, C}
int myfun(int x) {
    switch(x) {
        case A: x+=0; break;
        case B: x+=1; break;
        case C: x+=2; break;
    }
}

x86 assembly
push ebp
mov ebp, esp
cmp [ebp+8], 3
ja @ret
mov eax, [ebp+8]
shl eax, 2
add eax, JMPTBL
mov eax, [eax]
jmp eax
 [...]
ret
DSE on a switch

Source Code (C)

```c
enum E = {A, B, C}
int myfun(int x) {
    switch(x) {
    case A: x+=0; break;
    case B: x+=1; break;
    case C: x+=2; break;
    }
}
```

x86 assembly

```assembly
push ebp
mov ebp, esp
cmp [ebp+8], 3
ja @ret
mov eax, [ebp+8]
shl eax, 2
add eax, JMPTBL
mov eax, [eax]
jmp eax
jmp eax
 [...]
ret
```

Symbolic Execution

(input:esp, ebp, memory)

```symbolic
push ebp
@[esp] := ebp
ebp1 := esp

@[ebp1+8] < 3

eax1 := @[esp+8]
eax2 := eax1 << 2
eax3 := eax2 + JMPTBL
eax4 := @[eax3]
eax4 == 2
```

Path predicate $\Phi$:

- $@[ebp1+8] < 3 \land eax4 == 2$
- $@[esp+8] < 3 \land @[(@[esp+8] \ll 2) + JMPTBL] == 2$
DSE on a switch

Source Code (C)

```
enum E = {A, B, C}
int myfun(int x) {
    switch(x) {
    case A: x+=0; break;
    case B: x+=1; break;
    case C: x+=2; break;
    }
}
```

x86 assembly

```
push ebp
mov ebp, esp
cmp [ebp+8], 3
ja @ret
mov eax, [ebp+8]
shl eax, 2
add eax, JMPTBL
mov eax, [eax]
jmp eax
```

Symbolic Execution

(input: esp, ebp, memory)

```
push ebp
@[esp] := ebp
ebp1 := esp
@[ebp1+8] < 3
mov eax, [ebp+8]
eax1 := @[esp+8]
eax2 := eax1 << 2
eax3 := eax2 + JMPTBL
eax4 := @[eax3]
eax4 == 2
jmp eax
```

Path predicate \( \phi \):

\[
@[ebp1+8] < 3 \land eax4 \neq [0,2]
\]

\[
@[esp+8] < 3 \land @[(@[esp+8] \ll 2) + JMPTBL] \neq [0,2]
\]
DSE Vs Static & Dynamic approaches

Advantages:
- path sure to be feasible (unlike static)
- can generate new inputs (unlike dynamic)
- thwart basic tricks (code-overlapping, SMC, etc)
- easier than static semantic analysis
  - next instruction always known
  - loops unrolled

The challenge for DSE is to make it scale on huge path length and to cover all paths...
What if instead we want to check infeasibility properties?

- no any other target for dynamic jump
- opaque predicates

Dynamic and DSE allow to check feasibility properties

- find new targets for dynamic jump
- cover a new branch
3 Backward-Bounded DSE (bb-DSE)
Complementary approach for infeasibility-based problems
bb-DSE: Example of opaque predicate

Goal
Check that the branch to XX is infeasible

- **not enough**
  (still feasible w.r.t. ecx, eax)

- **minimal**
  (backtrack enough constraints to prove the infeasibility)

- **complete**
  (backtrack all dependencies)

mov eax, ds:X
mov ecx, ds:Y
imul ecx, ecx
imul ecx, 7
sub ecx, 1
imul eax, eax
cmp ecx, ecx
jz XX

xx infeasible branch?
bb-DSE: Example of a call stack tampering

Goal
Check that the return address cannot be tampered by the function

- **false negative**: miss the tampering (too small bound)
- **correct**: find the tampering
- **complete**: validate the tampering for all paths
Backward-Bounded DSE

Summary
- Backward: infeasibility
- Bounded reasoning: scale
- Adaptable bound (for the need)
- Dynamic: robustness (hence false positive)

Shortcomings
- False negative (FN): too small bound
- False positive (FP): not enough paths

<table>
<thead>
<tr>
<th></th>
<th>(forward) DSE</th>
<th>bb-DSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility queries</td>
<td><img src="green-circle.png" alt="Green Circle" /></td>
<td><img src="red-circle.png" alt="Red Circle" /></td>
</tr>
<tr>
<td>Infeasibility queries</td>
<td><img src="red-circle.png" alt="Red Circle" /></td>
<td><img src="green-circle.png" alt="Green Circle" /></td>
</tr>
<tr>
<td>Scale</td>
<td><img src="green-circle.png" alt="Green Circle" /></td>
<td><img src="green-circle.png" alt="Green Circle" /></td>
</tr>
</tbody>
</table>

Not FP/FN free, but very low rates
Bound selection

Need to be adapted to the problem to solve

Call stack tampering: ret → call

Opaque predicates: Trade-off FP/ FN

FN: OP missed (backtracking not enough)

FP: not OP but infeasible w.r.t. path taken

Empiric results obtained through benchmarking

-16-20 empiric results obtained through benchmarking

Detection rate (%)

False negatives

False positives

0 bound

Best
Combination

Intertwining Dynamic, Static and Symbolic
Combination: Principles

Goal: Obtaining a safer and more precise disassembly handling several obfuscation constructs.

The ultimate goal is to provide a semantic-aware disassembly based on information computed by symbolic execution.
Combination: Application

- safe dynamic disassembly with dynamic jumps
**Combination: Application**

- safe dynamic disassembly with dynamic jumps
- multiple self-modification segmentation
Combination: Application

- safe dynamic disassembly with dynamic jumps
- multiple self-modification segmentation
- enlarge partial CFG on genuine conditional jump
Combination: Application

- Safe dynamic disassembly with dynamic jumps
- Multiple self-modification segmentation
- Enlarge partial CFG on genuine conditional jump
- Do not disassemble dead branch of opaque predicate
**Combination:** Application

- safe dynamic disassembly with dynamic jumps
- multiple self-modification segmentation
- enlarge partial CFG on genuine conditional jump
- do not disassemble dead branch of opaque predicate
- disassemble the target of tampered ret
Combination: Application

- Safe dynamic disassembly with dynamic jumps
- Multiple self-modification segmentation
- Enlarge partial CFG on genuine conditional jump
- Do not disassemble dead branch of opaque predicate
- Disassemble the target of tampered ret
- Do not disassemble the return site of tampered ret
5 BINSEC
Binsec platform architecture

- BINSEC
  - main binary analysis platform
  - DSE, bb-DSE
  - static

- PINSEC
  - dynamic analysis instrumentation

- ØMQ
  - new inputs
  - queries

- IDASEC
  - IDA plugin for result exploitation

Open source, beta available at:
- Binsec+Pinsec: http://binsec.gforge.inria.fr
- IDASec: https://github.com/RobinDavid/idasec
Pintool based on Pin 2.14-71313

Features:

- Generate a protobuf execution trace (with all runtime values)
- Can limit the instrumentation time / space
- Working on Linux / Windows
- Configurable via JSON files
- Allow on-the-fly value patching
- Retrieve some function parameters on known library call
- Remote control (prototype)
- Self-modification layer tracking

Still lacks many anti-debug/anti-VM countermeasures..
**Binsec** (main platform)

**Features:**
- front-end: x86 (+simplification)
- disassembly: linear, recursive, linear+recursive
- static analysis: abstract interpretation

**Binsec/SE** (symbolic execution engine)

**Features:**
- generic C/S policy engine
- path selection for coverage (thanks to Josselin Feist & TDT)
- configurable via JSON file
- (basic) stub engine for library calls (+cdecl, stdcall)
- analysis implementation
- path predicate optimisations
- SMTLIB2, SMT solvers supported: Z3, boolector, Yices, CVC4

Many other DSE engines: Mayhem (ForAllSecure), Triton (QuarksLab), S2E, and all DARPA CGC challengers ....
Python plugin for IDA (from 6.4)

Goal:
- triggering analyses remotely from IDA and results post-processing
- leveraging Binsec features into IDA

Features:
- DBA decoding of an instruction
- reading an execution trace
- colorizing path taken
- dynamic disassembly (following the execution trace)
- triggering analyses via remote connection to Binsec
- analysis results exploitation
Packers study
Packers & X-Tunnel
Packer: deobfuscation evaluation

Evaluation of 33 packers
(packed with a stub binary)

Looking for (with bb-DSE):
  ◦ Opaque predicates
  ◦ Call stack tampering
  ◦ Record of self-modification layers

Settings:
  ◦ Execution trace limited to 10M instructions

Goal: To perform a systematic and fully automated evaluation of packers
### Packer: Analysis results

<table>
<thead>
<tr>
<th>packers</th>
<th>trace len.</th>
<th>#proc</th>
<th>#th</th>
<th>#SMC</th>
<th>opaque predicates</th>
<th>call stack tampering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK</td>
<td>OP</td>
</tr>
<tr>
<td>ACProtect v2.0</td>
<td>1.8M</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>83</td>
<td>159</td>
</tr>
<tr>
<td>ASPack v2.12</td>
<td>377K</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>168</td>
<td>24</td>
</tr>
<tr>
<td>Crypter v1.12</td>
<td>1.1M</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>399</td>
<td>24</td>
</tr>
<tr>
<td>Expressor</td>
<td>635K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>81</td>
<td>8</td>
</tr>
<tr>
<td>FSG v2.0</td>
<td>68k</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>Mew</td>
<td>59K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>PE Lock</td>
<td>2.3M</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>95</td>
<td>90</td>
</tr>
<tr>
<td>RLPack</td>
<td>941K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>TELock v0.51</td>
<td>406K</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Upack v0.39</td>
<td>711K</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>41</td>
<td>1</td>
</tr>
</tbody>
</table>

- Several have no such obfuscation, NeoLite, nPack, Packman, PE Compact ....
- Several packers still evade the DBI, Armadillo, BoxedApp, EP Protector, VMProtect....
- 3 reached the 10M instructions limit, Enigma, svk, Themida
Several have no such obfuscation, NeoLite, nPack, Packman, PE Compact ....

Several packers still evade the DBI, Armadillo, BoxedApp, EP Protector, VMProtect....

3 reached the 10M instructions limit, Enigma, svk, Themida

<table>
<thead>
<tr>
<th>packers</th>
<th>trace len.</th>
<th>#proc</th>
<th>#th</th>
<th>#SMC</th>
<th>opaque predicates</th>
<th>call stack tampering</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACProtect v2.0</td>
<td>1.8M</td>
<td>1</td>
<td></td>
<td></td>
<td>159</td>
<td>0</td>
</tr>
<tr>
<td>ASPack v2.12</td>
<td>377K</td>
<td>1</td>
<td></td>
<td></td>
<td>24</td>
<td>11</td>
</tr>
<tr>
<td>Crypter v1.12</td>
<td>1.1M</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>399</td>
<td>24</td>
</tr>
<tr>
<td>Expressor</td>
<td>635K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>81</td>
<td>8</td>
</tr>
<tr>
<td>FSG v2.0</td>
<td>68K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>Mew</td>
<td>59K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>PE Lock</td>
<td>2.3M</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>95</td>
<td>90</td>
</tr>
<tr>
<td>RLPack</td>
<td>941K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>TELock v0.51</td>
<td>406K</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Upack v0.39</td>
<td>711K</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>41</td>
<td>1</td>
</tr>
</tbody>
</table>

The technique scales on significant traces
### Packer: Analysis results

<table>
<thead>
<tr>
<th>packers</th>
<th>trace len.</th>
<th>#proc</th>
<th>#th</th>
<th>#SMC</th>
<th>opaque predicates</th>
<th>call stack tampering</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACProtect v2.0</td>
<td>1.8M</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>159</td>
<td>48</td>
</tr>
<tr>
<td>ASPack v2.12</td>
<td>377K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>11</td>
</tr>
<tr>
<td>Crypter v1.12</td>
<td>1.1M</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>399</td>
<td>125</td>
</tr>
<tr>
<td>Expressor</td>
<td>635K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>FSG v2.0</td>
<td>68k</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Mew</td>
<td>59K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Crypter v1.12</td>
<td>1.1M</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>399</td>
<td>125</td>
</tr>
<tr>
<td>PE Lock</td>
<td>2.3M</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>95</td>
<td>90</td>
</tr>
<tr>
<td>RLPack</td>
<td>941K</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>46</td>
<td>14</td>
</tr>
<tr>
<td>TELock v0.51</td>
<td>406K</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Upack v0.39</td>
<td>711K</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>41</td>
<td>7</td>
</tr>
</tbody>
</table>

- Several have no such obfuscation, NeoLite, nPack, Packman, PE Compact ....
- Several packers still evade the DBI, Armadillo, BoxedApp, EP Protector, VMProtect....
- 3 reached the 10M instructions limit, Enigma, svk, Themida
Several have no such obfuscation, NeoLite, nPack, Packman, PE Compact ....
Several packers still evade the DBI, Armadillo, BoxedApp, EP Protector, VMProtect....
3 reached the 10M instructions limit, Enigma, svk, Themida

<table>
<thead>
<tr>
<th>packers</th>
<th>trace len.</th>
<th>#proc</th>
<th>#th</th>
<th>#SMC</th>
<th>opaque predicates</th>
<th>call stack tampering</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPProtect v2.0</td>
<td>1.8M</td>
<td>1</td>
<td></td>
<td></td>
<td>159</td>
<td>0</td>
</tr>
<tr>
<td>ASPack v2.12</td>
<td>377K</td>
<td>1</td>
<td></td>
<td></td>
<td>24</td>
<td>11</td>
</tr>
<tr>
<td>Crypter v1.12</td>
<td>1.1M</td>
<td>1</td>
<td>1</td>
<td></td>
<td>399</td>
<td>125</td>
</tr>
<tr>
<td>Expressor</td>
<td>635K</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>FSG v2.0</td>
<td>68K</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Mew</td>
<td>59K</td>
<td>1</td>
<td></td>
<td></td>
<td>28</td>
<td>6</td>
</tr>
<tr>
<td>PE Lock</td>
<td>2.3M</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>95</td>
<td>4</td>
</tr>
<tr>
<td>RLPack</td>
<td>941K</td>
<td>1</td>
<td></td>
<td></td>
<td>46</td>
<td>14</td>
</tr>
<tr>
<td>TELock v0.51</td>
<td>406K</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Upack v0.39</td>
<td>711K</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

The technique scales on significant traces
Many true positives. Some packers are using it intensively
Packers using ret to perform the final tail transition to the original entrypoint
Packer: Tricks and patterns found

OP in ACProtect

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1018f7a</td>
<td>js</td>
<td>0x1018f92</td>
</tr>
<tr>
<td>0x1018f7c</td>
<td>jns</td>
<td>0x1018f92</td>
</tr>
</tbody>
</table>

(and all possible variants ja/jbe, jp/jnp, jo/jno..)
**Packer:** Tricks and patterns found

### OP in ACProtect

- `0x1018f7a js 0x1018f92`
- `0x1018f7c jns 0x1018f92`

(and all possible variants `ja/jbe, jp/jnp, jo/jno..`)

### OP in Armadillo

- `0x10330ae xor ecx, ecx`
- `0x10330b0 jnz 0x10330ca`
## Packer: Tricks and patterns found

### OP in ACProtect

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Address</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1018f7a</td>
<td>js</td>
<td>0x1018f92</td>
<td></td>
</tr>
<tr>
<td>1018f7c</td>
<td>jns</td>
<td>0x1018f92</td>
<td></td>
</tr>
</tbody>
</table>

(and all possible variants
ja/jbe, jp/jnp, jo/jno..)

### OP in Armadillo

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Address</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>10330ae</td>
<td>xor</td>
<td>ecx, ecx</td>
<td></td>
</tr>
<tr>
<td>10330b0</td>
<td>jnz</td>
<td>0x10330ca</td>
<td></td>
</tr>
</tbody>
</table>

### CST in ACProtect

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Address</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001000</td>
<td>push</td>
<td>16793600</td>
<td></td>
</tr>
<tr>
<td>1001005</td>
<td>push</td>
<td>16781323</td>
<td></td>
</tr>
<tr>
<td>100100a</td>
<td>ret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100100b</td>
<td>ret</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Packer: Tricks and patterns found

**OP in ACProtect**

- `1018f7a js 0x1018f92`
- `1018f7c jns 0x1018f92`
  (and all possible variants ja/jbe, jp/jnp, jo/jno..)

**OP in Armadillo**

- `10330ae xor ecx, ecx`
- `10330b0 jnz 0x10330ca`

**CST in ACProtect**

- `1001000 push 16793600`
- `1001005 push 16781323`
- `100100a ret`
- `100100b ret`

**CST in ACProtect**

- `1004328 call 0x1004318`
- `1004318 add [esp], 9`
- `100431c ret`
**Packer: Tricks and patterns found**

**OP in ACProtect**

- 1018f7a js 0x1018f92
- 1018f7c jns 0x1018f92

(and all possible variants ja/jbe, jp/jnp, jo/jno..)

**OP in Armadillo**

- 10330ae xor ecx, ecx
- 10330b0 jnz 0x10330ca

**CST in ACProtect**

- 1001000 push 16793600
- 1001005 push 16781323
- 100100a ret
- 100100b ret

**CST in ASPack**

- 10043a9 mov [ebp+0x3a8], eax
- 10043af popa
- 10043b0 jnz 0x10043ba

**Enter SMC Layer 1**

- 10043ba push 0
- 10043bf ret

**CST in ASPack**

- 1004328 call 0x1004318
- 1004318 add [esp], 9
- 100431c ret
**Packer: Tricks and patterns found**

**OP in ACProtect**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1018f7a</td>
<td>js 0x1018f92</td>
<td>1018f7c</td>
<td>jns 0x1018f92</td>
</tr>
</tbody>
</table>

(and all possible variants ja/jbe, jp/jnp, jo/jno..)

**OP in Armadillo**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10330ae</td>
<td>xor ecx, ecx</td>
<td>10330b0</td>
<td>jnz 0x10330ca</td>
</tr>
</tbody>
</table>

**CST in ACProtect**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001000</td>
<td>push 16793600</td>
<td>1001005</td>
<td>push 16781323</td>
</tr>
<tr>
<td>100100a</td>
<td>ret</td>
<td>100100b</td>
<td>ret</td>
</tr>
</tbody>
</table>

**CST in ASPack**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10043a9</td>
<td>mov [ebp+0x3a8], eax</td>
<td>10043af</td>
<td>popa 0x10043bb</td>
</tr>
<tr>
<td>10043b0</td>
<td>jnz 0x10043ba</td>
<td>10043ba</td>
<td>push 0x10011d7</td>
</tr>
<tr>
<td>10043bf</td>
<td>ret</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enter SMC Layer 1 at runtime 0x10011d7
Packer: Tricks and patterns found

**OP in ACProtect**

- 0x1018f7a
  - js 0x1018f92
- 0x1018f7c
  - jns 0x1018f92

(and all possible variants ja/jbe, jp/jnp, jo/jno..)

**OP in Armadillo**

- 0x10330ae
  - xor ecx, ecx
- 0x10330b0
  - jnz 0x10330ca

**CST in ACProtect**

- 0x1004328
  - call 0x1004318
- 0x1004318
  - add [esp], 9
  - ret
  - 0x100431c
  - ret

**CST in ASPack**

- 0x10043a9
  - mov [ebp+0x3a8], eax
- 0x10043af
  - popa 0x10043bb

Enter SMC Layer 1

- 0x10043ba
  - push 0x10011d7
  - ret

**CST in ACProtect**

- 0x1001000
  - push 16793600
- 0x1001005
  - push 16781323
- 0x100100a
  - ret
- 0x100100b
  - ret

**OP (decoy) in ASPack**

- 0x10040fe
  - mov bl, 0x0
- 0x10041c0
  - cmp bl, 0x0
  - jnz 0x1004163

ZF = 0

- 0x1004163
  - jmp 0x100416d
  - [...]

ZF = 1

- 0x1004105
  - inc [ebp+0xec]
  - [...]

- 0x1004103
  - jnz 0x100416b

- 0x1004163
  - [...]

- 0x10043b0
  - jnz 0x10043ba

- 0x10043ba
  - push 0x10011d7
**Packer: Tricks and patterns found**

**OP in ACProtect**
- `1018f7a js 0x1018f92`
- `1018f7c jns 0x1018f92`
  (and all possible variants `ja/jbe, jp/jnp, jo/jno..`)

**OP in Armadillo**
- `10330ae xor ecx, ecx`
- `10330b0 jnz 0x10330ca`

**CST in ACProtect**
- `1001000 push 16793600`
- `1001005 push 16781323`
- `100100a ret`
- `100100b ret`

**CST in ASPack**
- `1004328 call 0x1004318`
- `1004318 add [esp], 9`
- `100431c ret`

**OP (decoy) in ASPack**
- `10040fe: mov bl, 0x1`
- `10041c0: cmp bl, 0x0`
- `1004103: jnz 0x1004163`

**CST in ASPack**
- `1004039 mov [ebp+0x3a8], eax`
- `1004af popa 0x10043bb`
  (at runtime)
- `10043b0 jnz 0x10043ba`

**Enter SMC Layer 1**
- `10043ba push 0x10011d7`
- `10043bf ret`

**ZF = 0**
- `1004163: jmp 0x100416d [...]

**ZF = 1**
- `1004105: inc [ebp+0xec] [...]

**0x10040ff at runtime**
- `0x10040fe: mov bl, 0x1`
X-Tunnel

A dive into the APT28 ciphering proxy
**Introduction:** Sednit / APT28 / Pawn Storm

**Nicknames:** APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm
Introduction: Sednit / APT28 / Pawn Storm

Nicknames: APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm

Alleged attacks:
- NATO, EU institutions [2015]
- German Parliament [2015]
- TV5 Monde (France) [2015]
- Political activists (Russia) [2015]
- DNC: Democratic National Committee (US) [2016]

Data collected from: ESET, Trend Micro, CrowdStrike ...
**Introduction:** Sednit / APT28 / Pawn Storm

**Nicknames:** APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm

**Alleged attacks:**
- NATO, EU institutions [2015]
- **German Parliament** [2015]
- TV5 Monde (France) [2015]
- Political activists (Russia)
- **DNC: Democratic National Committee (US)**
- Data collected from: ESET, Trend Micro, CrowdStrike...

**0-days used:**
- 2 Java
- 1 Windows (LPE) [CVE-2015-1701] (delivered via their exploit kit “sedkit” + existing exploits)
Introduction: Sednit / APT28 / Pawn Storm

Nicknames: APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm

Alleged attacks:
- NATO, EU institutions [2015]
- German Parliament [2015]
- TV5 Monde (France) [2015]
- Political activists (Russia) [2015]
- DNC: Democratic National Committee (US) [2016]

Data collected from: ESET, Trend Micro, CrowdStrike ...

0-days used:
- 2 Flash [CVE-2015-7645]
- 1 Office (RCE) [CVE-2015-2424]
- 2 Java [CVE-2015-2590]
- 1 Windows (LPE) [CVE-2015-4902] (delivered via their exploit kit “sedkit” + existing exploits)

Tools used:
- Droppers / Downloader
- X-Agent / X-tunnel
- Rootkit / Bootkit
- Mac OS X trojan (Komplex)
- USB C&C
**Introduction**: Sednit / APT28 / Pawn Storm

**Nicknames**: APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm

**Alleged attacks**:
- NATO, EU institutions [2015]
- German Parliament [2015]
- TV5 Monde (France) [2015]
- Political activists (Russia)
- DNC: Democratic National Committee (US)

Data collected from: ESET, Trend Micro, CrowdStrike ...

**0-days used**:
- 2 Flash [CVE-2015-7645]
- 1 Office (RCE) [CVE-2015-2424]
- 2 Java [CVE-2015-2590]
- 1 Windows (LPE) [CVE-2015-4902]
  (delivered via their exploit kit “sedkit” + existing exploits)

**Tools used**:
- Droppers / Downloader
- X-Agent / X-tunnel
- Rootkit / Bootkit
- Mac OS X trojan (Komplex)
- USB C&C

**Bonus 0-day**: Flash + Windows 10
(sandbox escape win32k.sys)
(disclosed by Google*)

*https://security.googleblog.com/2016/10/disclosing-vulnerabilities-to-protect.html
**X-Tunnel**

**What is it?**
Ciphering proxy allowing X-Agent(s) not able to reach the C&C directly to connect to it through X-Tunnel. (first seen 2013)

**Features**
Encapsulate any TCP-based traffic into a RC4 cipher stream embedded into a TLS connection.
# X-Tunnel

**What is it?**
Ciphering proxy allowing X-Agent(s) not able to reach the C&C directly to connect to it through X-Tunnel. (first seen 2013)

**Features**
Encapsulate any TCP-based traffic into a RC4 cipher stream embedded into a TLS connection.

**Samples**

<table>
<thead>
<tr>
<th></th>
<th>Sample #0</th>
<th>Sample #1</th>
<th>Sample #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash</td>
<td>42DEE3[...]</td>
<td>C637E0[...]</td>
<td>99B454[...]</td>
</tr>
<tr>
<td>Size</td>
<td>1.1 Mo</td>
<td>2.1 Mo</td>
<td>1.8 Mo</td>
</tr>
<tr>
<td>Creation date</td>
<td>25/06/2015</td>
<td>02/07/2015</td>
<td>02/11/2015</td>
</tr>
<tr>
<td>#functions</td>
<td>3039</td>
<td>3775</td>
<td>3488</td>
</tr>
<tr>
<td>#instructions (IDA)</td>
<td>231907</td>
<td>505008</td>
<td>434143</td>
</tr>
</tbody>
</table>

---

A huge thanks to ESET Montreal and especially to Joan Calvet ☺
X-Tunnel

What is it?
Ciphering proxy allowing X-Agent(s) not able to reach the C&C directly to connect to it through X-Tunnel. (first seen 2013)

Features
Encapsulate any TCP-based traffic into a RC4 cipher stream embedded into a TLS connection.

Samples

<table>
<thead>
<tr>
<th>Sample #0</th>
<th>Sample #1</th>
<th>Sample #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash</td>
<td>42DEE3[...]</td>
<td>C637E0[...]</td>
</tr>
<tr>
<td>Size</td>
<td>1.1 Mo</td>
<td>2.1 Mo</td>
</tr>
<tr>
<td>Creation date</td>
<td>25/06/2015</td>
<td>02/07/2015</td>
</tr>
<tr>
<td>#functions</td>
<td>3039</td>
<td>3775</td>
</tr>
<tr>
<td>#instructions (IDA)</td>
<td>231907</td>
<td>505008</td>
</tr>
</tbody>
</table>

A huge thanks to ESET Montreal and especially to Joan Calvet 🎉

widely obfuscated with opaque predicates
Can we remove the obfuscation?

Are there new functionalities?
Can we remove the obfuscation?

spoiler:

Are there new functionalities?
Can we remove the obfuscation?

Are there new functionalities?

spoiler: I DUNNO LOL

spoiler: SUPER HAPPY
X-Tunnel: Analysis

Goal: Detect, remove all OPs and extract a clean CFG of functions

Analysis context
fully static analysis

[no self-modification]
[need to connect C2C]
[need to wait clients]
Goal: Detect, remove all OPs and extract a clean CFG of functions

Analysis context:
- fully static analysis
- opaque predicate detection

- [no self-modification]
- [need to connect C2C]
- [need to wait clients]
- [with bb-DSE and IDASec]
Goal: Detect, remove all OPs and extract a clean CFG of functions

Analysis context
- fully static analysis

1. opaque predicate detection
   - [no self-modification]
   - [need to connect C2C]
   - [need to wait clients]

2. high-level predicate recovery
   - [with bb-DSE and IDASec]
   - [to identify predicates used]
X-Tunnel: Analysis

Goal: Detect, remove all OPs and extract a clean CFG of functions

Analysis context
fully static analysis

opaque predicate detection

high-level predicate recovery

dead and spurious instruction removal

1. [no self-modification]
   [need to connect C2C]
   [need to wait clients]

2. [with bb-DSE and IDASec]

3. [to identify predicates used]
   [with liveness propagation]
**X-Tunnel: Analysis**

**Goal:** Detect, remove all OPs and extract a clean CFG of functions

1. **Analysis context**
   - fully static analysis
   - opaque predicate detection
     - [no self-modification]
     - [need to connect C2C]
     - [need to wait clients]

2. **High-level predicate recovery**
   - [with bb-DSE and IDASec]
   - [to identify predicates used]

3. **Dead and spurious instruction removal**
   - [with liveness propagation]

4. **Reduced CFG extraction**
High-level predicate recovery (synthesis)

**Behavior:** Computes the dependency, generates the predicate (+ instructions involved in computation)

```
High-level predicate recovery

Behavior: Computes the dependency, generates the predicate (+ instructions involved in computation)

 CFG

mov esi, dword_5D7A84
mov edi, dword_5D7A80
jz loc_44D9FA

imul esi, esi
imul eax, esi, 7
dec eax
imul edi, edi
cmp eax, edi
jnz loc_44D922

SMT Formula

(define-fun esi2 (load32_at memory #x005d7a84))
(define-fun edi0 (load32_at memory #x005d7a80))
(assert (not (= ZF2 #b1)))

(define-fun esi3 (bvmul (esi2 esi2)))
(define-fun eax2 (bvmul (esi3 #x00000007)))
(define-fun eax3 (bvsub (eax2 #x00000001)))
(define-fun edi1 (bvmul (edi0 edi0)))
(define-fun res328 (bvsub (eax3 edi1)))
(define-fun ZF4 (bvcomp res328 #x00000000))
(assert (= ZF4 #b1))

((bvsub (bvmul (bvmul esi2 esi2) 7) 1) ≠ (bvmul edi0 edi0) → 7x^2 - 1 ≠ y^2
```
**Analysis: Results**

<table>
<thead>
<tr>
<th></th>
<th>#cond jmp</th>
<th>bb-DSE</th>
<th>Synthesis</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>C637 #1</td>
<td>34505</td>
<td>57m36</td>
<td>48m33</td>
<td>1h46m</td>
</tr>
<tr>
<td>99B4 #2</td>
<td>30147</td>
<td>50m59</td>
<td>40m54</td>
<td>1h31m</td>
</tr>
</tbody>
</table>

(only one path per conditional jump is analysed)
**Analysis: Results**

<table>
<thead>
<tr>
<th></th>
<th>#cond jmp</th>
<th>bb-DSE</th>
<th>Synthesis</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>C637 #1</td>
<td>34505</td>
<td>57m36</td>
<td>48m33</td>
<td>1h46m</td>
</tr>
<tr>
<td>99B4 #2</td>
<td>30147</td>
<td>50m59</td>
<td>40m54</td>
<td>1h31m</td>
</tr>
</tbody>
</table>

(only one path per conditional jump is analysed)

Only 2 different opaque predicates

\[ 7y^2 - 1 \neq x^2 \]

\[ \frac{2}{x^2 + 1} \neq y^2 + 3 \]

both present in the same proportions..

**Diagrams:**

- C637 #1: 35.2% Ok, 8.6% Opaque predicate, 3% False positive, 3% OP missed
- 99B4 #2: 33% Ok, 8.4% Opaque predicate, 3% False positive, 3% OP missed

**Notes:**
- Only 2 different opaque predicates
- **[Good candidate for signature?]**
Analysis: Obfuscation distribution

Goal: Compute the percentage of conditional jump obfuscated within a function
Analysis: Obfuscation distribution

Goal: Compute the percentage of conditional jump obfuscated within a function

Many not obfuscated functions (due to statically linked library OpenSSL...)
Analysis: Obfuscation distribution

Goal: Compute the percentage of conditional jump obfuscated within a function

Many not obfuscated functions (due to statically linked library OpenSSL...)

Allow to narrow the in-depth analysis on these functions (~500 more likely of interest)
## Analysis: Code coverage

Results of the liveness propagation and identification of spurious instructions

<table>
<thead>
<tr>
<th></th>
<th>C637 Sample #1</th>
<th>99B4 Sample #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Total instruction</td>
<td>505,008</td>
<td>434,143</td>
</tr>
<tr>
<td>#Alive</td>
<td>+279,483</td>
<td>+241,177</td>
</tr>
<tr>
<td>#Dead</td>
<td>-121,794</td>
<td>-113,764</td>
</tr>
<tr>
<td>#Spurious</td>
<td>-103,731</td>
<td>-79,202</td>
</tr>
<tr>
<td>#Delta with sample #0</td>
<td>47,576</td>
<td>9,270</td>
</tr>
</tbody>
</table>

In both samples the difference with the un-obfuscated binary is very low, and probably due to some noise.
Analysis: Reduced CFG extraction
Analysis: Reduced CFG extraction

Tagged CFG

- Alive
- Spurious
- Dead
Analysis: Reduced CFG extraction
Demo!

X-Tunnel deobfuscation
**X-Tunnel: Conclusion**

Manual checking of difference did not appeared to yield significant differences or any new functionalities...

**Obfuscation: Differences with O-LLVM** (like)
- some predicates have far dependencies (use local variables)
- some computation reuse between opaque predicates

**Next:**
- **in-depth graph similarity** (*Bindiff*) (to find new functionalities)
- integration as an IDA processor module (IDP)?

**For more:** Visiting the Bear Den
Joan Calvet, Jessy Campos, Thomas Dupuy

[RECON 2016][Botconf 2016]
Binsec Takeaways

Tip of what can be done with Binsec:
dynamic symbolic execution, abstract interpretation, simulation, optimizations, simplifications, on-the-fly value patching...

More is yet to come (still a young platform):
documentation, stabilized API, ARMv7, code flattening and VM deobfuscation...

Take part!

- Download it, try it, experiment it!
- Don’t hesitate contacting us for questions!

Open source and available at:
- Binsec+Pinsec: http://binsec.gforge.inria.fr
- IDASec: https://github.com/RobinDavid/idasec
Takeaways

- Backward-bounded DSE scales well
- Very good results on X-Tunnel (completely deobfuscated)
- Combining dynamic, static and symbolic is the way to go on obfuscated binaries
Thank you!

Q & A

Robin David
robin.david@riseup.net
@RobinDavid1

Sébastien Bardin
sebastien.bardin@cea.fr