TARGETING INFEASIBILITY QUESTIONS
on
OBUFSCATED CODES

Sébastien Bardin (CEA LIST)

Robin David (CEA LIST) & Jean-Yves Marion (LORIA)

IN A NUTSHELL

- Challenge: malware ***deobfuscation***
- ***Infeasibility questions*** **are a blind spot of current automated techniques**
- **We propose an efficient, robust and precise method for them**
- **Very promising case-studies**

CONTEXT: MALWARE COMPREHENSION

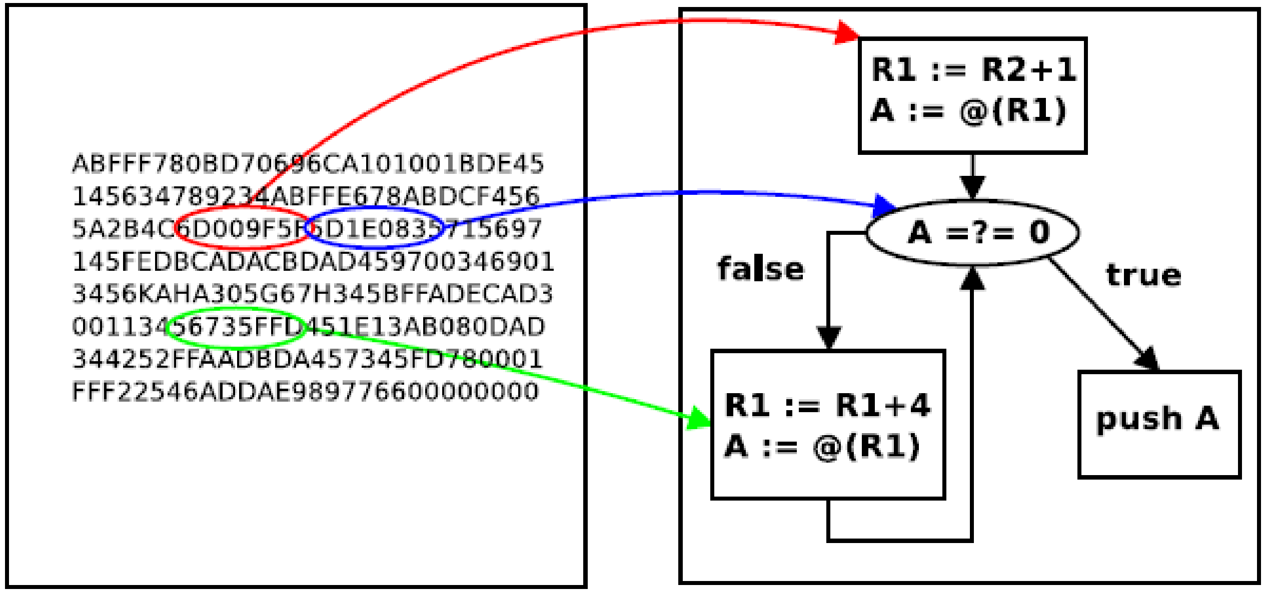
APT: highly sophisticated attacks
- **Targeted malware**
- **Written by experts**
- Attack: 0-days
- Defense: stealth, **obfuscation**
- **Sponsored by states or mafia**

The day after: **malware comprehension**
- understand what has been going on
- mitigate, fix and clean
- improve defense

USA elections: DNC Hack

Goal: help malware comprehension
- **Reverse of heavily obfuscated code**
- **Identify and simplify protections**

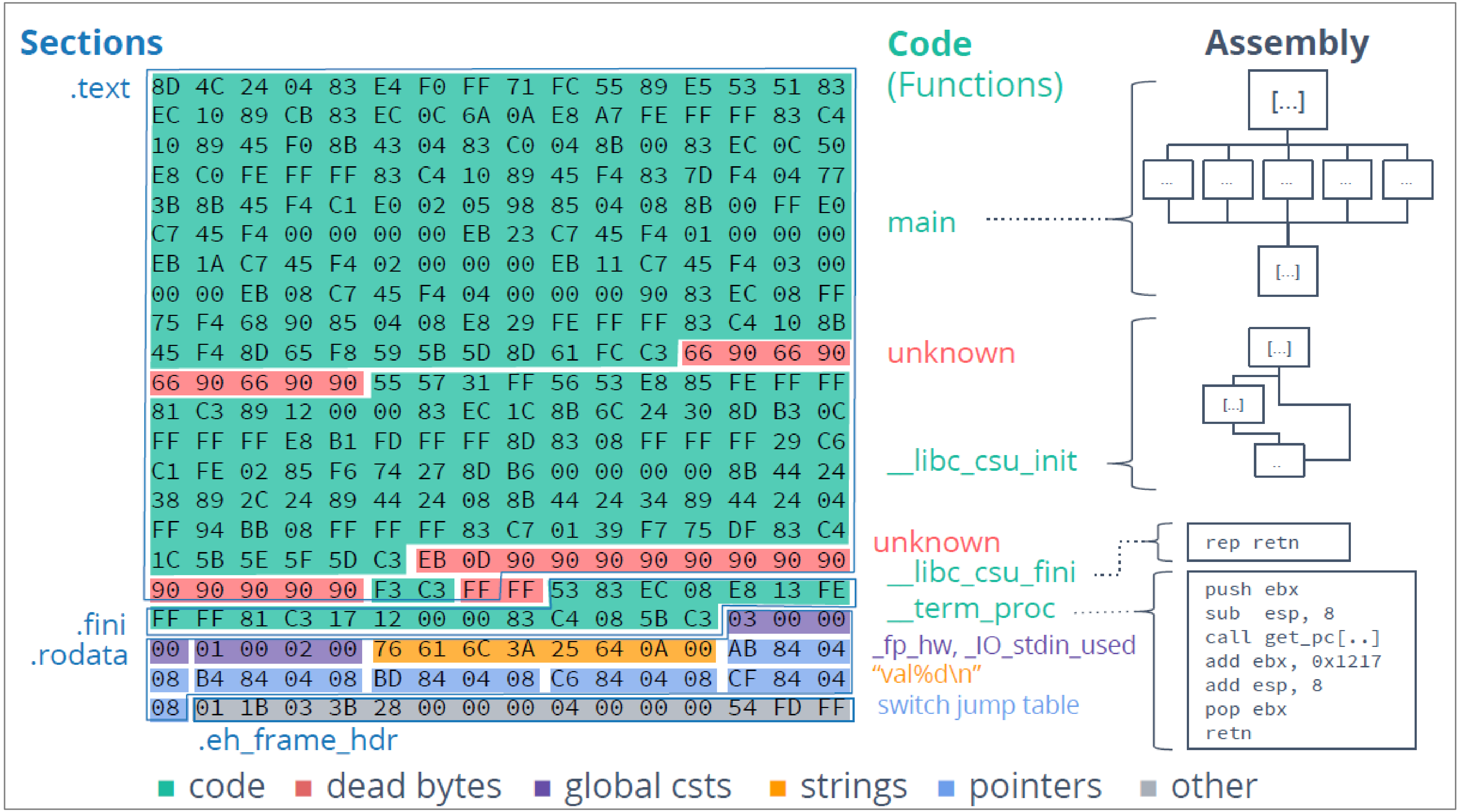
CHALLENGE: CORRECT DISASSEMBLY



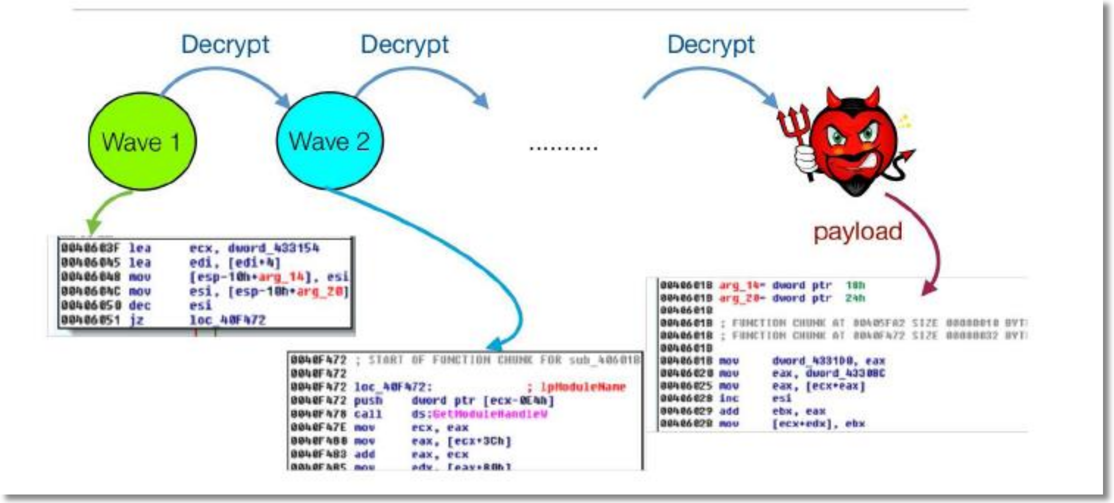
Basic reverse problem
- **aka model recovery**
- **aka CFG recovery**

CAN BE TRICKY!

- code – data
- dynamic jumps (jmp eax)



CAN BECOME A NIGHTMARE (OBFUSCATION)



Obfuscation: make a code hard to reverse

- self-modification
- encryption
- virtualization
- code overlapping
- opaque predicates
- callstack tampering
- …

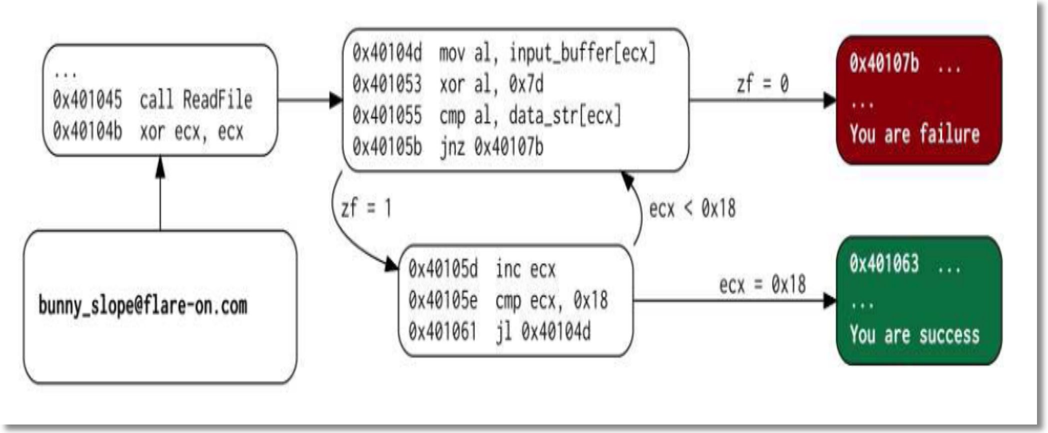
eg: $7y^2 - 1 \neq x^2$

(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>
```

address	instr
80483d1	call +5
80483d6	pop edx
80483d7	add edx, 8
80483da	push edx
80483db	ret
80483dc	.byte{invalid}
80483de	[...]



EXAMPLE: OPAQUE PREDICATE

Constant-value predicates

(always true, always false)

- dead branch points to spurious code
- goal = waste reverser time & efforts

eg: $\mathbf{7y^2 - 1 \neq x^2}$

(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>
```

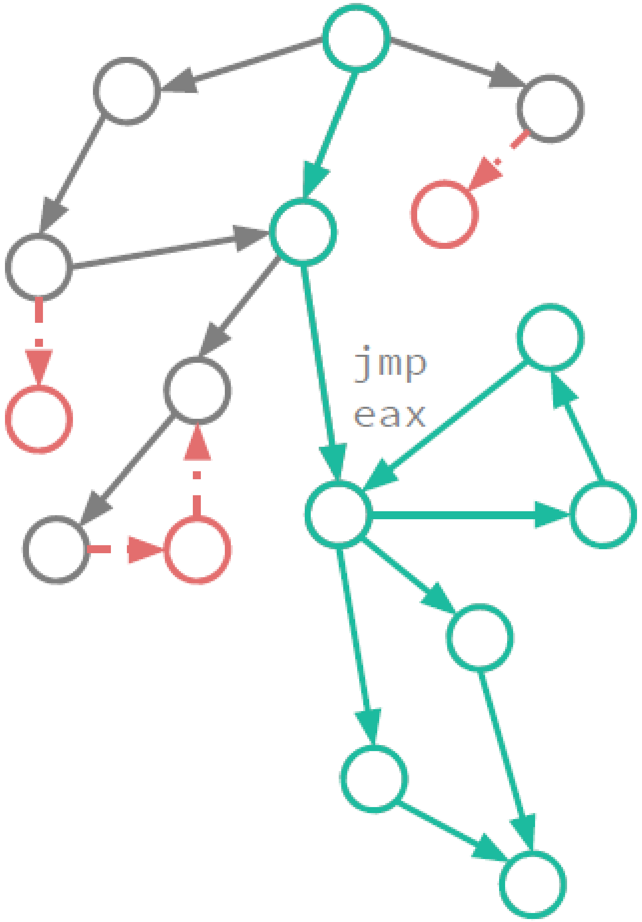
EXAMPLE: STACK TAMPERING

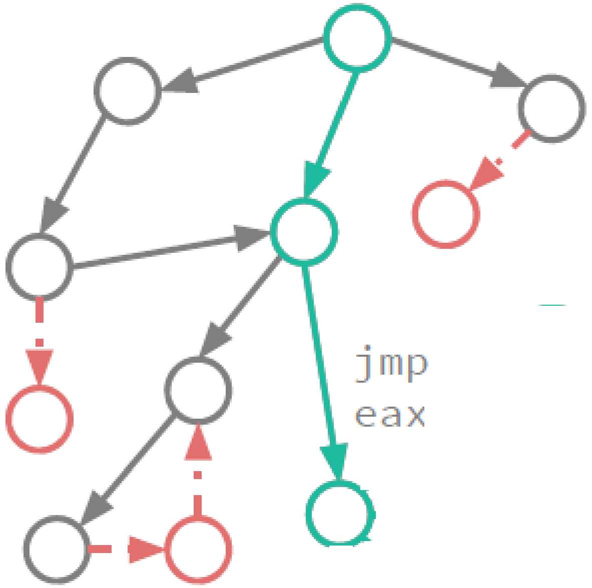
Alter the standard compilation scheme:
ret do not go back to call

- hide the real target
- return site may be spurious code

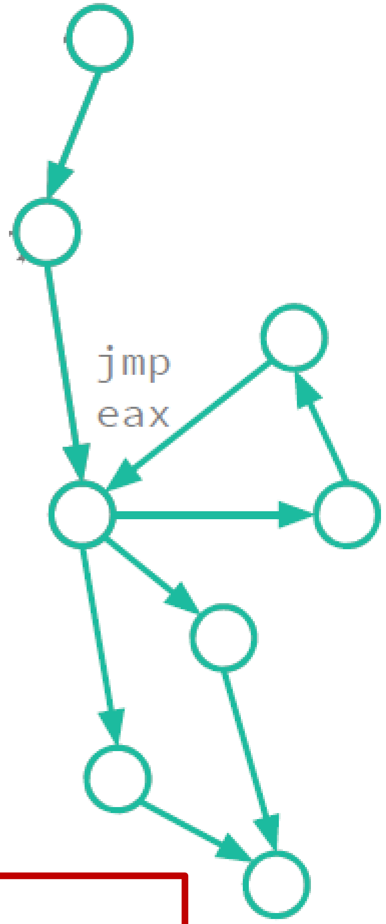
address	instr
80483d1	call +5
80483d6	pop edx
80483d7	add edx, 8
80483da	push edx
80483db	ret
80483dc	.byte{invalid}
80483de	[...]

STANDARD DISASSEMBLY TECHNIQUES ARE NOT ENOUGH





Static analysis
- **too fragile vs obfuscation**
- **junk instr, missed instr.**

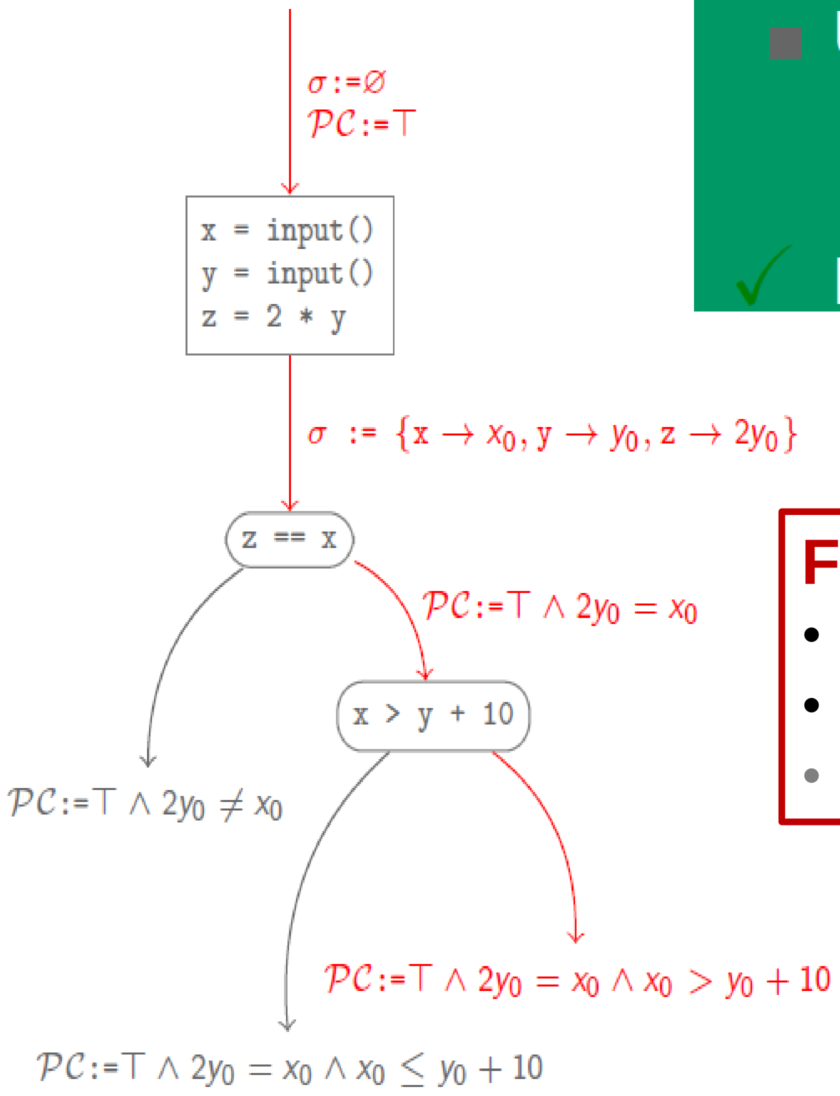


Dynamic analysis
- **robust vs obfuscation**
- **too incomplete**

DYNAMIC SYMBOLIC EXECUTION CAN HELP

```
int main () {
    int x = input();
    int y = input();
    int z = 2 * y;
    if (z == x) {
        if (x > y + 10)
            failure;
    }
    success;
}
```

- given a path of the program
- automatically find input that follows the path
- then, iterate over all paths



- Under-approximation
  - correct
  - relatively complete
- ✓ No false alarm

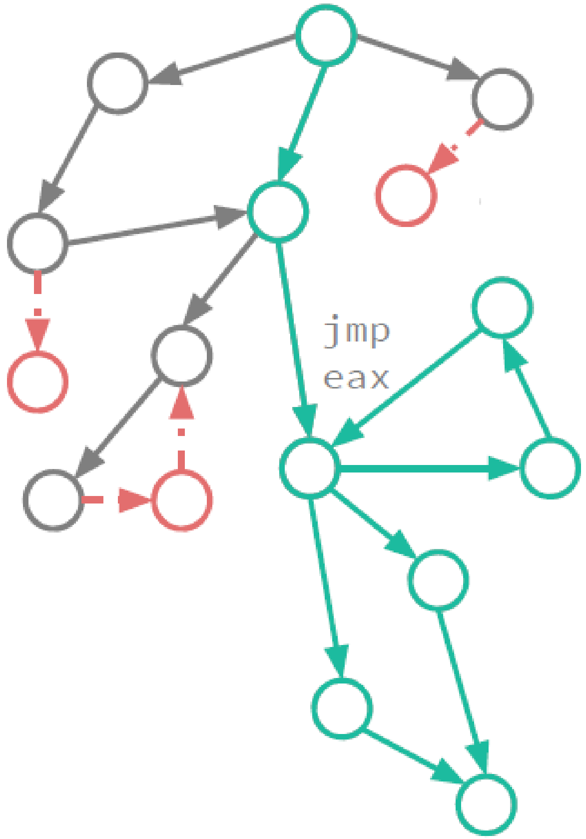
For deobfuscation
- **find new real paths**
- **robust**
- still incomplete

YET … WHAT ABOUT INFEASIBILITY QUESTIONS?

Prove that something is always true (resp. false)

Many such issues in reverse

- **is a branch dead?**
- **does the ret always return to the call?**
- **have i found all targets of a dynamic jump?**



And more

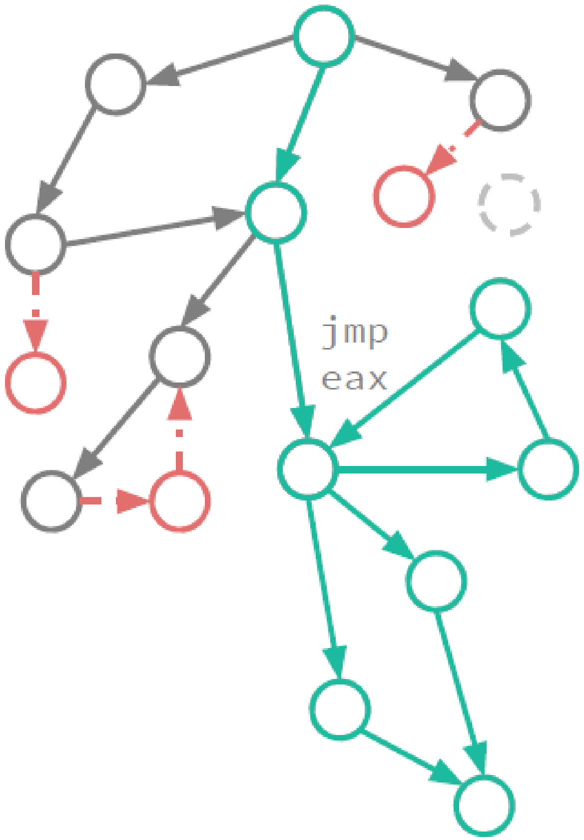
- **does this malicious ret always go there?**
- **does this expression always evaluate to 15?**
- **does this self-modification always write this opcode?**
- **does this self-modification always rewrite this instr.?**
- **…**

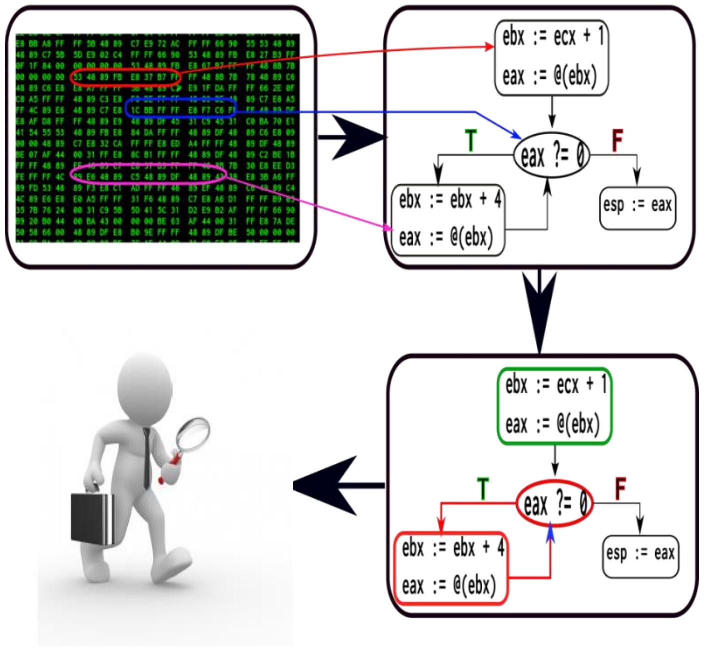
Not addressed by DSE

- **Cannot enumerate all paths**

OUR CHALLENGE

Check infeasibility questions in obfuscated codes
- **scale to realistic malware sizes**
- **robust to obfuscation such as self-modification**
- **precise**
- **generic**





Rest of the talk:
- **opaque predicate**
- **stack tampering**

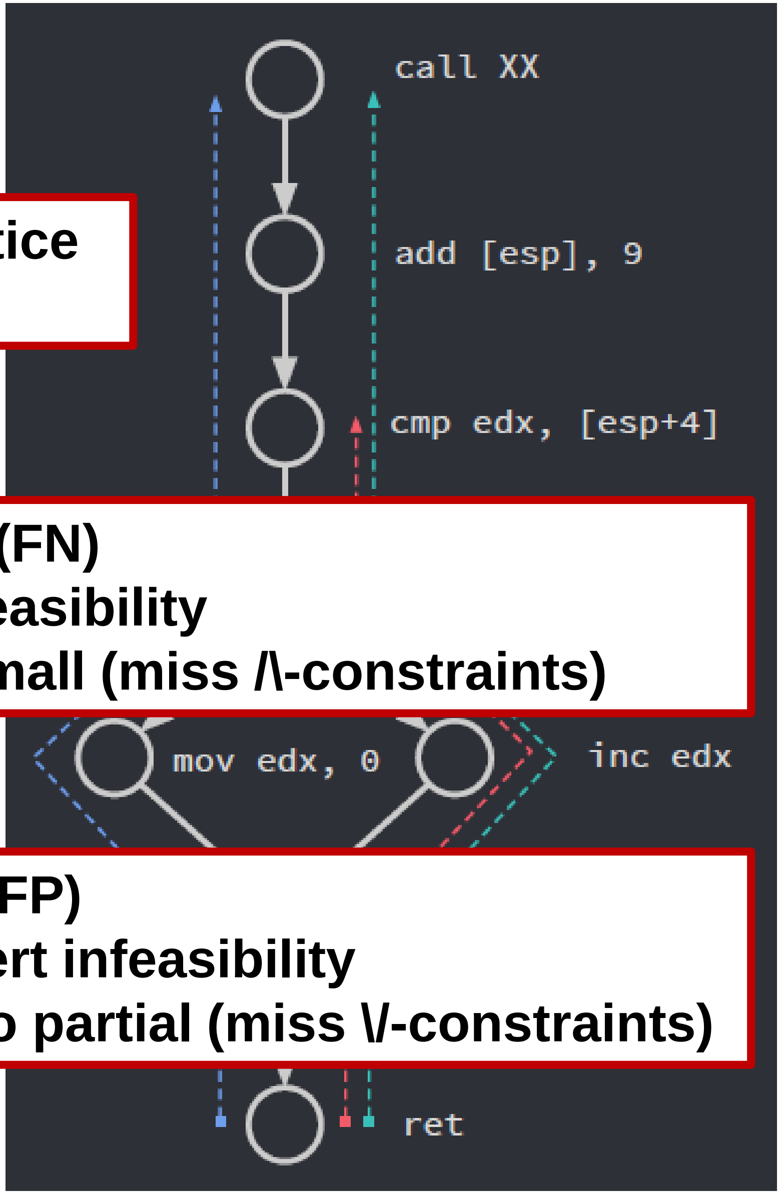
OUR PROPOSAL: BACKWARD-BOUNDED SYMBOLIC EXECUTION

Insight 1: symbolic reasoning
- **precision**
- **But: need finite #paths**

Insight 2: backward-bounded
- **pre_k(c)=0 => c is infeasible**
- **finite #paths**
- **efficient, depends on k**
- **But: backward on jump eax?**

Insight 3: dynamic partial CFG
- **solve (partially) dyn. jumps**
- **robustness**

Low FP/FN rates in practice
- **ground truth xp**



False negative (FN)
- **can miss infeasibility**
- **why: k too small (miss /\\-constraints)**

False positive (FP)
- **wrongly assert infeasibility**
- **why: CFG too partial (miss \\/-constraints)**

EXPERIMENTAL EVALUATION

- **Controlled experiments** (ground truth) → **precision**
- **Large-scale experiment: packers** → **scalability, robustness**
- **Case-study: X-tunnel malware** → **usefulness**

CONTROLLED EXPERIMENTS

- **Goal = assess the precision of the technique**
  - ground truth value
- **Experiment 1: opaque predicates (o-llvm)**
  - 100 core utils, 5x20 obfuscated codes
  - k=16: 3.46% error, no false negative
  - robust to k
  - efficient: 0.02s / query
- **Experiment 2: stack tampering (tigress)**
  - 5 obfuscated codes, 5 core utils
  - almost all genuine ret are proved (no false positive)
  - many malicious ret are proved « single-targets »

k	OP (5556) ok	OP (5556) miss (FN)	Genuine (5183) ok	Genuine (5183) miss (FP)	TO	Error rate (FP+FN)/Tot (%)	Time (s)	avg/query (s)
2	0	5556	5182	1	0	51.75	89	0.008
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	96	0.009
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	120	0.011
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**152**	**0.014**
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**197**	**0.018**
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**272**	**0.025**
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**384**	**0.036**
32	5552	4	4579	604	25	5.66	699	0.065
40	5548	8	4523	660	39	6.22	1145	0.107
50	5544	12	4458	725	79	6.86	2025	0.189

- **Very precise résults**
- **Seems efficient**

Sample	runtime genuine #ret †	runtime genuine proved genuine	runtime genuine proved a/d	runtime violation #ret †	runtime violation proved a/d	runtime violation proved single
obfuscated programs						
simple-if	6	6	6/0	9	0/0	8
bin-search	15	15	15/0	25	0/0	24
bubble-sort	6	6	6/0	15	0/1	13
mat-mult	31	31	31/0	69	0/0	68
huffman	19	19	19/0	23	0/3	19
non-obfuscated programs						
ls	30	30	30/0	0	-	-
dir	35	35	35/0	0	-	-
mktemp	21	20	20/0	0	-	-
od	21	21	21/0	0	-	-
vdir	49	43	43/0	0	-	-

CASE-STUDY: PACKERS

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	[illegible]	[illegible]	[illegible]	159	0	48
ASPack v2.12	377K	1	[illegible]	[illegible]	[illegible]	24	11	6
Crypter v1.12	1.1M	1	1	1	[illegible]	24	125	78
Expressor	635K	1	1	[illegible]	[illegible]	[illegible]	14	0
FSG v2.0	68k	1	1	[illegible]	[illegible]	[illegible]	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	[illegible]	[illegible]	[illegible]	14	0
TELock v0.51	406K	1	1	[illegible]	[illegible]	[illegible]	3	1
Upack v0.39	711K	1	1	[illegible]	[illegible]	[illegible]	7	1

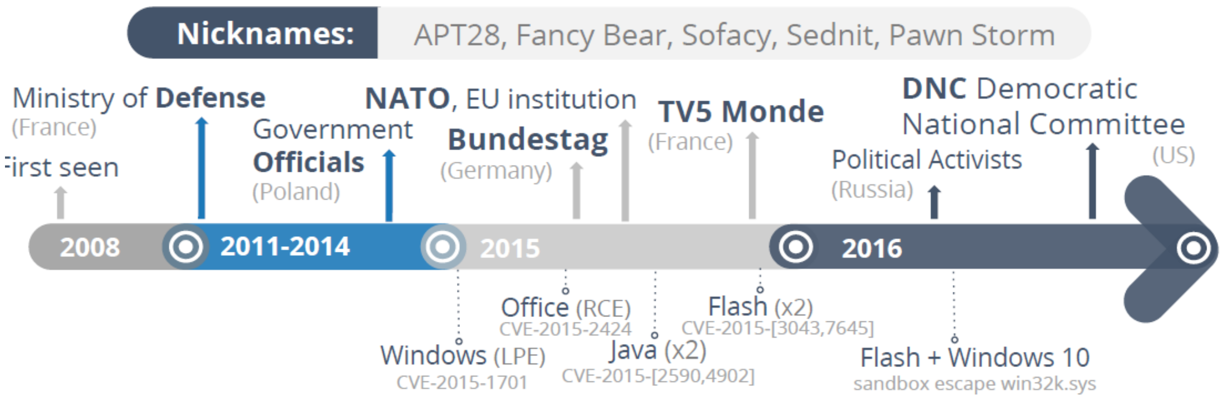
The technique scale on significant traces

Many true positives. Some packers are using it intensively

Packers using ret to perform the final tail transition to the entrypoint

Packers: legitimate software protection tools
(basic malware: the sole protection)

CASE-STUDY: THE XTUNNEL MALWARE (part of DNC hack)



Two heavily obfuscated samples

- **Many opaque predicates**

Goal: detect & remove protections

- Identify 50% of code as spurious
- Fully automatic, < 3h

	C637 Sample #1	99B4 Sample #2
#total instruction	**505,008**	**434,143**
#alive	+279,483	+241,177

SECURITY ANALYSIS: COUNTER-MEASURES (and mitigations)

- **Long dependecy chains (evading the bound k)**
  - Not always requires the whole chain to conclude!
  - Can use a more flexible notion of bound (data-dependencies, formula size)

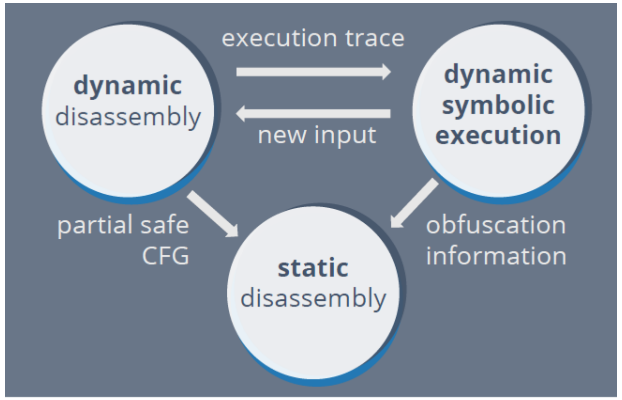
- **Hard-to-solve predicates (causing timeouts)**
  - A time-out is already a valuable information
  - Opportunity to find infeasible patterns (then matching), or signatures
  - Tradeoff between performance penalty vs protection focus
  - Note: must be input-dependent, otherwise removed by standard DSE optimizations

- **Anti-dynamic tricks (fool initial dynamic recovery)**
  - Can use the appropriate mitigations
  - Note: some tricks can be circumvent by symbolic reasoning

Current state-of-the-art
- **push the cat-and-mouse game further**
- **raise the bar for malware designers**

CONCLUSION & TAKE AWAY

- **What has been done**
  - Identify infeasibility questions as a blind spot of deobfuscation techniques
  - Propose an efficient, robust and precise method
  - Controlled experiments and large-scale studies



- **Semantic analysis can change the game of deobfuscation**
  - Complement existing approaches
  - Open the way to fruitful combinations [see the paper]

- **Formal methods can be useful for malware, but must be adapted**
  - Need robustness and scalability!
  - Accept to lose both correctness & completeness – in a controlled way

- **BINSEC platform: looking for collaborations and users ☺**
  - Open-source, still in its infancy

Commissariat à l'énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019