black hat EUROPE 2016

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 C
  - BINSEC

FROM RESEARCH TO INDUSTRY
cea tech
list

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

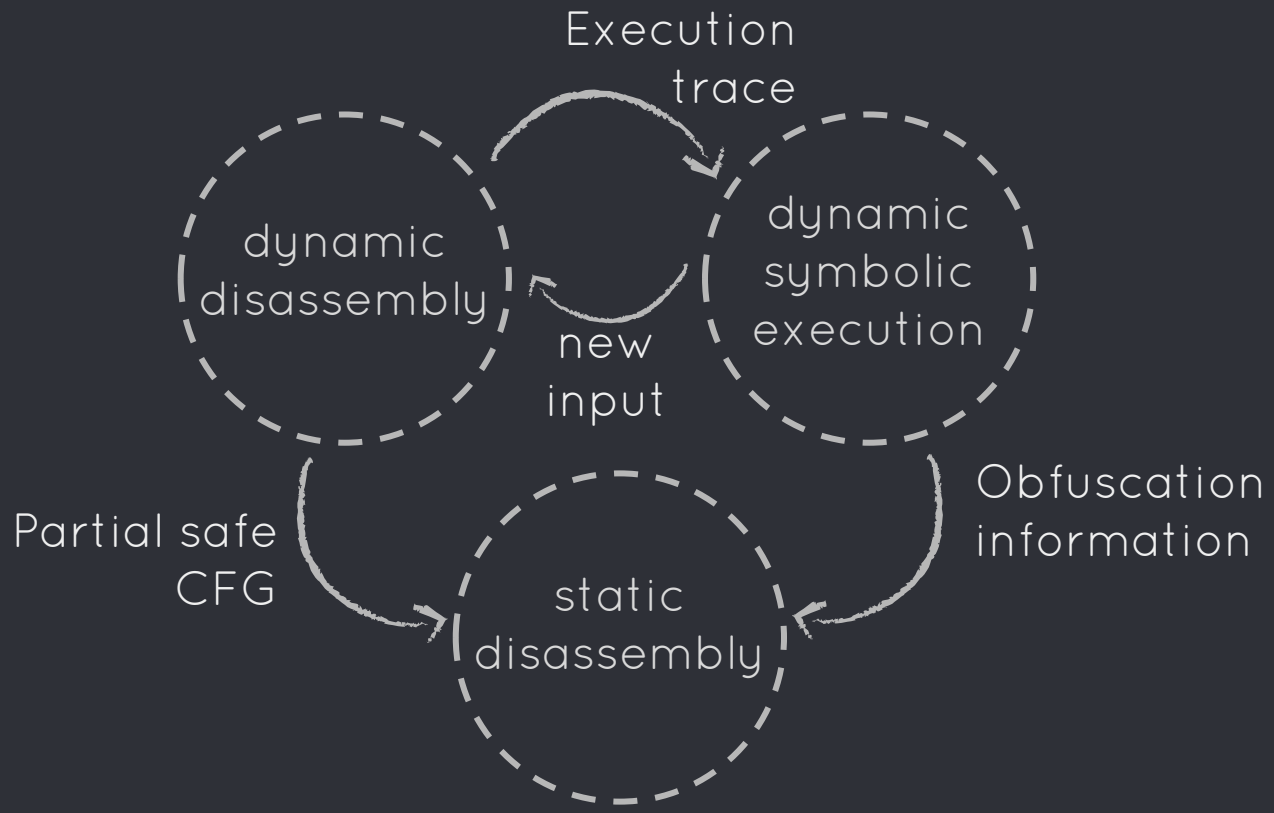
Our proposal

- A new symbolic method for infeasiblity-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



Takeaway message

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

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

1

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**

function calls, edges — strings, constants..

	Target		Against	
	Control	Data	Static	Dynamic
CFG flattening	●		●	
Jump encoding (direct → indirect/computed)	●		●	
Opaque predicates	●		●	
VM (Virtual-Machines)	●	●	●	●
Polymorphism (self-modification, resource ciphering)	●	●	●	
Call stack tampering	●		●	
Anti-debug/Anti-tampering	●	●		●
Signal/Exception	●		●	

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

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

```
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>
```

Corollary, the dead branch allows to:

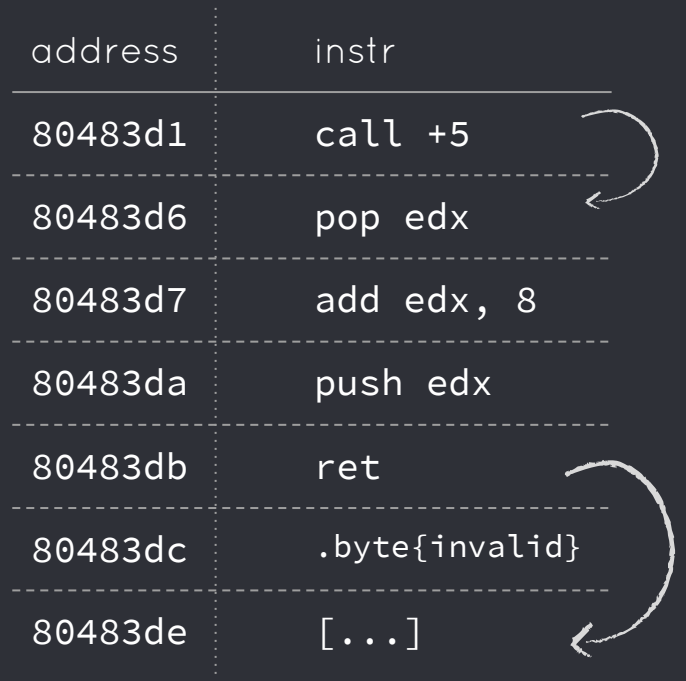
- Grow the code (artificially)
- Drown the genuine code

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

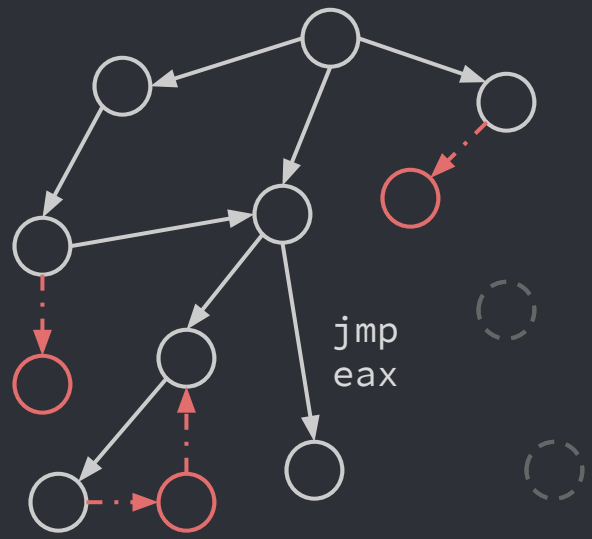
Standard approaches

	static	dynamic
scale	●	●
robust *(obfuscation)*	●	●
correct	●	●
complete	●	●

Disassembly

Notations

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



Standard approaches

- Static disassembly

	static	dynamic
scale	●	●
robust *(obfuscation)*	●	●
correct	●	●
complete	●	●

dynamic jump

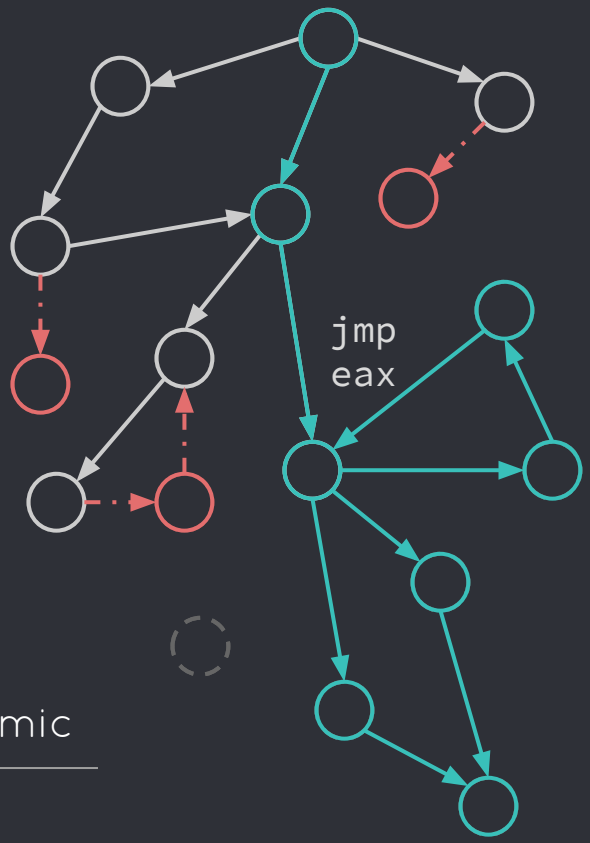
Disassembly

Notations

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

Standard approaches

- Static disassembly
- Dynamic disassembly



	static	dynamic
scale	●	●
robust *(obfuscation)*	●	●
correct	●	●
complete	●	●

dynamic jump

input dependent

2

Dynamic Symbolic Execution

a.k.a Concolic Execution

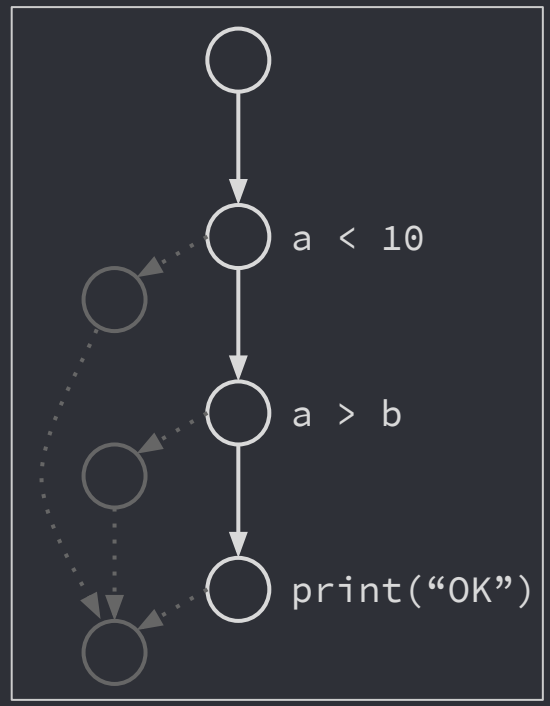
Symbolic Execution

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)

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



Formula:
$a < 10 \wedge a > b$

Solution:
a=5, b=1

(using SMT solvers)

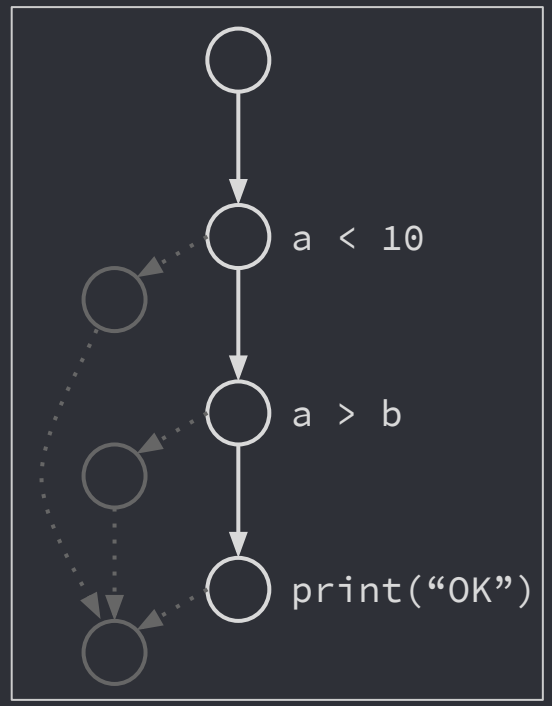
Symbolic Execution

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)

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



Formula:
$a < 10 \wedge a > b$

Solution:
a=5, b=1

(using SMT solvers)

Dynamic Symbolic Execution: works on a dynamic path

Why use DSE?

- Obfuscation alters the syntax but keeps the semantic

- DSE finds new paths

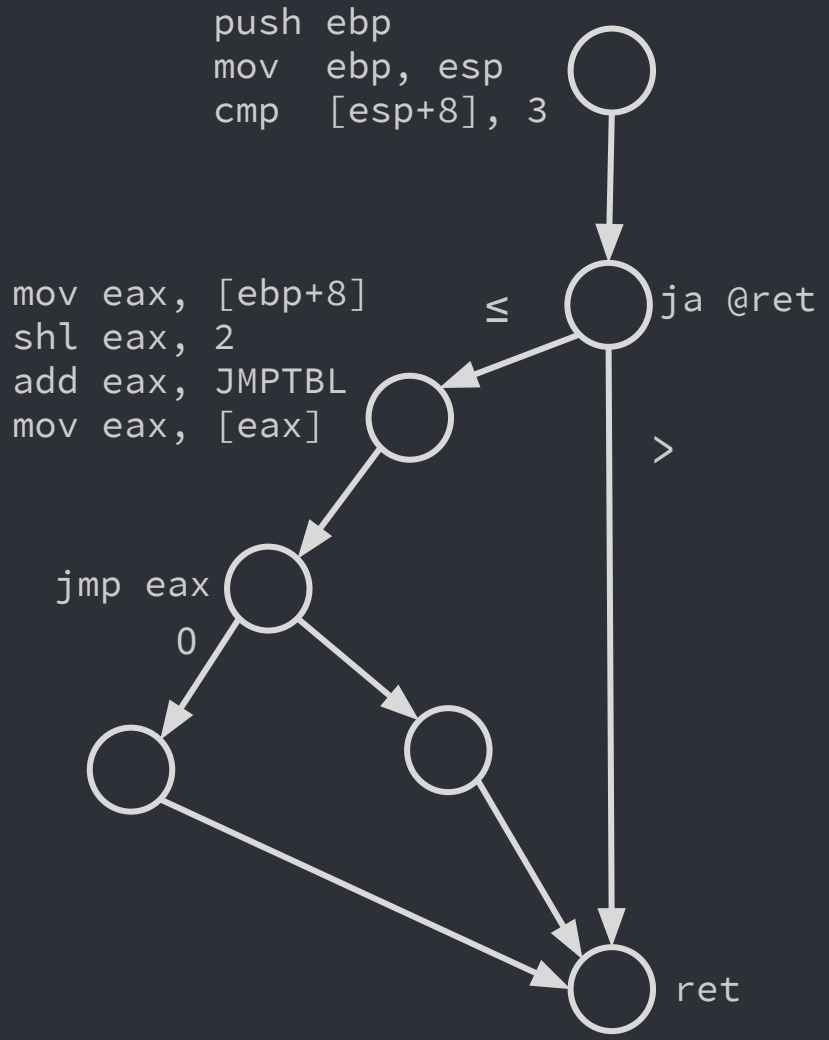
DSE on a switch

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
```



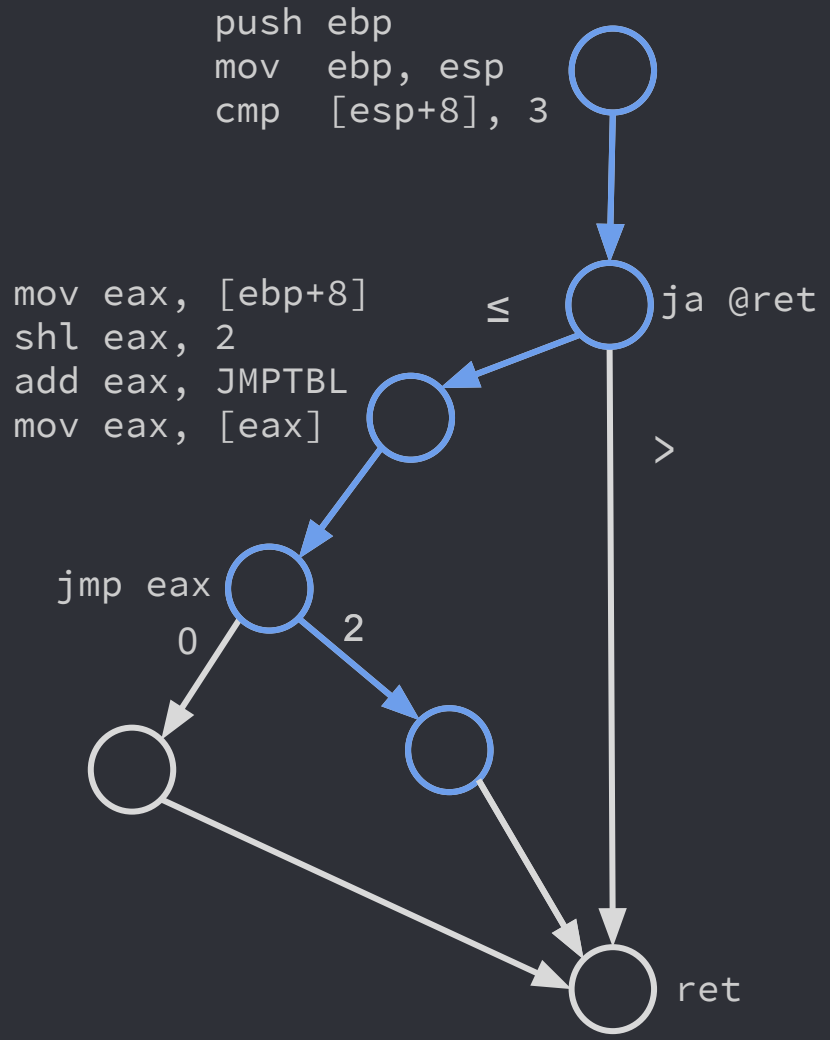
DSE on a switch

Source Code (C)

```
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

Symbolic Execution (input:esp, ebp, memory)
@[esp] := ebp
ebp1 := esp
@[ebp1+8] < 3
eax1 := @[esp+8]
eax2 := eax1 << 2
eax3 := eax2 + JMPTBL
eax4 := @[eax3]
eax4 == 2



Path predicate φ :

@[ebp1+8] < 3 ∧ eax4 == 2

@[esp+8] < 3 ∧ @[(@[esp+8]≪ 2) + JMPTBL] == 2

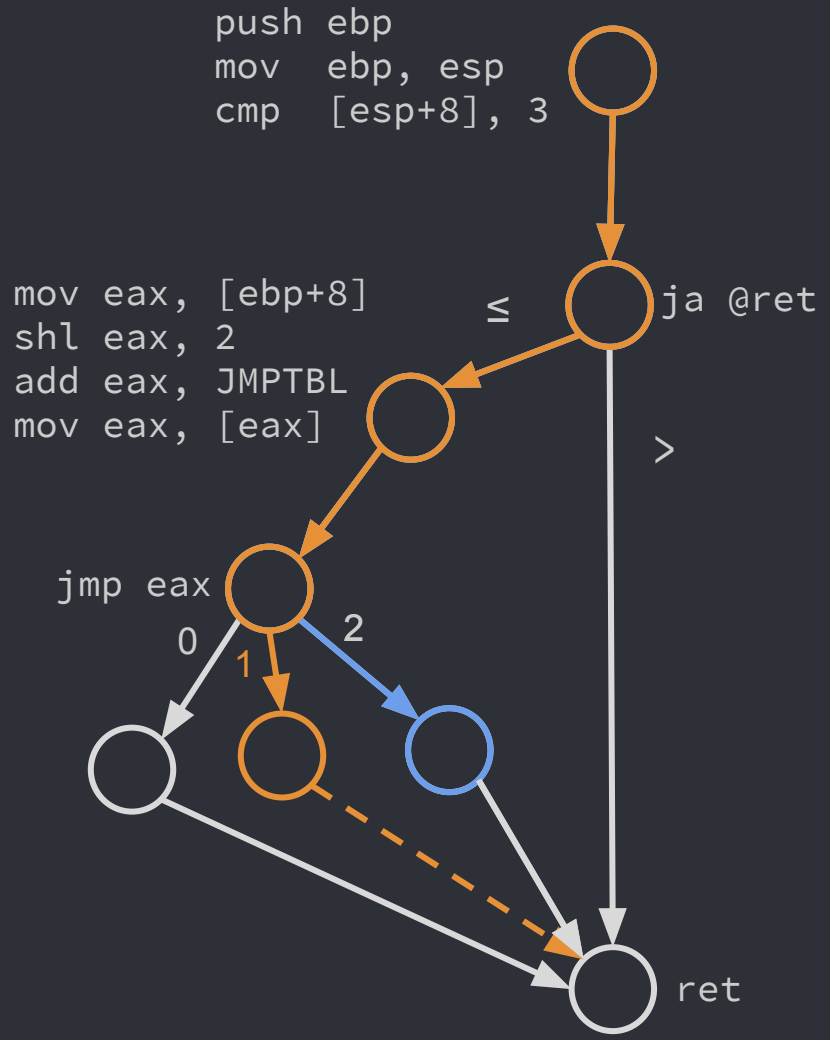
DSE on a switch

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
[...]
ret

Symbolic Execution (input:esp, ebp, memory)
@[esp] := ebp
ebp1 := esp
@[ebp1+8] < 3
eax1 := @[esp+8]
eax2 := eax1 << 2
eax3 := eax2 + JMPTBL
eax4 := @[eax3]
eax4 == 2



Path predicate φ :

@[ebp1+8] < 3 ∧ eax4 ≠ [0,2]

@[esp+8] < 3 ∧ @[(@[esp+8]≪ 2) + JMPTBL] ≠ [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

	static	dynamic	DSE (symbolic)
scale	●	●	●
robust *(obfuscation)*	●	●	●
correct	●	●	●
complete *(coverage)*	●	●	●

The challenge for DSE is to make it scale on huge path length and to cover all paths...

Dynamic and DSE allow to check feasibility properties

- find new targets for dynamic jump
- cover a new branch

What if instead we want to check infeasibility properties?

- no any other target for dynamic jump
- opaque predicates

standard DSE and dynamic analysis
not adapted

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**

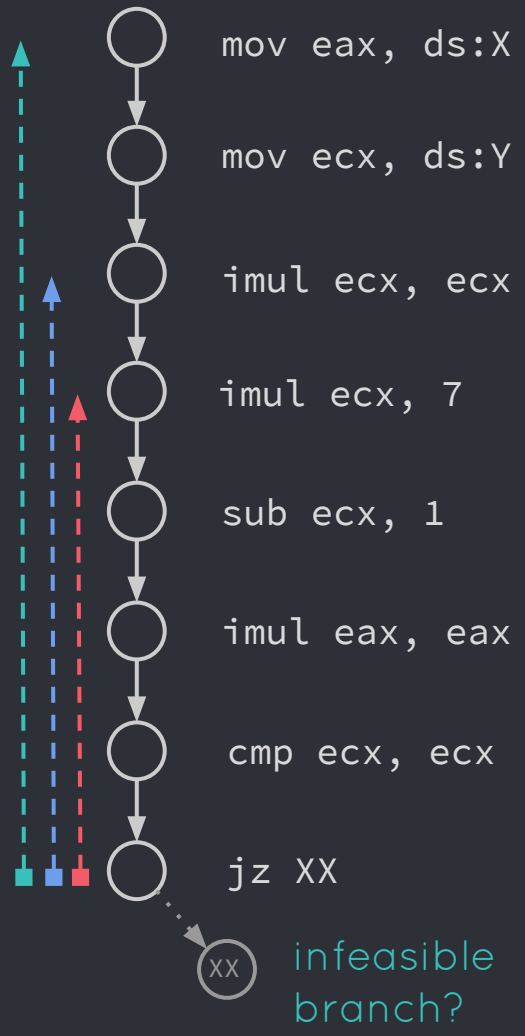
  (still feasible w.r.t. ecx, eax)

- **minimal**

  (backtrack enough constraints to prove the infeasibility)

- **complete**

  (backtrack all dependencies)

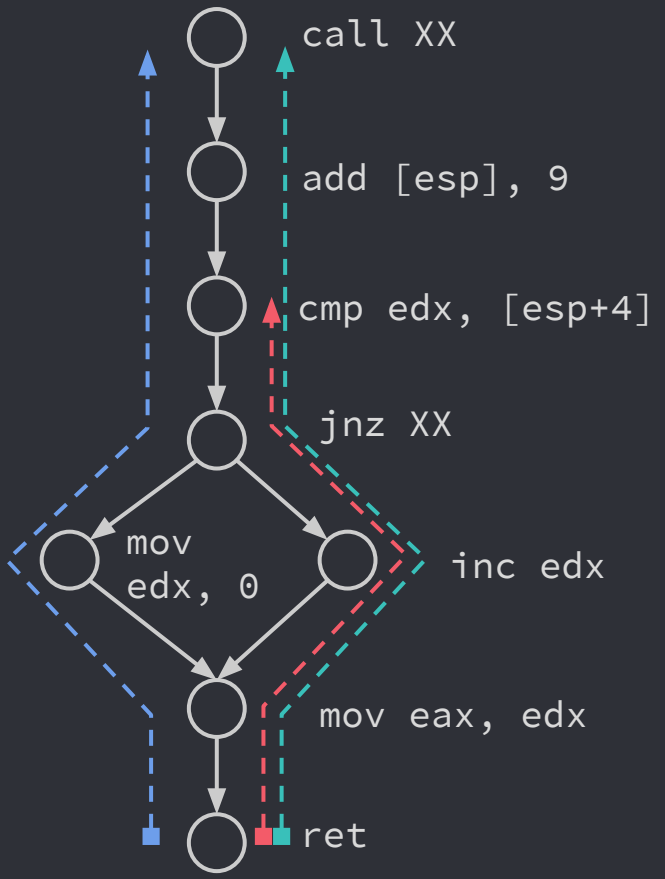


bb-DSE: Example of a call stack tampering

Goal

Check that the return address cannot be tampered by the function

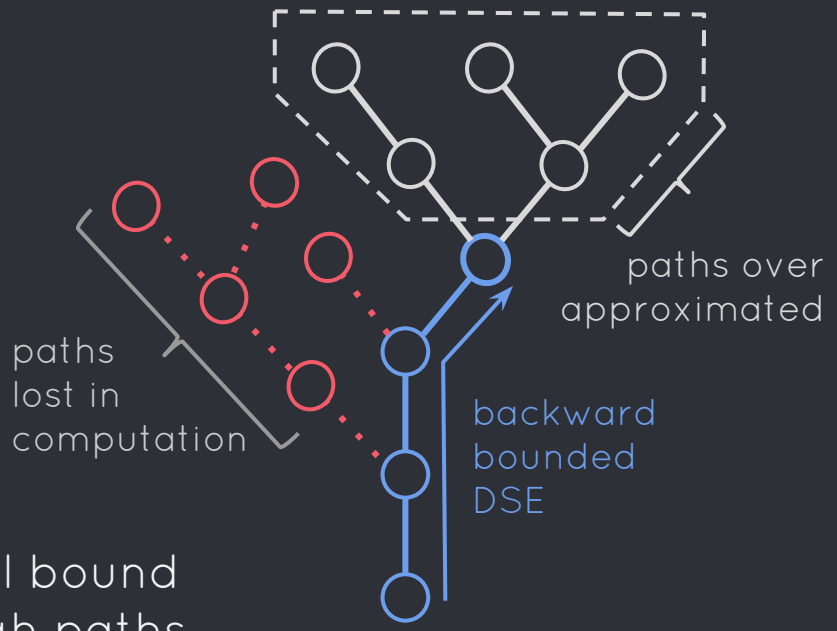
- ■ **false negative**: miss the tampering (too small bound)
- ■ **correct**: find the tampering
- ■+■ **complete**: validate the tampering for all paths



Backward-Bounded DSE

Summary

- backward: infeasibility
- bounded reasoning: scale
- adaptable bound (for the need)
- dynamic: robustness
  (hence false positive)



Shortcomings

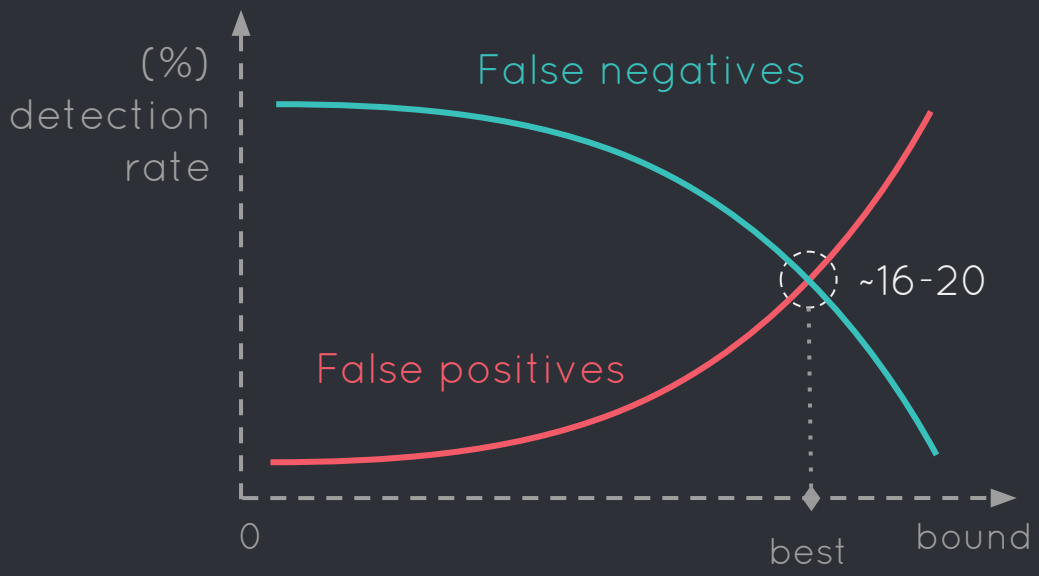
- False negative (FN): too small bound
- False positive (FP): not enough paths

	(forward) DSE	bb-DSE
feasibility queries	●	●
infeasibility queries	●	●
scale	●	●

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)

empiric results obtained through benchmarking

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

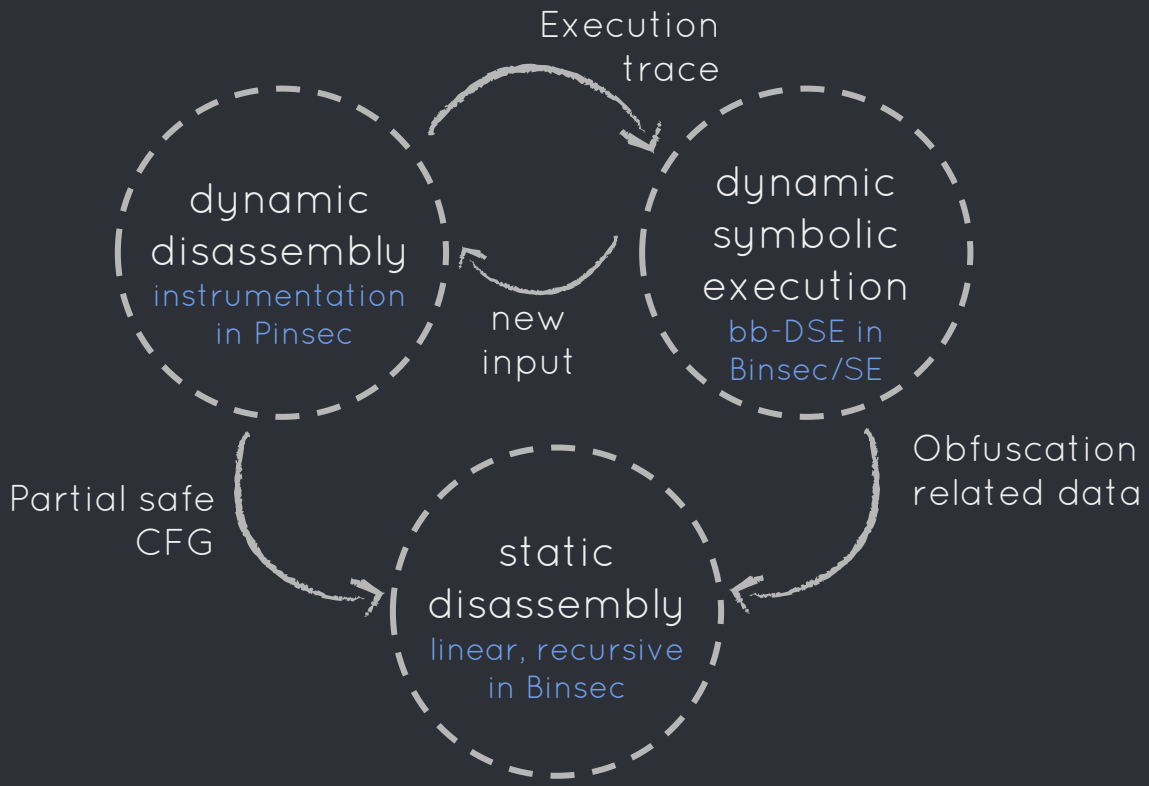
4

Combination

Intertwining Dynamic, Static and Symbolic

Combination: Principles

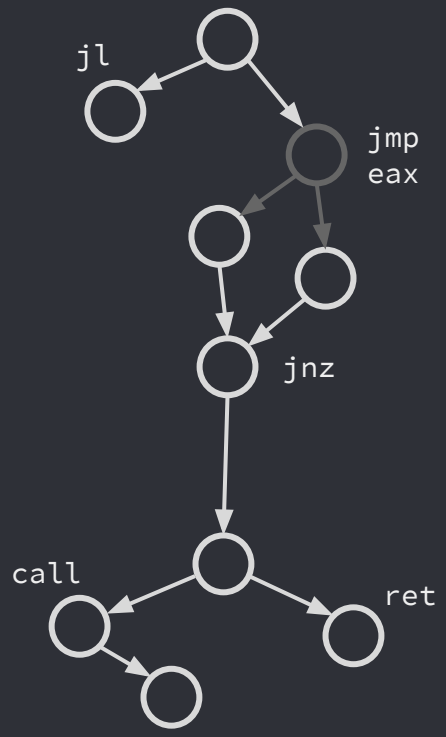
Goal: Obtaining a safer and more precise disassembly handling several obfuscation constructs.



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

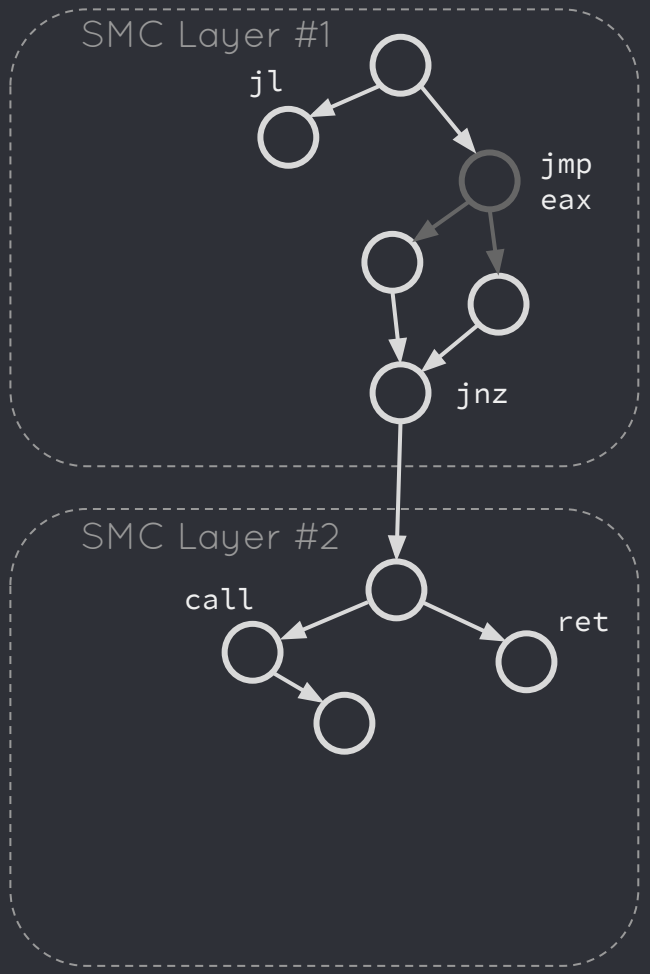
Combination: Application

- ■ + ■ safe dynamic disassembly with dynamic jumps



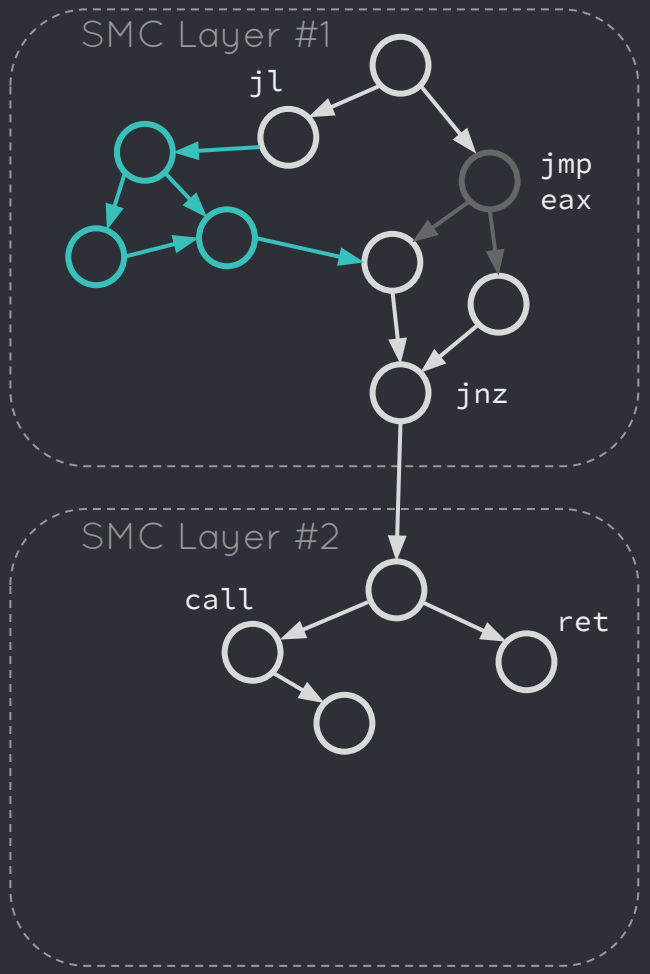
Combination: Application

- ■ + ■ safe dynamic disassembly with dynamic jumps
- ⬚ multiple self-modification segmentation



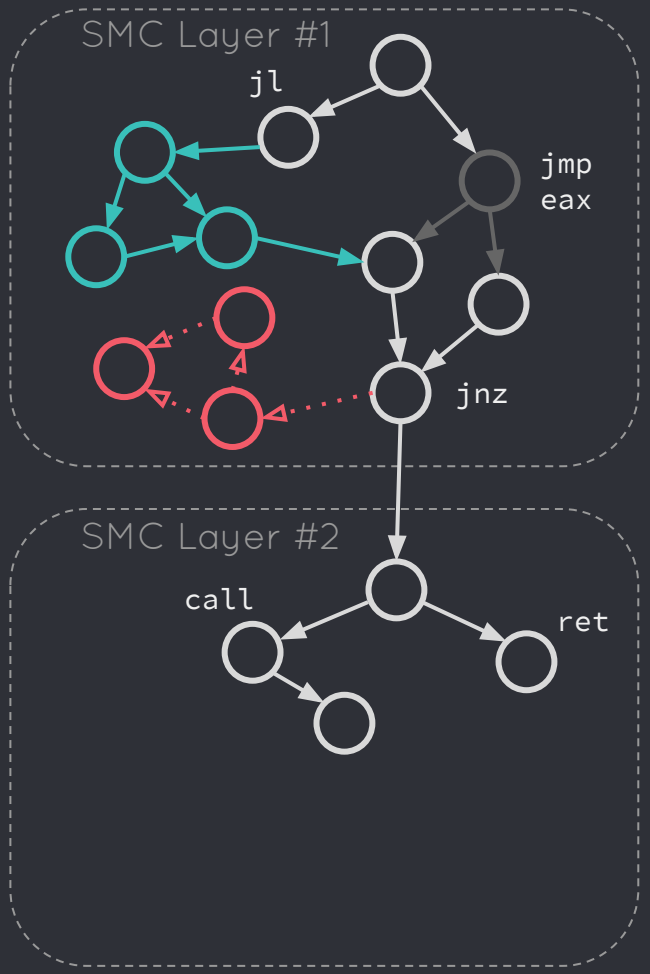
Combination: Application

- ■ + ■ safe dynamic disassembly with dynamic jumps
- ⬚ multiple self-modification segmentation
- ■ enlarge partial CFG on genuine conditional jump



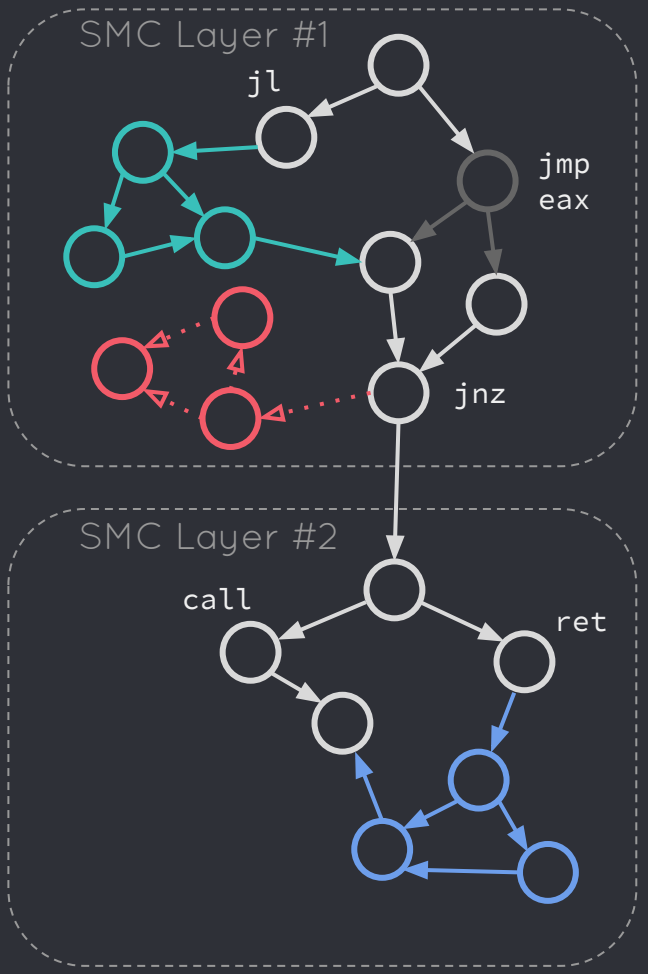
Combination: Application

- ■ + ■ 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



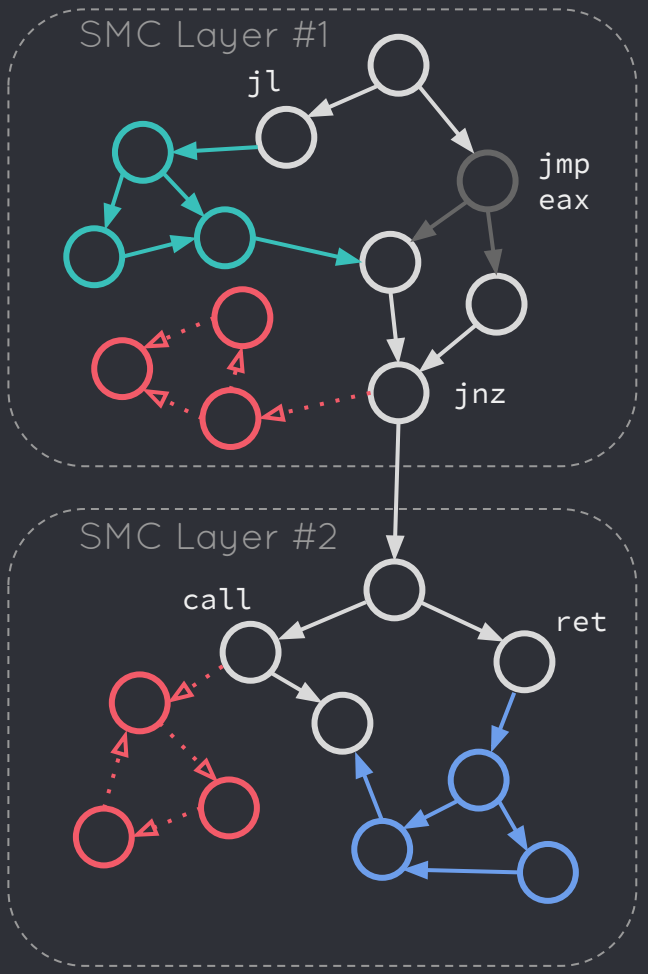
Combination: Application

- ■ + ■ 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

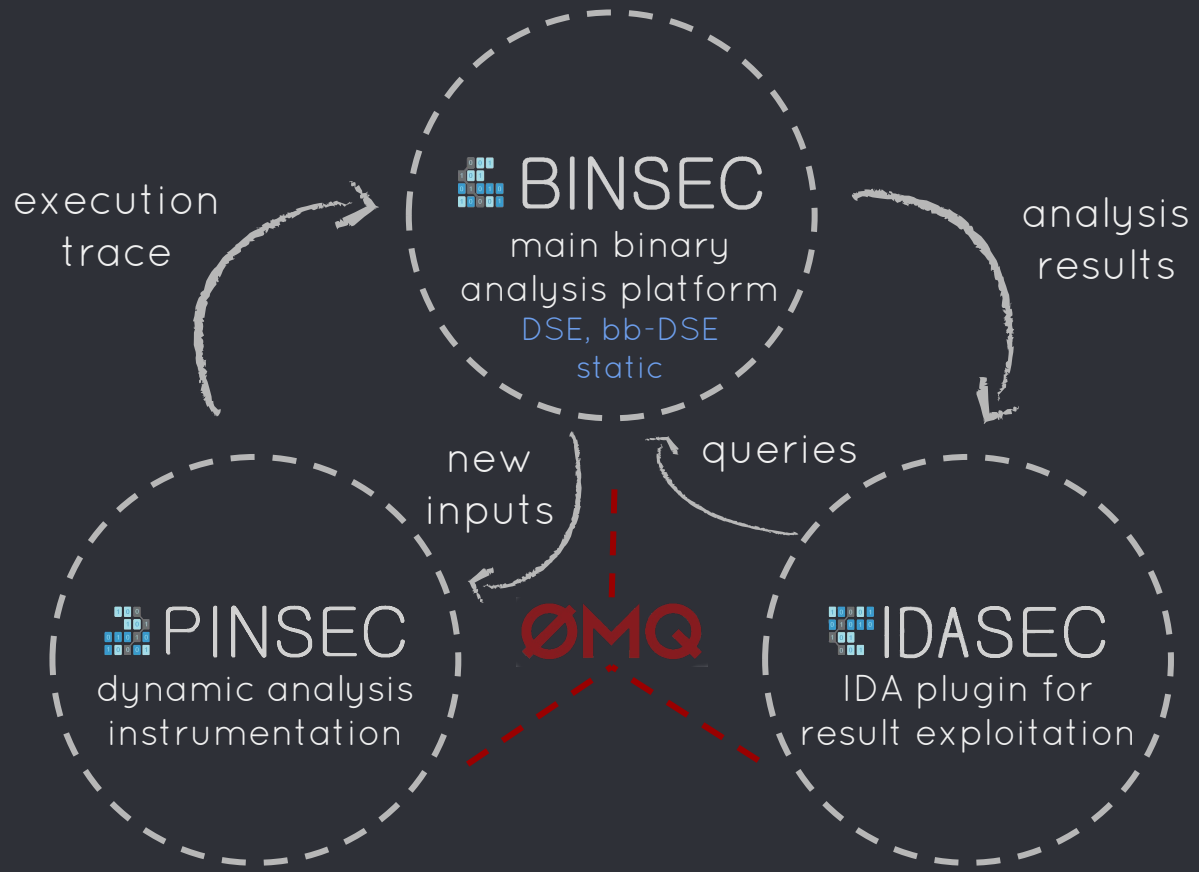
- ■ + ■ safe dynamic disassembly with dynamic jumps
- ⬚ multiple self-modification segmentation
- ■ enlarge partial CFG on genuine conditional jump
- ■ do not disassemble dead branch of opaque predicate
- ■ disassemble the target of tampered ret
- ■ do not disassemble the return site of tampered ret



5

BINSEC

Binsec platform architecture



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

- **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 ....

IDASEC

Python plugin for IDA (from 6.4)

Goal:

- triggering analyses remotly 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**

6

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

packers	trace len.	#proc	#th	#SMC	opaque predicates OK	opaque predicates OP	call stack tampering OK	call stack tampering tamper
ACProtect v2.0	1.8M	1	1	4	83	159	0	48
ASPack v2.12	377K	1	1	2	168	24	11	6
Crypter v1.12	1.1M	1	1	1	399	24	125	78
Expressor	635K	1	1	1	81	8	14	0
FSG v2.0	68k	1	1	1	24	1	6	0
Mew	59K	1	1	1	28	1	6	1
PE Lock	2.3M	1	1	6	95	90	4	3
RLPack	941K	1	1	1	46	2	14	0
TELock v0.51	406K	1	1	5	5	2	3	1
Upack v0.39	711K	1	1	2	41	1	7	1

- 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

Packer: Analysis results

packers	trace len.	#proc	#th	#SMC	opaque predicates OK	opaque predicates OP	call stack tampering OK	call stack tampering tamper
ACProtect v2.0	1.8M	1				159	0	48
ASPack v2.12	377K	1				24	11	6
Crypter v1.12	1.1M	1	1	1	399	24	125	78
Expressor	635K	1	1	1	81	8	14	0
FSG v2.0	68k	1	1	1	24	1	6	0
Mew	59K	1	1	1	28	1	6	1
PE Lock	2.3M	1	1	6	95	90	4	3
RLPack	941K	1	1	1	46	2	14	0
TELock v0.51	406K	1	1	5	5	2	3	1
Upack v0.39	711K	1	1	2	41	1	7	1

The technique scales on significant traces

- 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

Packer: Analysis results

packers	trace len.	#proc	#th	#SMC	opaque predicates OK	opaque predicates OP	call stack tampering OK	call stack tampering tamper
ACProtect v2.0	1.8M	1				159	0	48
ASPack v2.12	377K	1				24	11	6
Crypter v1.12	1.1M	1	1	1	399	24	125	78
Expressor	635K	1	1	1			14	0
FSG v2.0	68k	1	1	1			6	0
Mew	59K	1	1	1	28	1	6	1
PE Lock	2.3M	1	1	6	95	90	4	3
RLPack	941K	1	1	1	46	2	14	0
TELock v0.51	406K	1	1	5	5	2	3	1
Upack v0.39	711K	1	1	2	41	1	7	1

The technique scales on significant traces

Many true positives. Some packers are using it intensively

- 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

Packer: Analysis results

packers	trace len.	#proc	#th	#SMC	opaque predicates OK	opaque predicates OP	call stack tampering OK	call stack tampering tamper
ACProtect v2.0	1.8M	1				159	0	48
ASPack v2.12	377K	1				24	11	6
Crypter v1.12	1.1M	1	1	1	399	24	125	78
Expressor	635K	1	1	1			14	0
FSG v2.0	68k	1	1	1			6	0
Mew	59K	1	1	1	28	1	6	1
PE Lock	2.3M	1	1	6	95	90	4	3
RLPack	941K	1	1	1	46	2	14	0
TELock v0.51	406K	1	1	5			3	1
Upack v0.39	711K	1	1	2			7	1

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

- 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

Packer: Tricks and patterns found

OP in ACProtect

1018f7a	js	0x1018f92
1018f7c	jns	0x1018f92

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

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
```

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
```

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 ACProtect

```
1004328   call 0x1004318
1004318   add  [esp], 9
100431c   ret
```

CST in ASPack

```
10043a9       mov  [ebp+0x3a8], eax
10043af       popa
10043b0       jnz  0x10043ba
        Enter SMC Layer 1
10043ba       push 0
10043bf       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 ACProtect

```
1004328   call 0x1004318
1004318   add  [esp], 9
100431c   ret
```

CST in ASPack

```
10043a9   mov  [ebp+0x3a8], eax
10043af   popa   0x10043bb
                 at runtime
10043b0   jnz  0x10043ba
        Enter SMC Layer 1
10043ba   push 0x10011d7
10043bf   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
```

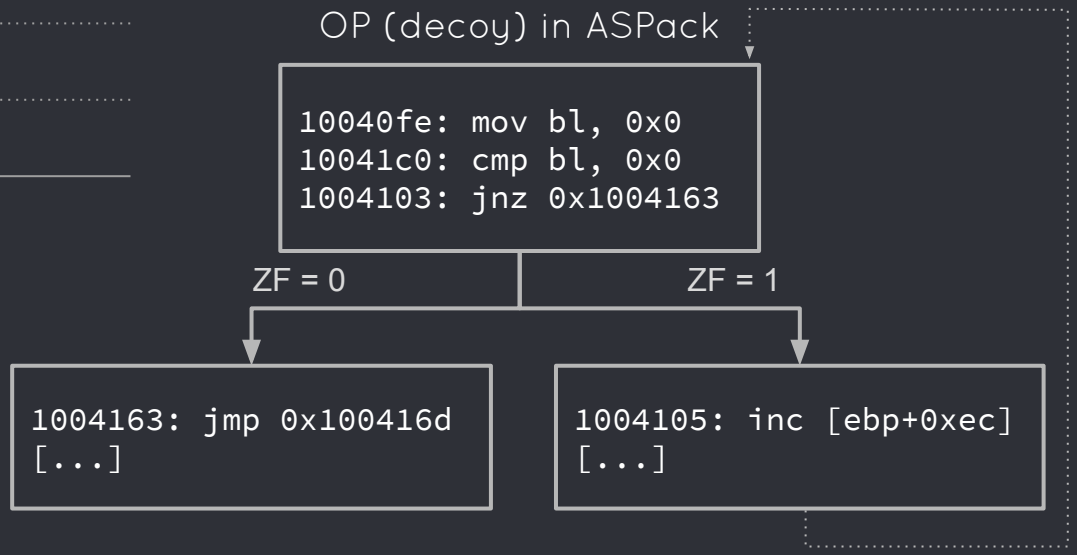
CST in ACProtect

```
1004328    call 0x1004318
1004318    add  [esp], 9
100431c    ret
```

CST in ASPack

```
10043a9        mov  [ebp+0x3a8], eax
10043af        popa   0x10043bb
                      at runtime
10043b0        jnz  0x10043ba
        Enter SMC Layer 1
10043ba        push 0x10011d7
10043bf        ret
```

OP (decoy) in ASPack



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
```

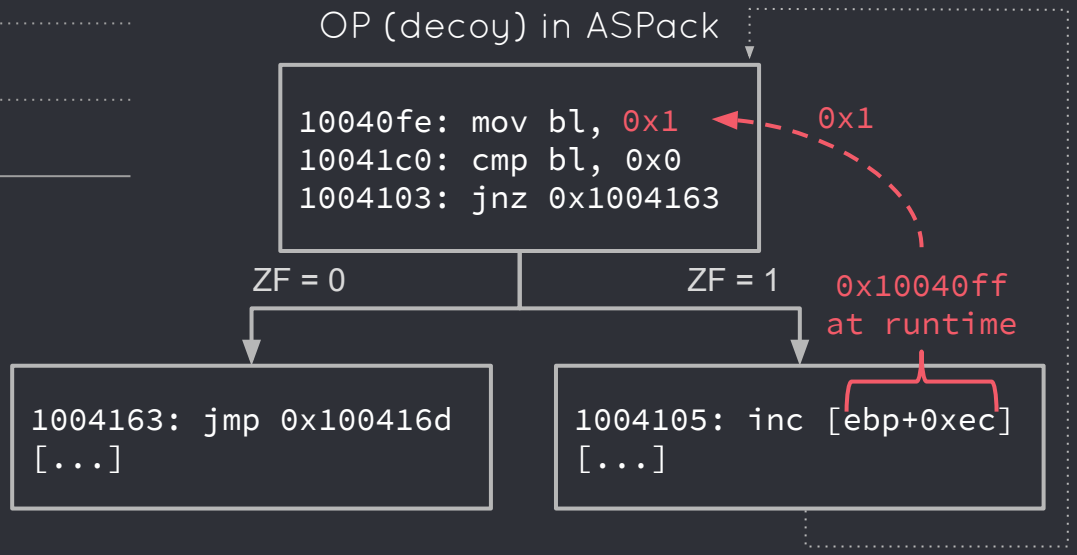
CST in ACProtect

```
1004328    call 0x1004318
1004318    add  [esp], 9
100431c    ret
```

CST in ASPack

```
10043a9        mov  [ebp+0x3a8], eax
10043af        popa   0x10043bb
                      at runtime
10043b0        jnz  0x10043ba
        Enter SMC Layer 1
10043ba        push 0x10011d7
10043bf        ret
```

OP (decoy) in ASPack



7

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)
- **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) [2016]

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

0-days used:

- 2 Flash [CVE-2015-7645] [CVE-2015-3043]
- 1 Office (RCE) [CVE-2015-2424]
- 2 Java [CVE-2015-2590] [CVE-2015-4902]
- 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)
- **DNC: Democratic National Committee** (US) [2016]

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

0-days used:

- 2 Flash [CVE-2015-7645] [CVE-2015-3043]
- 1 Office (RCE) [CVE-2015-2424]
- 2 Java [CVE-2015-2590] [CVE-2015-4902]
- 1 Windows (LPE) [CVE-2015-1701]

(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) [2016]

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

0-days used:

- 2 Flash [CVE-2015-7645] [CVE-2015-3043]
- 1 Office (RCE) [CVE-2015-2424]
- 2 Java [CVE-2015-2590] [CVE-2015-4902]
- 1 Windows (LPE) [CVE-2015-1701]

(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

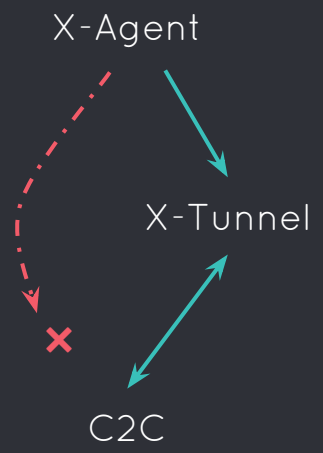
X-Tunnel

What is it ?

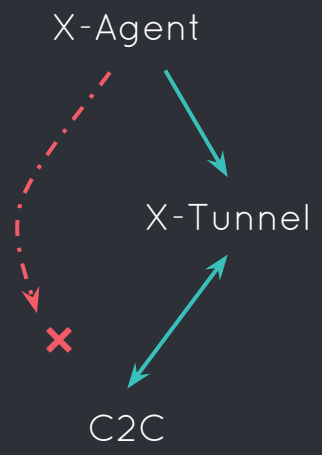
Ciphering proxy allowing X-Agent(s) not able to reach the C&C directly to connect to it through X-Tunnel. (first seen 2013)

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

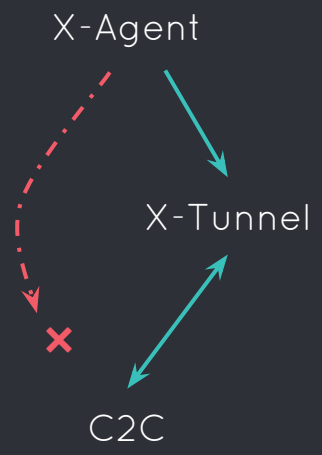
Encapsulate any TCP-based traffic into a RC4 cipher stream embedded into a TLS connection.

Samples

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

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

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

widely obfuscated with opaque predicates

A huge thanks to ESET Montreal and especially to Joan Calvet

Can we remove the obfuscation?

Are there new functionalities?

Can we remove the obfuscation?

spoiler:

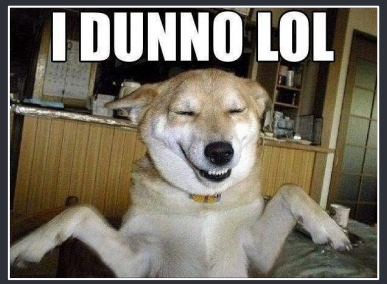
Are there new functionalities?

Can we remove the obfuscation?

spoiler:

Are there new functionalities?

spoiler:



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]

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]

opaque predicate detection

[with bb-DSE and IDASec]

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]

1 opaque predicate detection

[with bb-DSE and IDASec]

2 high-level predicate recovery

[to identify predicates used]

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]

1. opaque predicate detection — [with bb-DSE and IDASec]
2. high-level predicate recovery — [to identify predicates used]
3. dead and spurious instruction removal — [with liveness propagation]

X-Tunnel: Analysis

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

Analysis context
fully static analysis

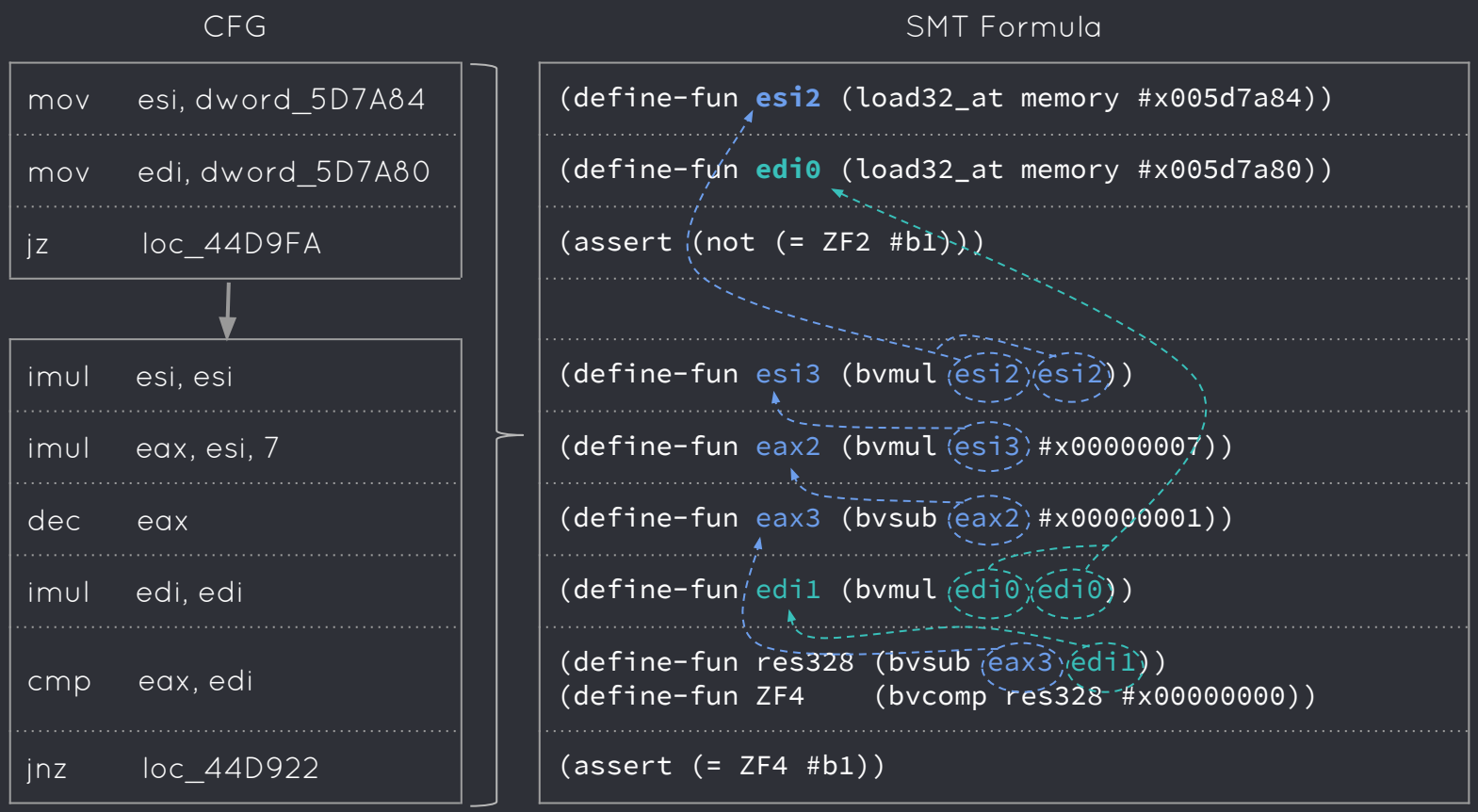
[no self-modification]
[need to connect C2C]
[need to wait clients]

1. opaque predicate detection — [with bb-DSE and IDASec]
2. high-level predicate recovery — [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)

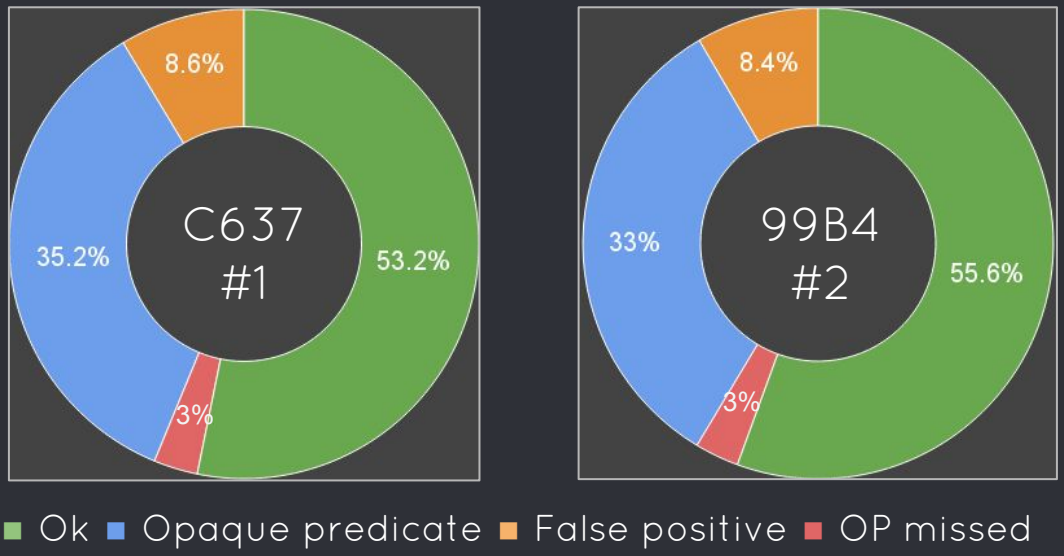


((bvsub (bvmul (bvmul esi2 esi2) 7) 1) ≠ (bvmul edi0 edi0) ↦ $7x^2 - 1 \neq y^2$

Analysis: Results

	#cond jmp	bb-DSE	Synthesis	Total
C637 #1	34505	57m36	48m33	1h46m
99B4 #2	30147	50m59	40m54	1h31m

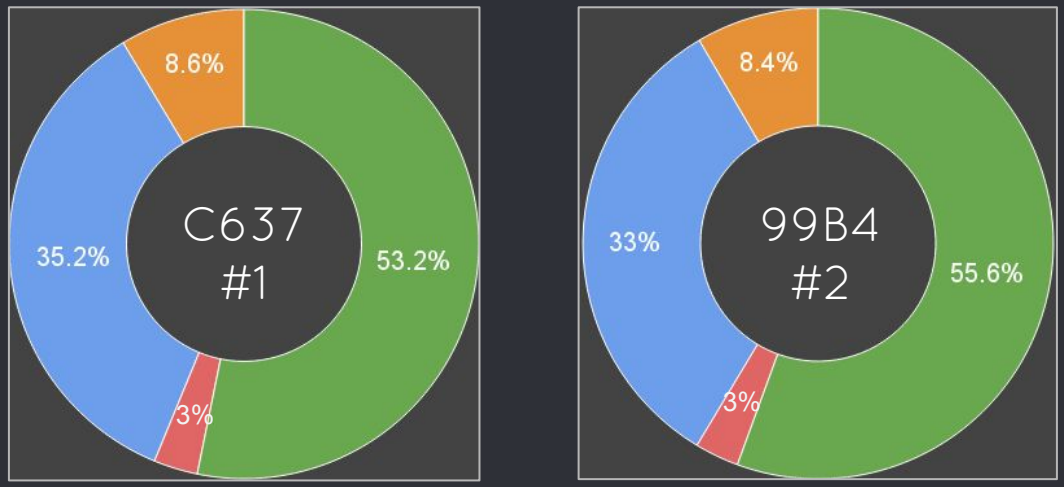
(only one path per conditional jump is analysed)



Analysis: Results

	#cond jmp	bb-DSE	Synthesis	Total
C637 #1	34505	57m36	48m33	1h46m
99B4 #2	30147	50m59	40m54	1h31m

(only one path per conditional jump is analysed)



good candidate for signature ?

Only 2 different opaque predicates

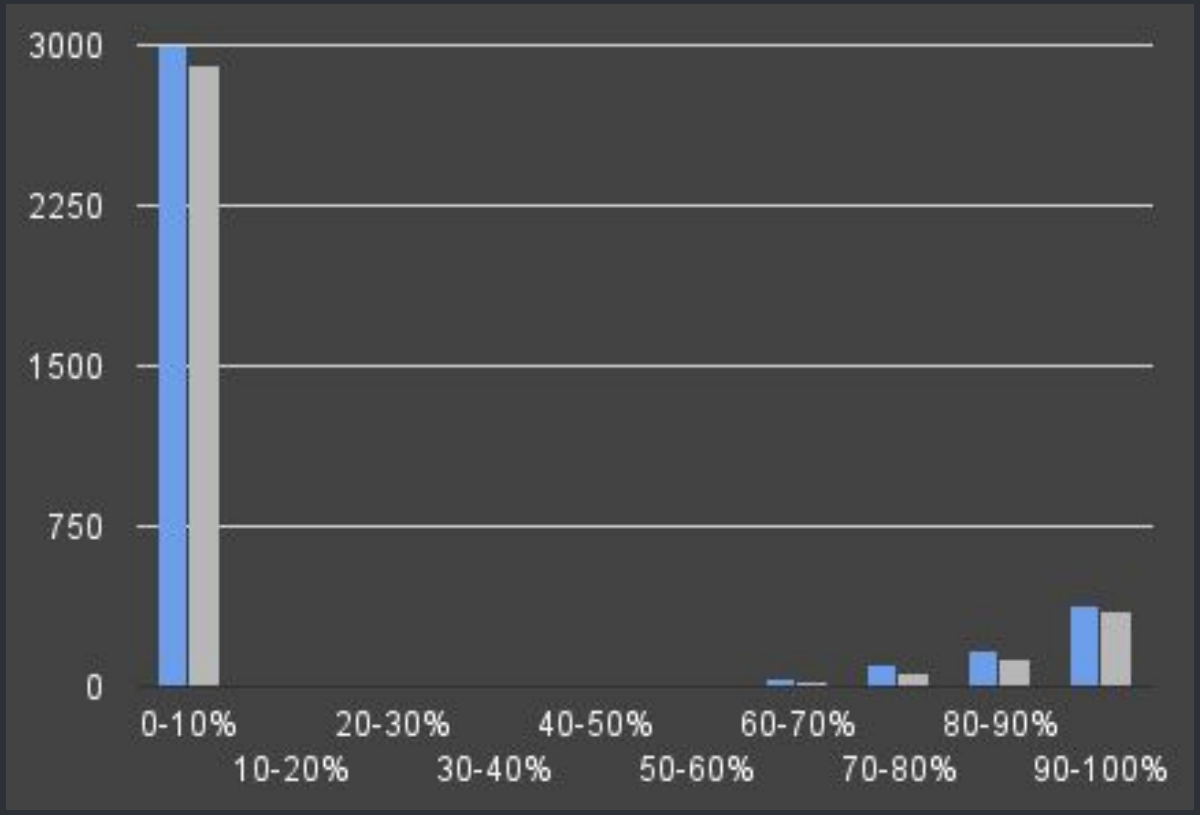
$$7y^2 - 1 \neq x^2 \qquad \frac{2}{x^2+1} \neq y^2 + 3$$

unseen elsewhere

both present in the same proportions..

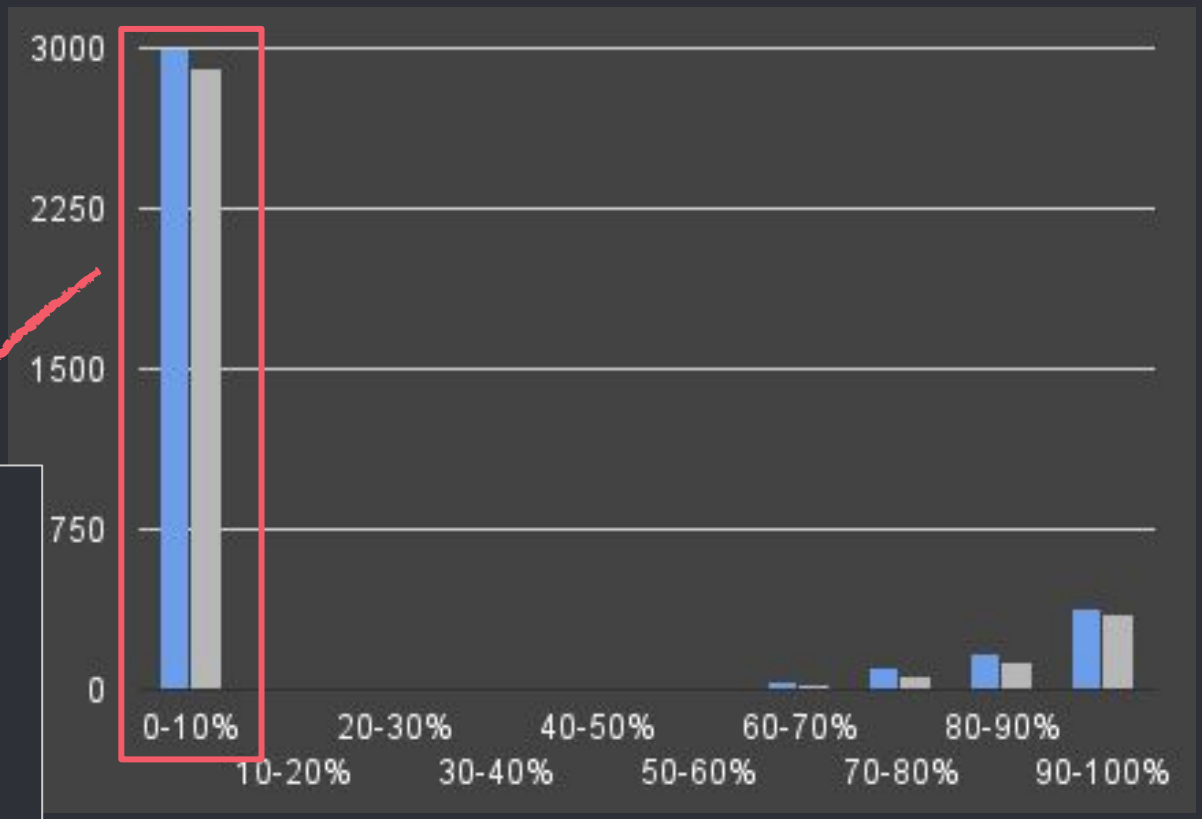
Analysis: Obfuscation distribution

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



Analysis: Obfuscation distribution

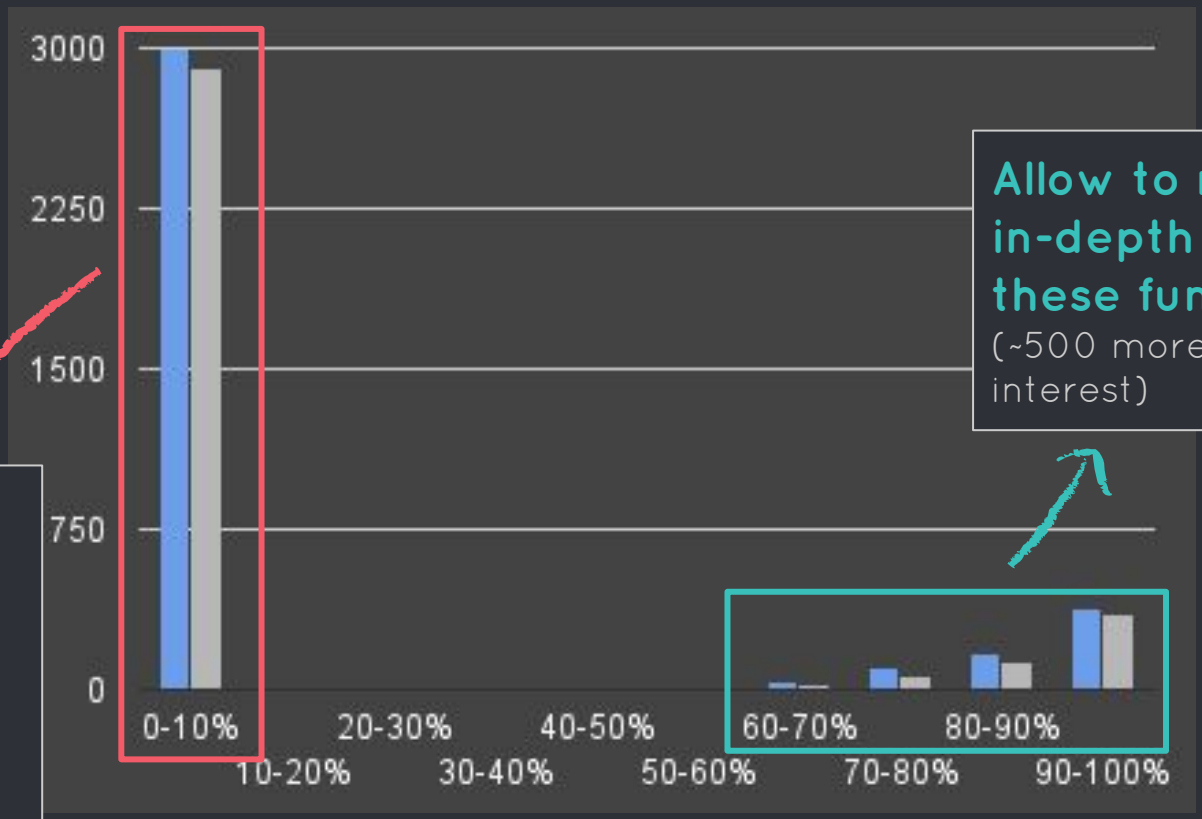
Goal: Compute the percentage of conditional jump obfuscated within a function



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)

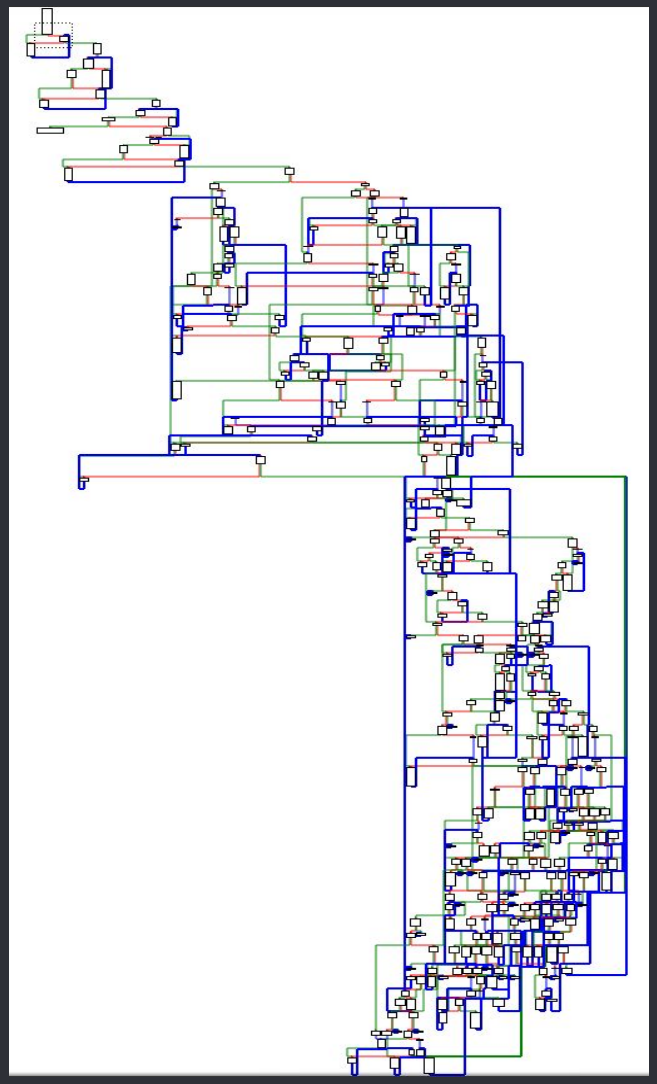
Analysis: Code coverage

Results of the liveness propagation and identification of spurious instructions

	C637 Sample #1	99B4 Sample #2
#Total instruction	505,008	434,143
#Alive	+279,483	+241,177
#Dead	-121,794	-113,764
#Spurious	-103,731	-79,202
#Delta with sample #0	**47,576**	**9,270**

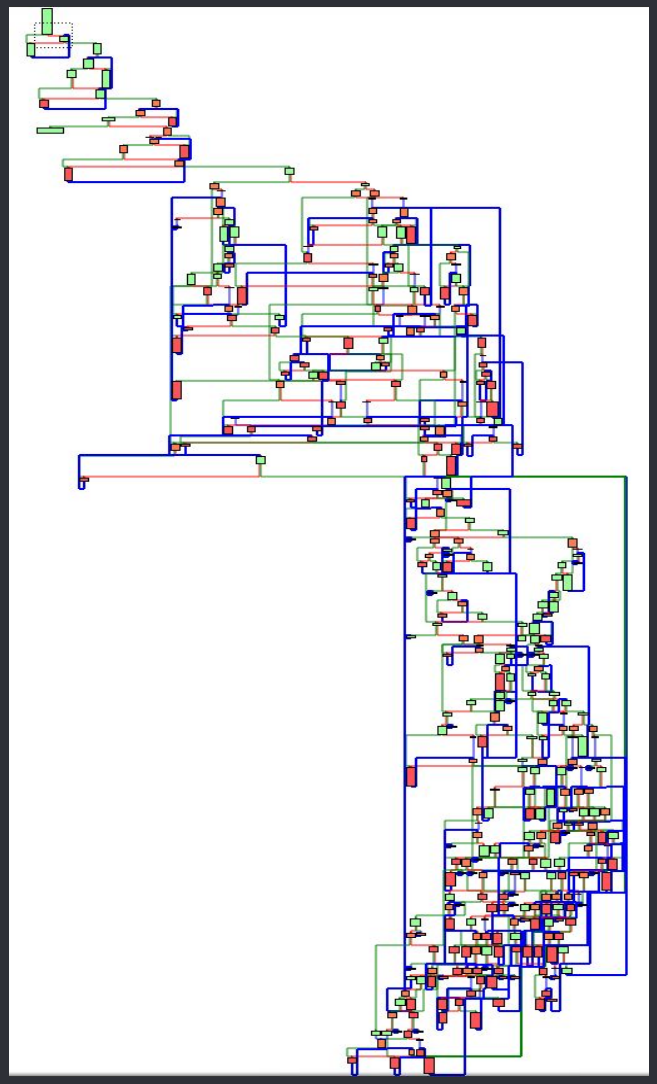
In both samples the difference with the un-obfuscated binary is very low, and probably due to some noise

Analysis: Reduced CFG extraction



Original CFG

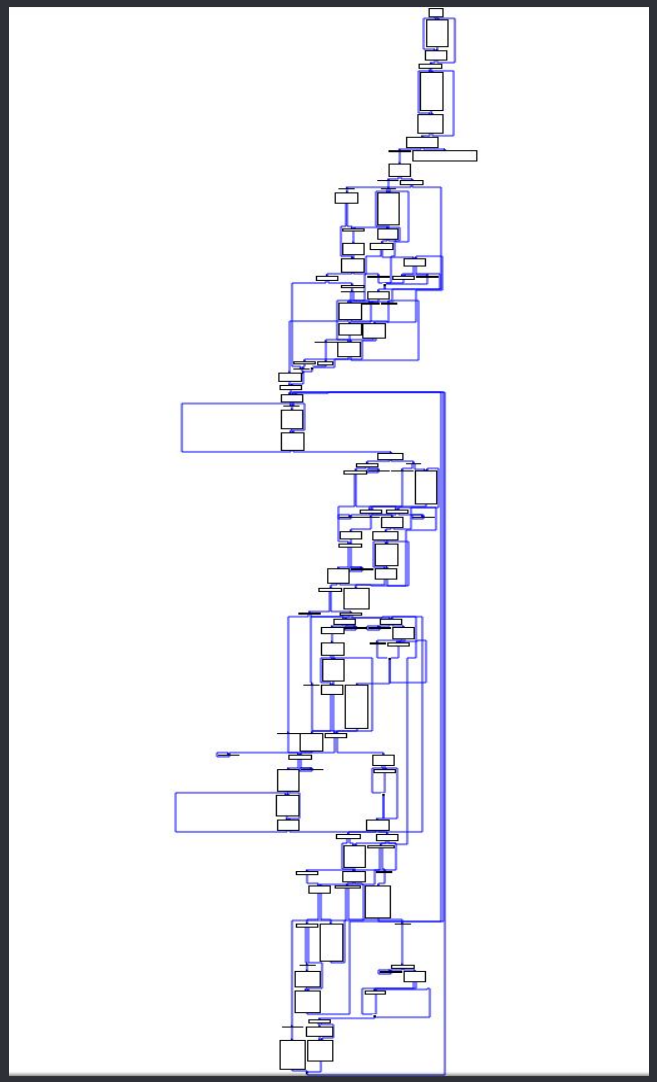
Analysis: Reduced CFG extraction



- Alive
- Spurious
- Dead

Tagged CFG

Analysis: Reduced CFG extraction



Extracted CFG

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

Robin David
robin.david@riseup.net
@RobinDavid1

Sébastien Bardin
sebastien.bardin@cea.fr

black hat EUROPE 2016