

# FROM SAFETY TO SECURITY: The Case of Binary-Level Code Analysis

Sébastien Bardin  
(CEA LIST, France)

With Richard Bonichon, Matthieu Lemerre,  
Robin David, Josselin Feist, Adel Djoudi, Benjamin Farinier, etc.

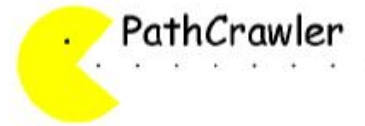
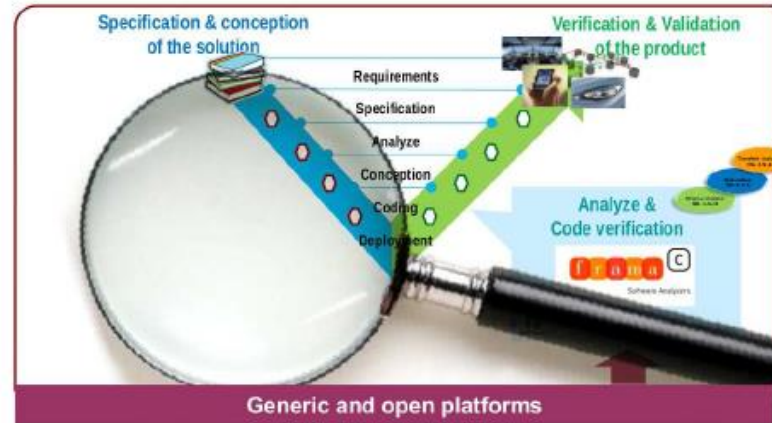


# ABOUT MY LAB @CEA

- Located in Paris, France

## CEA LIST, Software Safety & Security Lab

- rigorous tools for building high-level quality software
- second part of V-cycle
- automatic software analysis
- mostly source code

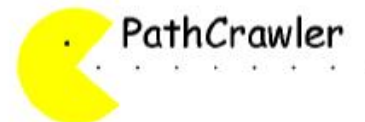
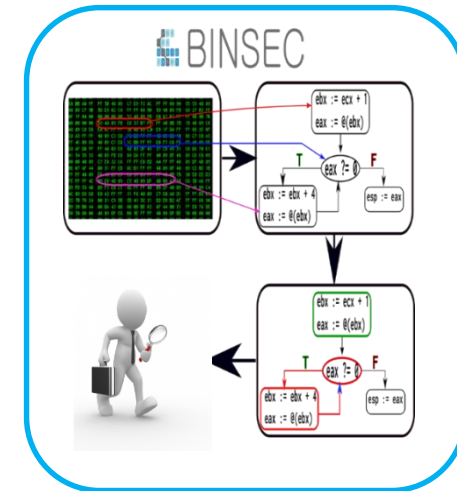
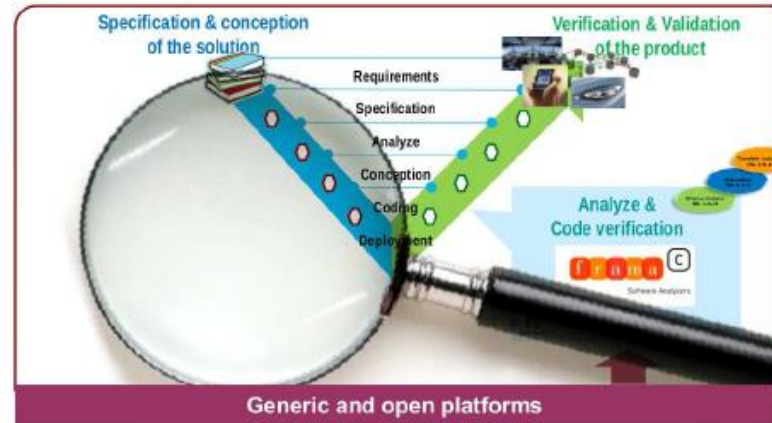


# ABOUT MY LAB @CEA

• Located in Paris, France

## CEA LIST, Software Safety & Security Lab

- rigorous tools for building high-level quality software
- second part of V-cycle
- automatic software analysis
- mostly source code



# IN A NUTSHELL

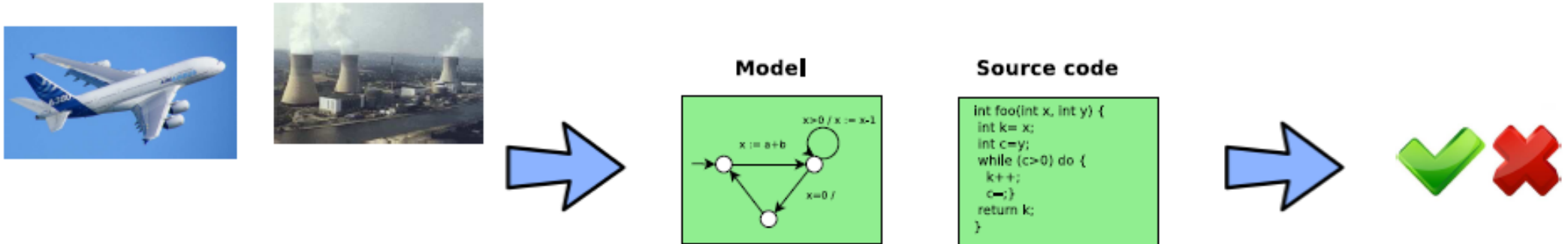
- **Binary-level security analysis: many applications, many challenges**
- **Standard techniques not enough**
- **Formal methods can help ... but must be strongly adapted**
  - [Complement existing methods]
  - Need robustness, precision and scalability!
  - Acceptable to lose both correctness & completeness – in a controlled way
  - **New challenges and variations, many things to do!**
- ***This talk: our experience on adapting source-level safety analysis for binary-level security***



# ABOUT FORMAL METHODS

- Between Software Engineering and Theoretical Computer Science
- Goal = proves correctness in a mathematical way

Success in safety-critical



Key concepts :  $M \models \varphi$

- $M$  : semantic of the program
- $\varphi$  : property to be checked
- $\models$  : algorithmic check

Ex : Airbus

Verification of

- runtime errors [Astrée]
- functional correctness [Frama-C \*]
- numerical precision [Fluctuat \*]
- source-binary conformance [CompCert]
- ressource usage [Absint]

\* : by CEA DILS/LSL

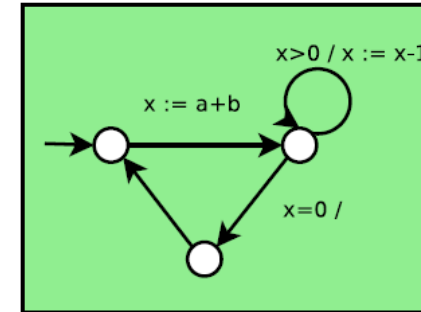




# NOW: MOVING TO BINARY-LEVEL SECURITY ANALYSIS



## Model



## Source code

```

int foo(int x, int y) {
  int k= x;
  int c=y;
  while (c>0) do {
    k++;
    c--;}
  return k;
}
  
```

## Assembly

```

_start:
  load A 100
  add B A
  cmp B 0
  jle label

label:
  move @100 B
  
```

## Executable

```

ABFFF780BD70696CA101001BDE45
145634789234ABFFE678ABDCF456
5A2B4C6D009F5F5D1E0835715697
145FEDBCADACBDAD459700346901
3456KAHA305G67H345BFFADECAD3
00113456735FFD451E13AB080DAD
344252FFAADBDA457345FD780001
FFF22546ADDAE989776600000000
  
```

- The success of formal methods for safety
- **Why binary-level security analysis?**
- The hard journey from source-level safety to binary-level security
- Our approach
- Conclusion



# WHY BINARY-LEVEL SECURITY ANALYSIS?

## Malware comprehension

## Protection-evaluation

## Vulnerability analysis



- Obsidium
- JD Pack
- WinUpack
- Expressor
- Armadillo
- EP Protector
- ACProtect
- TELockSVK
- Yoda's Crypter
- Neolite
- UPX MoleBox
- FSG Upack
- ASPack
- Petite
- nPack
- PE Spin
- Enigma
- RLPack
- Mystic VMProtect



```

4800 0000 5dc3 5589 e5c7 0812 0009 00b8 4800 0000 5dc3 5589
0000 00b8 4500 0900 0820 0000 00b8 4500 0000
bf0e 0821 0000 090b 0540 bf0e 0821 0000 00b8
e5c7 0540 bf0e 0822 0540 090b 090b 5dc3 5589 bf0e 082
5dc3 5589 0583 ec10 090b 090b 090b 5dc3 5589 0583 ec1
0000 a148 bf0e 0883 f809 48bf 0e09 0100 0000 a148 bf0e 088
8b04 8548 e10b 08ff e0c5 e927 0002 0000 8b04 8548 e10b 08f
09c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 09c6 45f9 00c6 45f
0900 00e9 d901 0000 c645 0548 bf0e 0882 0000 00e9 d901 090
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0
48bf 0e08 0300 0900 807d fb00 750a c705 48bf 0e08 0300 090
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 7410 807d fc00
fc00 7415 807d fb00 740f 0900 0000 467d fc00 7415 807d fb0
0500 0000 e988 0100 00e9 c705 48bf 0e08 0500 0000 e988 010
f701 c645 f800 c645 f900 0000 0000 c645 f701 c645 f800 c64
fc00 740f c705 48bf 0e08 c645 fa02 807d fc00 740f c705 48b
0100 00e9 5901 0900 c645 0900 0009 095e 0100 00e9 5901 090
c645 f900 c645 fa03 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b
fc00 750a c705 48bf 0e08 0500 0000 807d fc00 750a c705 48b
fc00 740f c705 48bf 0e08 0300 0000 807d fe00 740f c705 48b
0100 740f 901 0900 c645 0500 0000 e96e 0100 00e9 0901 000
c645 f900 e0c5 091 807d f701 c645 f800 c645 f901 c645 fa0
48bf 460 0900 09c4 00 750f c705 48bf 0e08 0400 0000
0000 c645 f701 c645 f800 0900 00e9 df00 0000 c645 f701 c64
fa04 807d fc00 7410 807d c645 9009 c645 fa04 807d fc00 741
48bf 0e08 0700 0900 807d ff00 750a c705 48bf 0e08 0700 090
ff00 740f c705 48bf 0e08 fc00 740f c705 48b
c645 f900 c645 fa05 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 f701 c645 f800 c645 f900 c645 fa0
fc00 750a c705 48bf 0e08 0800 0000 807d fc00 750a c705 48b
fe00 7506 807d ff00 740c 0900 0000 807d fe00 7506 807d ff0
0500 0000 eb4b eb49 c645 c705 48bf 0e08 0500 0000 eb4b eb4
c645 f901 c645 fa02 807d f701 c645 f800 c645 f901 c645 fa0
5dc3 5589 e5c7 0540 bf0e 0822 0540 090b 5dc3 5589 e5c7 054
09 5dc3 5589
0000 00b8 4500 0900
e583 ec11
bf0e 0883
e10b 08f8
09c6 45f7
d901 0000
09 c645 fa0
08 0300 0900
0a c705 48b
05 807d fb00
09 e988 0100
    
```



Find a needle in the heap!



# BUT ... THIS IS HARD!!!



# DISASSEMBLY IS ALREADY TRICKY!

- code – data ??
- dynamic jumps (jmp eax)

### Sections

.text															
8D	4C	24	04	83	E4	F0	FF	71	FC	55	89	E5	53	51	83
EC	10	89	CB	83	EC	0C	6A	0A	E8	A7	FE	FF	FF	83	C4
10	89	45	F0	8B	43	04	83	C0	04	8B	00	83	EC	0C	50
E8	C0	FE	FF	FF	83	C4	10	89	45	F4	83	7D	F4	04	77
3B	8B	45	F4	C1	E0	02	05	98	85	04	08	8B	00	FF	E0
C7	45	F4	00	00	00	00	EB	23	C7	45	F4	01	00	00	00
EB	1A	C7	45	F4	02	00	00	00	EB	11	C7	45	F4	03	00
00	00	EB	08	C7	45	F4	04	00	00	00	90	83	EC	08	FF
75	F4	68	90	85	04	08	E8	29	FE	FF	FF	83	C4	10	8B
45	F4	8D	65	F8	59	5B	5D	8D	61	FC	C3	66	90	66	90
66	90	66	90	90	55	57	31	FF	56	53	E8	85	FE	FF	FF
81	C3	89	12	00	00	83	EC	1C	8B	6C	24	30	8D	B3	0C
FF	FF	FF	E8	B1	FD	FF	FF	8D	83	08	FF	FF	FF	29	C6
C1	FE	02	85	F6	74	27	8D	B6	00	00	00	00	8B	44	24
38	89	2C	24	89	44	24	08	8B	44	24	34	89	44	24	04
FF	94	BB	08	FF	FF	FF	83	C7	01	39	F7	75	DF	83	C4
1C	5B	5E	5F	5D	C3	EB	0D	90	90	90	90	90	90	90	90
90	90	90	90	90	F3	C3	FF	FF	53	83	EC	08	E8	13	FE
.fini															
FF	FF	81	C3	17	12	00	00	83	C4	08	5B	C3	03	00	00
.rodata															
00	01	00	02	00	76	61	6C	3A	25	64	0A	00	AB	84	04
08	B4	84	04	08	BD	84	04	08	C6	84	04	08	CF	84	04
08	01	1B	03	3B	28	00	00	00	04	00	00	00	54	FD	FF
.eh_frame_hdr															

### Code (Functions)

main

unknown

\_\_libc\_csu\_init

unknown

\_\_libc\_csu\_fini

\_\_term\_proc

\_\_fp\_hw, \_IO\_stdin\_used

switch jump table

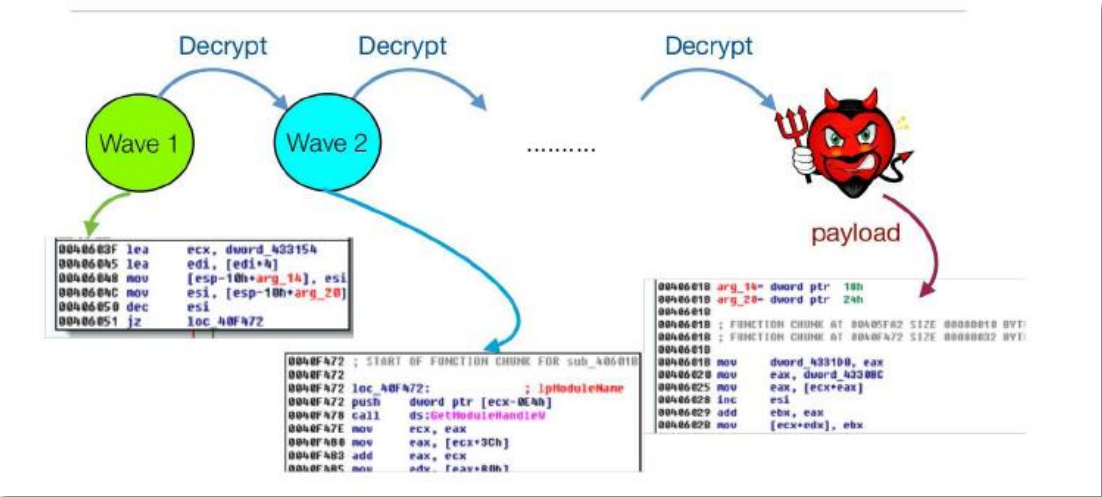
```

rep retn
push ebx
sub esp, 8
call get_pc[...]
add ebx, 0x1217
add esp, 8
pop ebx
retn
    
```

■ code  
 ■ dead bytes  
 ■ global csts  
 ■ strings  
 ■ pointers  
 ■ other



# AND IT CAN GET WORST! (adversarial setting)



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

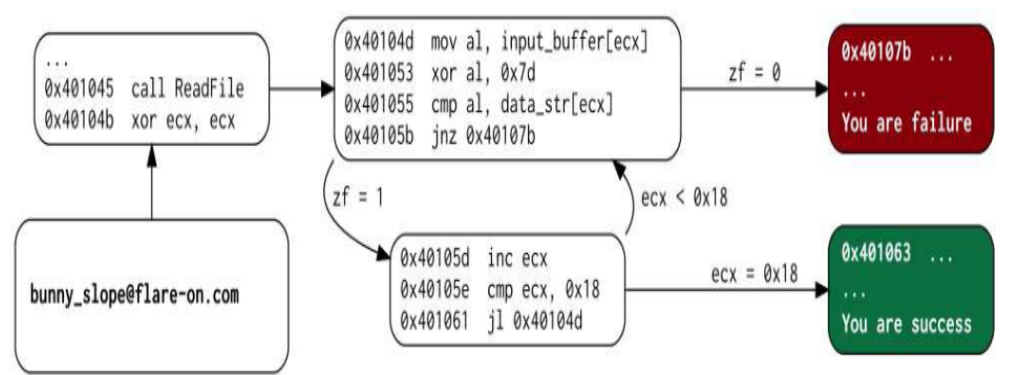


- self-modification
- encryption
- virtualization
- code overlapping
- opaque predicates
- callstack tampering
- ...

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>
    
```



# An old proverb comes true

Key concepts :  $M \models \varphi$

- $M$  : semantic of the program
- $\varphi$  : property to be checked
- $\models$  : algorithmic check



*We have a beautiful hammer*

*And it seems we found a nail 😊*

- The success of formal methods for safety
- Why binary-level security analysis?
- **The hard journey from source-level safety to binary-level security**
- Our approach
- Conclusion

# BINARY-LEVEL ANALYSIS



Break an implicit assumption in code analysis

- Low-level control (CFG?)
- Low-level data & memory

At the edge of current methods

Model	Source code
	<pre>int foo(int x, int y) {   int k= x;   int c=y;   while (c&gt;0) do {     k++;     c--;}   return k; }</pre>
Assembly	Executable
<pre>_start: load A 100 add B A cmp B 0 jle label  label: move @100 B</pre>	<pre>ABFFF780BD70696CA101001BDE45 145634789234ABFFE678ABDCF456 5A2B4C6D009F5F5D1E0835715697 145FEDBCADACBDAD459700346901 3456KAHA305G67H345BFFADECAD3 00113456735FFD451E13AB080DAD 344252FFAADBDA457345FD780001 FFF22546ADDAE989776600000000</pre>

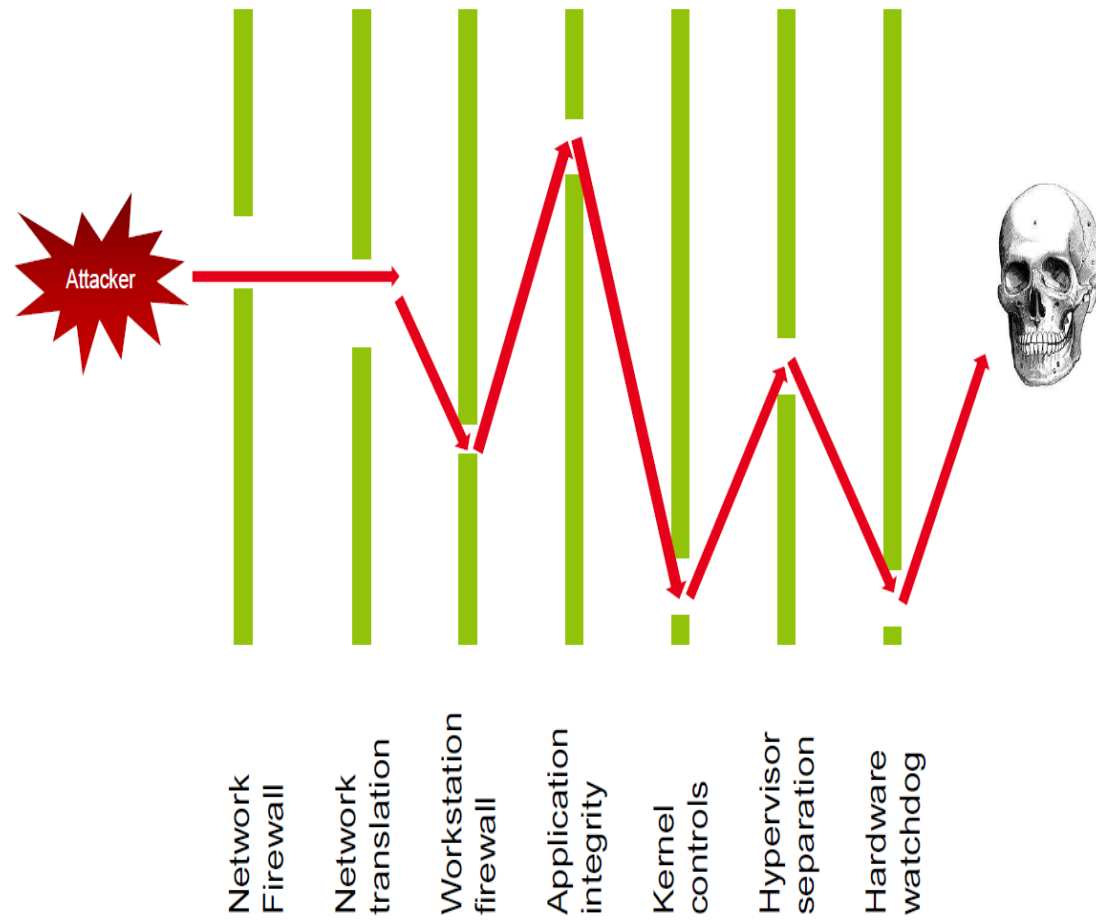
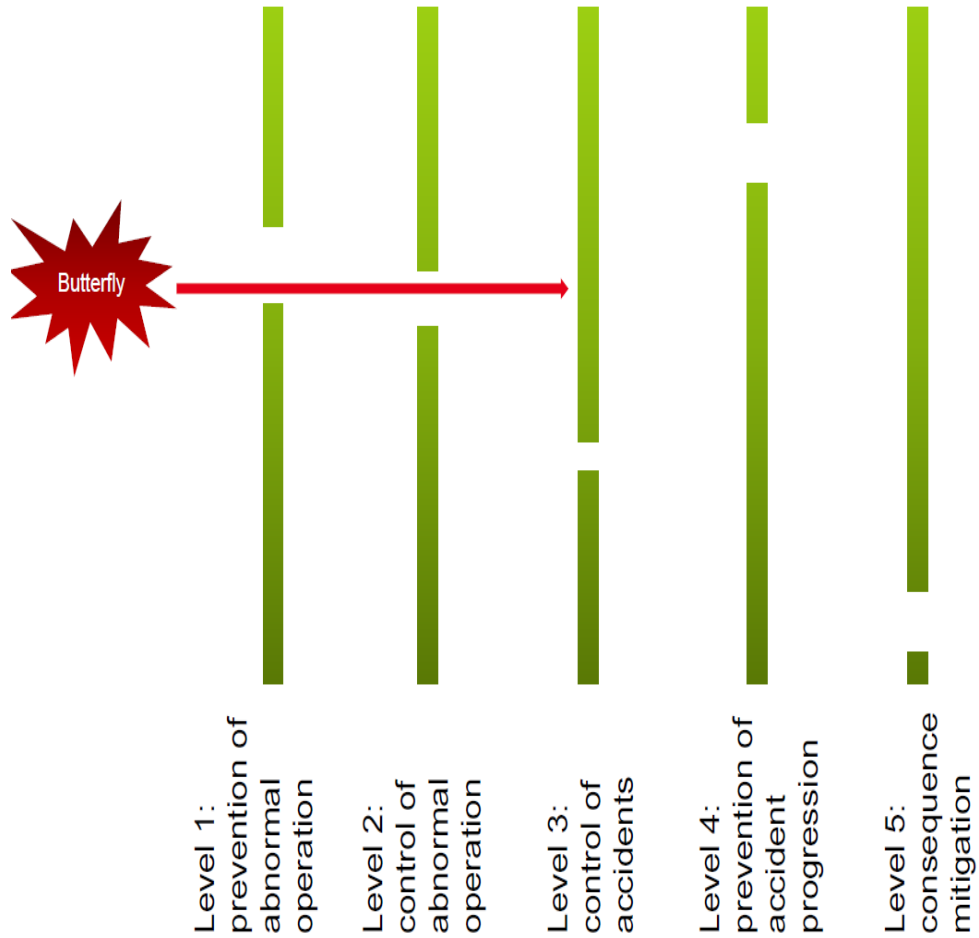


# ATTACKER <not treated today>



Nature is not nice

Attacker is evil



# NOT STRONGLY REGULATED ECOSYSTEM



- No coding guideline
- No annotation
- Require full automation
- Low tolerance to false positive

But absolute correctness is not required

- « correct enough »

# SECURITY IS NOT SAFETY

## • Source-level SAFETY

- Model: High-level language
- Properties: safety
- Algorithm: full correctness, possible help from user

- Strong incentive to human assistance
- Spec., parameter tuning, etc.

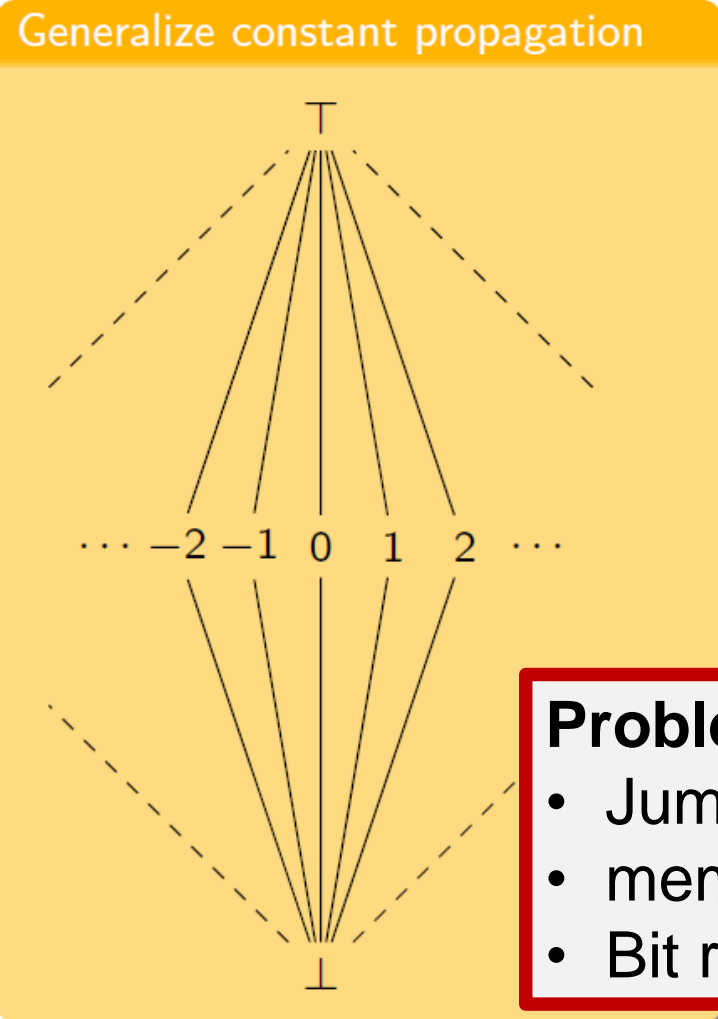
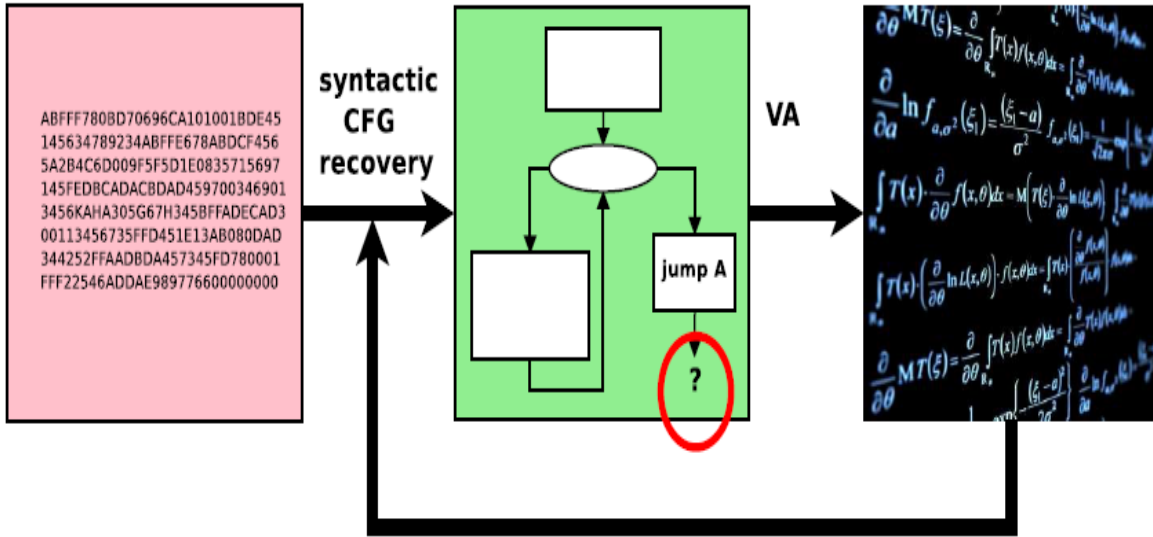
## • Binary-level SECURITY

- Model: **binary-level code, possibly adversarial, + attacker**
- Properties: safety, **k-safety, « bugs vs vulnerabilities »**
- Algorithm: **robust & precise enough**, fully automated

- no human assistance
- low tolerance to false positive



# <apparté> STATIC SEMANTIC ANALYSIS IS VERY VERY HARD ON BINARY CODE



Framework : abstract interpretation

- notion of abstract domain  
 $\perp, \top, \sqcup, \sqcap, \sqsubseteq, \text{eval}^\#$
- more or less precise domains  
. intervals, polyhedra, etc.
- fixpoint until stabilization

**Problems**

- Jump eax
- memory
- Bit reasoning

## Robustness

- able to survive dynamic jumps, self-modification, unpacking, etc
- *outside the scope of standard methods*

## Precision

- Machine arithmetic (overflow) and bit-level operations
- Byte-level memory, possible overlaps
- *hard for state-of-art formal methods*

## Reasonable scale

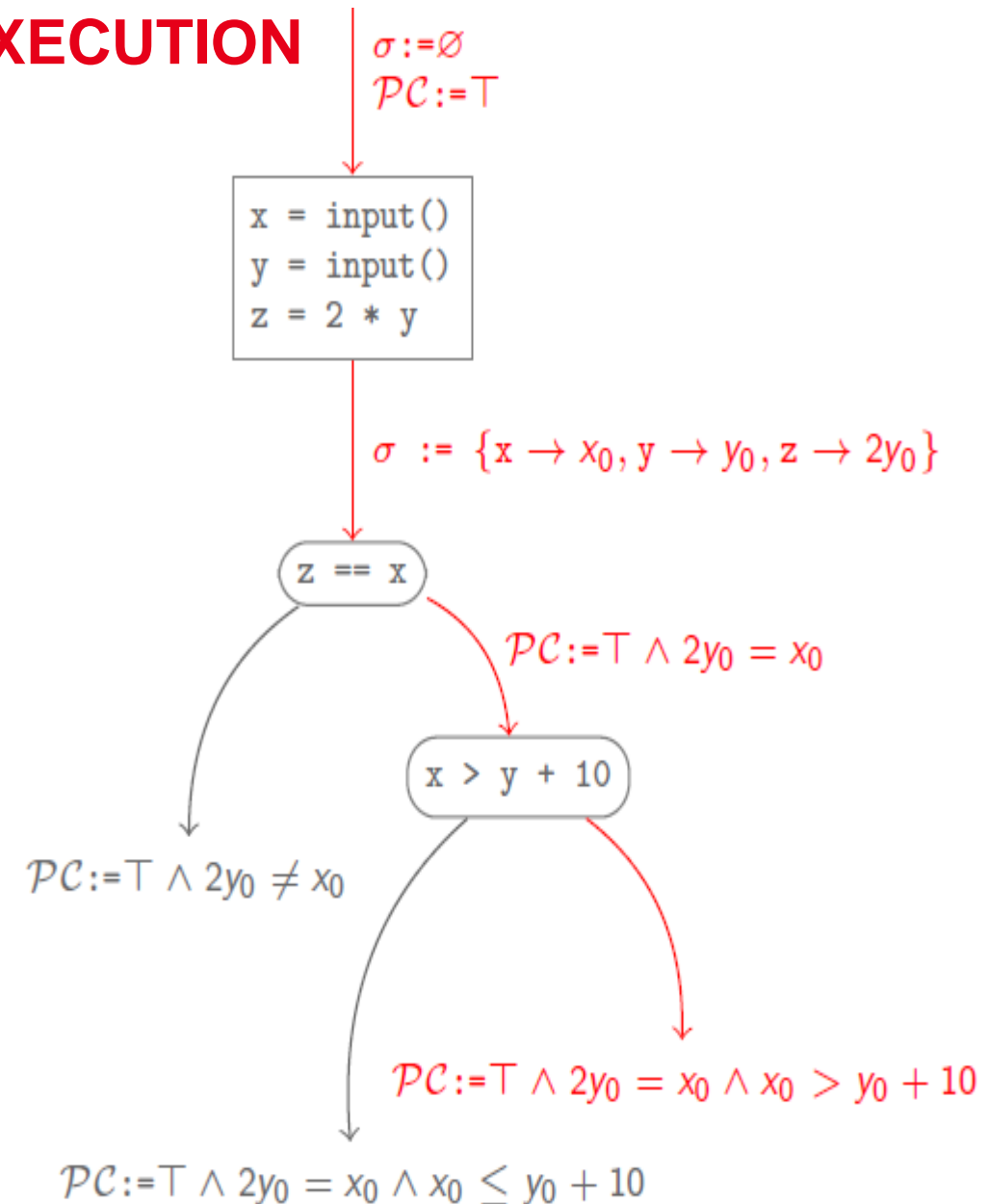
- The success of formal methods for safety
- Why binary-level security analysis?
- The hard journey from source-level safety to binary-level security
- **Our approach**
- Conclusion

# THE GOOD CANDIDATE: SYMBOLIC EXECUTION (Godefroid, 2005)

```
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 a program

- Compute its « **path predicate** »  $f$
- Solution of  $f \Leftrightarrow$  input following the path
- Solve it with powerful existing solvers





# THE GOOD CANDIDATE: SYMBOLIC EXECUTION

(Godefroid, 2005)

```
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 a program

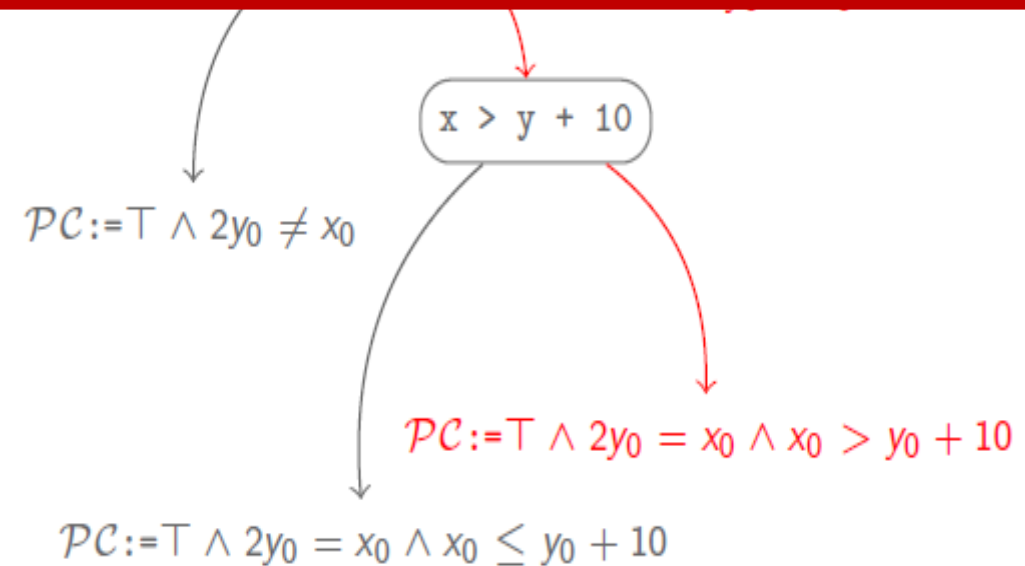
- Compute its « **path predicate** »  $f$
- Solution of  $f \Leftrightarrow$  input following the path
- Solve it with powerful existing solvers

$\sigma := \emptyset$   
 $PC := T$

$x = \text{input}()$   
 $y = \text{input}()$

**Good points:**

- **Precise (no false positive)**
- **Robust (symb. + dynamic)**
- **Extend rather well to binary code**



# THE GOOD CANDIDATE: SYMBOLIC EXECUTION (Godefroid, 2005)

```
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 a program

- Compute its « **path predicate** »  $f$
- Solution of  $f \Leftrightarrow$  input following the path
- Solve it with powerful existing solvers

Good points:

- Precise (no false positive)
- **Robust** (symb. + dynamic)
- Extend rather well to binary code

« **concretization** »

- Replace symbolic values by runtime values
- **Keep going when symbolic reasoning fails**
- Tune the tradeoff genericity - cost

$PC := T \wedge 2y_0 = x_0 \wedge x_0 \leq y_0 + 10$

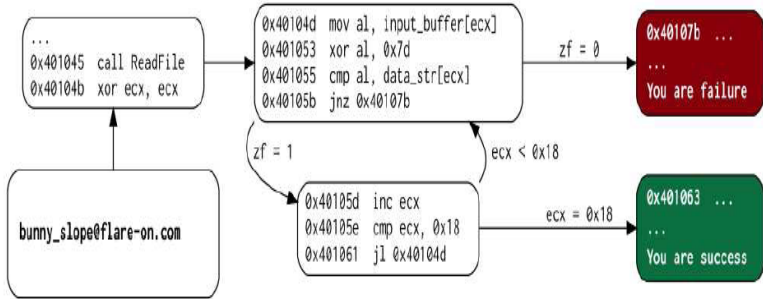
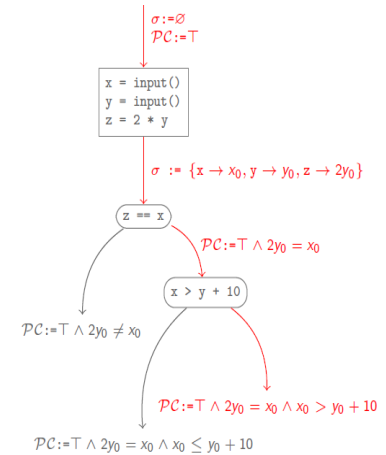
# ALLOWS TO EXPLORE A PROGRAM

- ## Forward reasoning
- Follows path
  - Find new branch / jumps
  - Standard DSE setting

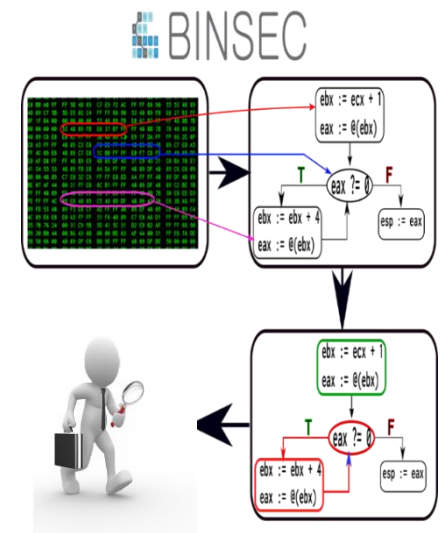
- ## Advantages
- Find new real paths
  - Even rare paths
- « dynamic analysis on steroids »

### GRUB2 CVE 2015-8370

Elevation of privilege  
 Information disclosure  
 Denial of service



- DSE is a good starting point for robustness & precision
- Can be adapted beyond the basic reachability case
  - variants
  - combination with other techniques
- **Loss of guarantees**
  - Accept ... But control!
  - Look for « correct enough » solutions
- **Finely tune the technology**
  - Tools for safety are not fully adequate



# CASE 1: COMPLEX VULNERABILITY DETECTION

## [SSPREW'16](with Josselin Feist et al.)

- Use-after-free bugs
- Very hard to find
  - Sequence of events
  - DSE lost



Find a needle in the heap!

```

4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558
0000 00b8 4500 0000 5dc3 0540 bf0e 0821 0000 00b8 4500 00
bf0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 0821 0000 00b
e5c7 0540 bf0e 0822 0000 0000 5dc3 5589 e5c7 0540 bf0e 082
5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 5589 e583 ec1
0000 a148 bf0e 0883 f809 48bf 0e08 0100 0000 a148 bf0e 088
8b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 e10b 08f
00c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f
0000 00e9 d901 0000 c645 0548 bf0e 0882 0000 00e9 d901 000
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0
48bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 000
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b
fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0
0600 0000 e988 0100 00e9 c705 48bf 0e08 0600 0000 e988 010
f701 c645 f800 c645 f900 8301 0000 c645 f701 c645 f800 c64
fc00 740f c705 48bf 0e08 c645 fa02 807d fc00 740f c705 48b
0100 00e9 5901 0000 c645 0400 0000 e95e 0100 00e9 5901 000
c645 f900 c645 fa03 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b
fc00 750a c705 48bf 0e08 0500 0000 807d fc00 750a c705 48b
fe00 740f c705 48bf 0e08 0300 0000 807d fe00 740f c705 48b
0100 00e9 901 0000 c645 0600 0000 e90e 0100 00e9 0901 000
c645 0403 fa01 807d f701 c645 f800 c645 f901 c645 fa0
48bf 0e08 0400 0000 e9c4 fd00 750f c705 48bf 0e08 0400 000
0000 c645 f701 c645 f800 0000 00e9 df00 0000 c645 f701 c64
fa04 807d fc00 7410 807d c645 f900 c645 fa04 807d fc00 741
48bf 0e08 0700 0000 807d ff00 730a c705 48bf 0e08 0700 000
ff00 740f c705 48bf 0e08 fc00 7415 807d ff00 740f c705 48b
0000 00e9 9900 0000 c645 0600 0000 e99e 0000 00e9 9900 000
c645 f900 c645 fa05 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b
fc00 750a c705 48bf 0e08 0800 0000 807d fc00 750a c705 48b
fe00 7506 807d ff00 740c 0900 0000 807d fe00 7506 807d ff0
0600 0000 eb4b eb49 c645 c705 48bf 0e08 0600 0000 eb4b eb4
c645 f901 c645 fa02 807d f701 c645 f800 c645 f901 c645 fa0
5dc3 5589 e5c7 0540 bf0e 00b8 5400 0000 5dc3 5589 e5c7 0540
1800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558
3000 00b8 4500 0000 5dc3 0540 bf0e 0820 0000 00b8 4500 000
bf0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 0821 0000 00b
e5c7 0540 bf0e 0822 0000 0000 5dc3 5589 e5c7 0540 bf0e 082
5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 5589 e583 ec1
3000 a148 bf0e 0883 f809 48bf 0e08 0100 0000 a148 bf0e 088
3b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 e10b 08f
30c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f
3000 00e9 d901 0000 c645 0548 bf0e 0882 0000 00e9 d901 000
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0
18bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 000
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b
fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0
3600 0000 e988 0100 00e9 c705 48bf 0e08 0600 0000 e988 010

```

Entry point

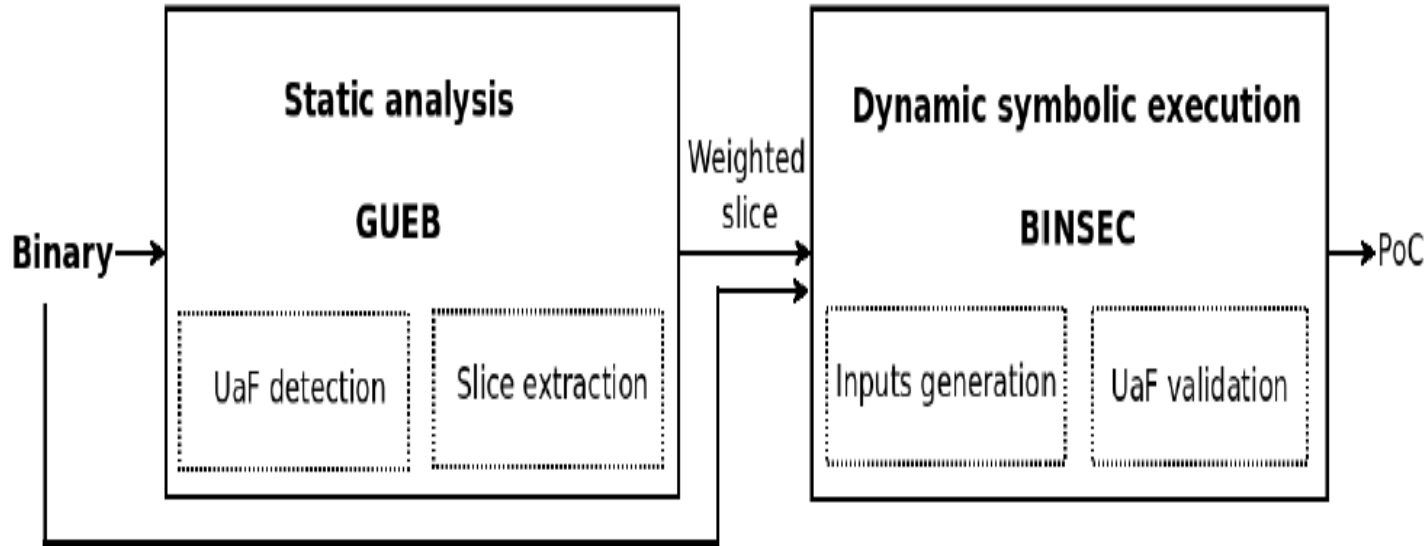
free

use



# CASE 1: COMPLEX VULNERABILITY DETECTION

[SSPREW'16](with Josselin Feist et al.)



**A Pragmatic 2-step approach**

- Step 1: incorrect but scalable
- Steps 1+2: scalable and correct



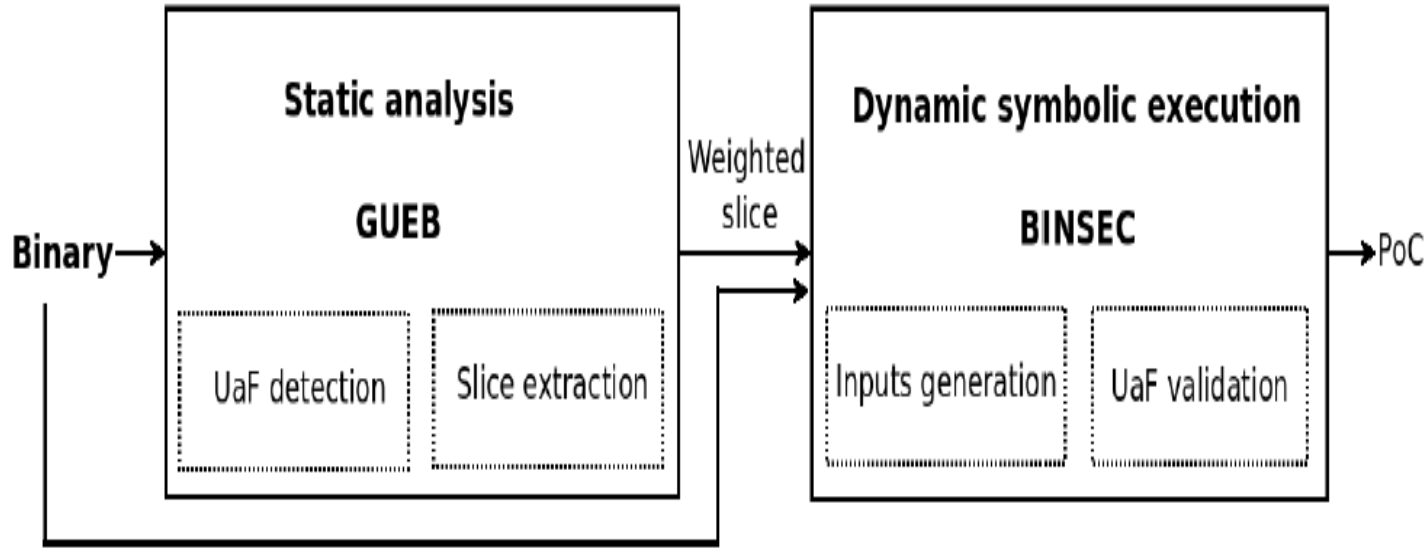
Find a needle in the heap!

```

4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558
0000 00b8 4500 0000 5dc3 0540 bf0e 0821 0000 00b8 4500 000
bf0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 0821 0000 00b
e5c7 0540 bf0e 0822 0000 0000 0000 5dc3 5589 e583 ec1
0000 a148 bf0e 0883 f809 48bf 0e08 0100 0000 a148 bf0e 088
8b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 e10b 08f
00c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f
0000 00e9 d901 0000 c645 0548 bf0e 0802 0000 0000 00e9 d901 900
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0
48bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 000
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b
fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0
0600 0000 e988 0100 00e9 c705 48bf 0e08 0600 0000 e988 010
f701 c645 f800 c645 f900 8301 0000 c645 f701 c645 f800 c64
fc00 748f c705 48bf 0e08 c645 fa02 807d fc00 740f c705 48b
0100 00e9 5901 0000 c645 0400 0000 e95e 0100 00e9 5901 000
c645 f900 c645 fa03 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b
fc00 7506 c705 48bf 0e08 0500 0000 807d fc00 750a c705 48b
fe00 740f c705 48bf 0e08 0300 0000 807d fe00 740f c705 48b
0100 0000 9901 0000 c645 0600 0000 e99e 0100 00e9 0901 000
e645 f400 0000 0000 f701 c645 f800 c645 f901 c645 fa0
48bf 0400 0000 0000 0000 7501 c705 48bf 0e08 0400 0000
0000 c645 f701 c645 f800 0000 00e9 d100 0000 c645 f701 c64
fa04 807d fc00 7410 807d c645 f900 c645 fa04 807d fc00 741
48bf 0e08 0700 0000 807d ff00 750a c705 48bf 0e08 0700 000
ff00 740f c705 48bf 0e08 fc00 7415 807d ff00 740f c705 48b
0000 00e9 9900 0000 c645 0600 0000 e99e 0000 00e9 9900 000
c645 f900 c645 fa05 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 f100 7410 807d fe00 750a c705 48b
fc00 750a c705 48bf 0e08 0800 0000 807d fc00 750a c705 48b
fe00 7506 807d ff00 740c 0900 0000 807d fe00 7506 807d ff0
0600 0000 eb4b eb49 c645 c705 48bf 0e08 0600 0000 eb4b eb4
c645 f901 c645 fa02 807d f701 c645 f800 c645 f901 c645 fa0
5dc3 5589 e5c7 0540 bf0e 00b8 5400 0000 5dc3 5589 e5c7 0540
4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558!
3000 00b8 4500 0000 5dc3 0540 bf0e 0821 0000 00b8 4500 0000
bf0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 0821 0000 00b8
e5c7 0540 bf0e 0822 0000 0000 0000 5dc3 5589 e5c7 0540 e0821
5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 5589 e583 ec10
0000 a148 bf0e 0883 f809 48bf 0e08 0100 0000 a148 bf0e 088:
8b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 e10b 08f:
30c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f:
3000 00e9 d901 0000 c645 0548 bf0e 0802 0000 00e9 d901 000:
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0:
48bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 000:
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b:
fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0:
3600 0000 e988 0100 00e9 c705 48bf 0e08 0600 0000 e988 010:
  
```

# CASE 1: COMPLEX VULNERABILITY DETECTION

[SSPREW'16](with Josselin Feist et al.)



```

4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558
0000 00b8 4500 0000 5dc3 0540 bf0e 0821 0000 00b8 4500 000
bf0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 0821 0000 00b
e5c7 0540 bf0e 0822 0000 0000 5dc3 5589 e583 ec10
5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 5589 e583 ec1
0000 a148 bf0e 0883 f809 48bf 0e08 0100 0000 a148 bf0e 088
8b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 e10b 08f
00c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f
0000 00e9 d901 0000 c645 0548 bf0e 0802 0000 00e9 d901 000
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0
48bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 000
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b
fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0
0600 0000 e988 0100 00e9 c705 48bf 0e08 0600 0000 e988 010
f701 c645 f800 c645 f900 8301 0000 c645 f701 c645 f800 c64
fc00 740f c705 48bf 0e08 c645 fa02 807d fc00 740f c705 48b
0100 00e9 5901 0000 c645 0400 0000 e95e 0100 00e9 5901 000
c645 f900 c645 fa03 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b
fc00 750a c705 48bf 0e08 0500 0000 807d fc00 750a c705 48b
fe00 740f c705 48bf 0e08 0300 0000 807d fe00 740f c705 48b
0100 0000 901 0000 c645 0600 0000 e99e 0100 00e9 0901 000
c645 045 fa01 807d f701 c645 f800 c645 f901 c645 fa0
48bf 400 0000 e07d f000 750a c705 48bf 0e08 0400 000
0000 c645 f701 c645 f800 0000 00e9 d100 0000 c645 f701 c64
5004 807d fc00 7410 807d c645 f900 c645 fa04 807d fc00 741
48bf 0e08 0700 0000 807d ff00 750a c705 48bf 0e08 0700 000
ff00 740f c705 48bf 0e08 fc00 7415 807d ff00 740f c705 48b
0000 00e9 9900 0000 c645 0600 0000 e99e 0000 00e9 9900 000
c645 f900 c645 fa05 807d f701 c645 f800 c645 f900 c645 fa0
fe00 750a c705 48bf 0e08 fc00 7410 807d fe00 750a c705 48b
fc00 750a c705 48bf 0e08 0800 0000 807d fc00 750a c705 48b
fe00 7506 807d ff00 740c 0900 0000 807d fe00 7506 807d ff0
0600 0000 eb4b eb49 c645 c705 48bf 0e08 0600 0000 eb4b eb4
c645 f901 c645 fa02 807d f701 c645 f800 c645 f901 c645 fa0
5dc3 5589 e5c7 0540 bf0e 00b8 5400 0000 5dc3 5589 e5c7 0540
4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558
0000 00b8 4500 0000 5dc3 0540 bf0e 0821 0000 00b8 4500 000
bf0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 0821 0000 00b
e5c7 0540 bf0e 0822 0000 0000 5dc3 5589 e5c7 0540 bf0e 082
5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 5589 e583 ec1
0000 a148 bf0e 0883 f809 48bf 0e08 0100 0000 a148 bf0e 088
8b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 e10b 08f
30c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f
0000 00e9 d901 0000 c645 0548 bf0e 0802 0000 00e9 d901 000
c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0
48bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 000
fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b
fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0
3600 0000 e988 0100 00e9 c705 48bf 0e08 0600 0000 e988 010
  
```

**A Pragmatic 2-step approach**

- Step 1: incorrect but scalable
- Steps 1+2: scalable and correct

- Find a few new CVEs
- Much better than AFL here



Find a needle in the heap!



# CASE 2: BINARY-LEVEL PROOF

**Not addressed by DSE**

- Cannot enumerate all paths

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>
    
```

The predicate is always true

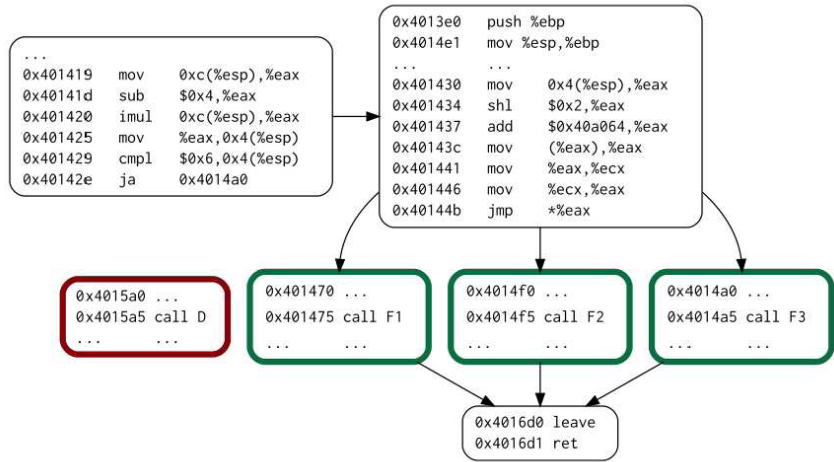
```

if (ax > bx) X = -1;
else X = 1;
    
```

```

OF := ((ax{31,31}#bx{31,31}) &
      (ax{31,31}#(ax-bx){31,31}));
SF := (ax-bx) < 0;
ZF := (ax-bx) = 0;
if (~ZF ^ (OF = SF)) goto 11
X := 1
goto 12
11: X := -1
12:
    
```

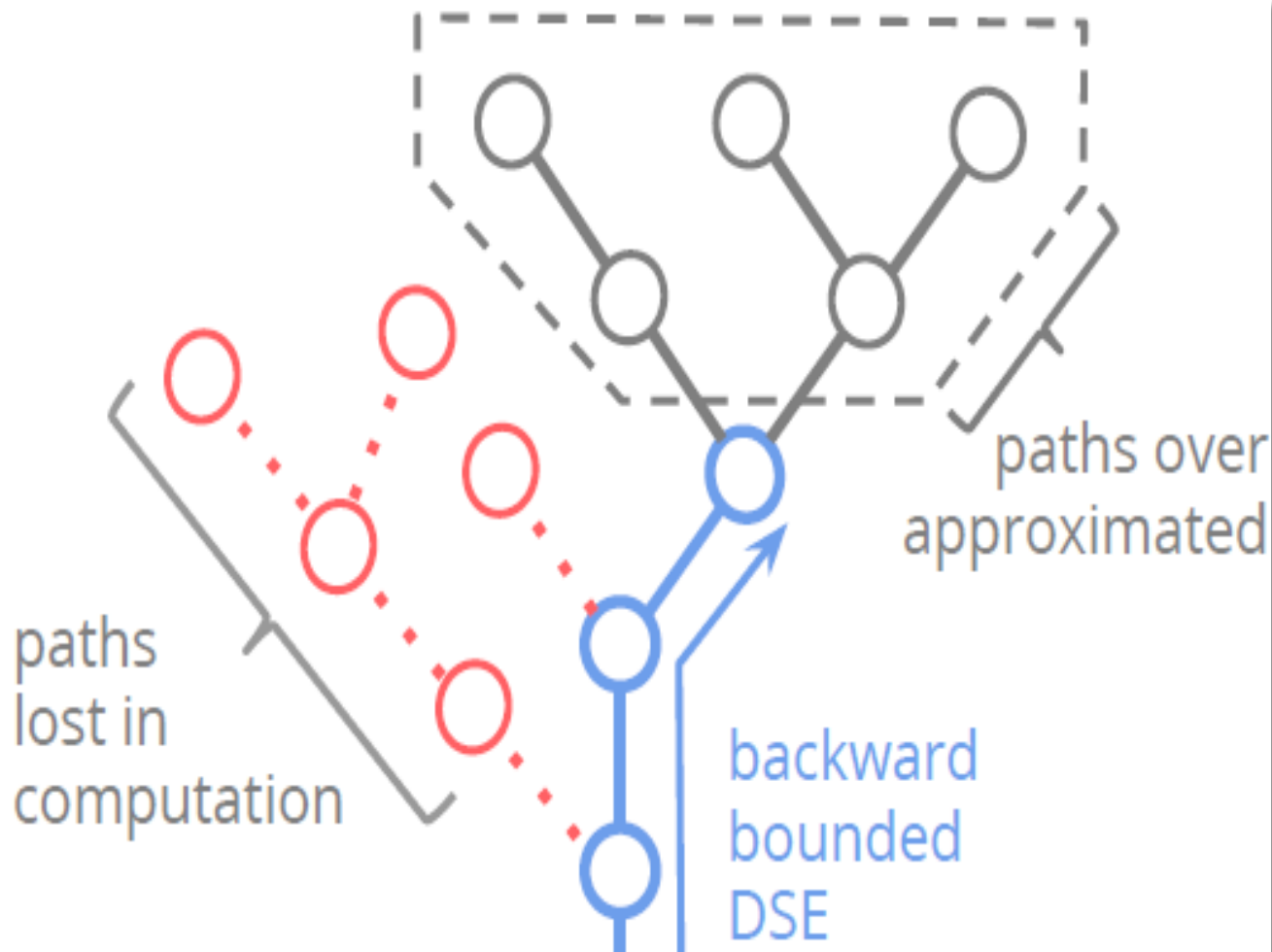
The two blocks are equivalent



With IDA + BINSEC

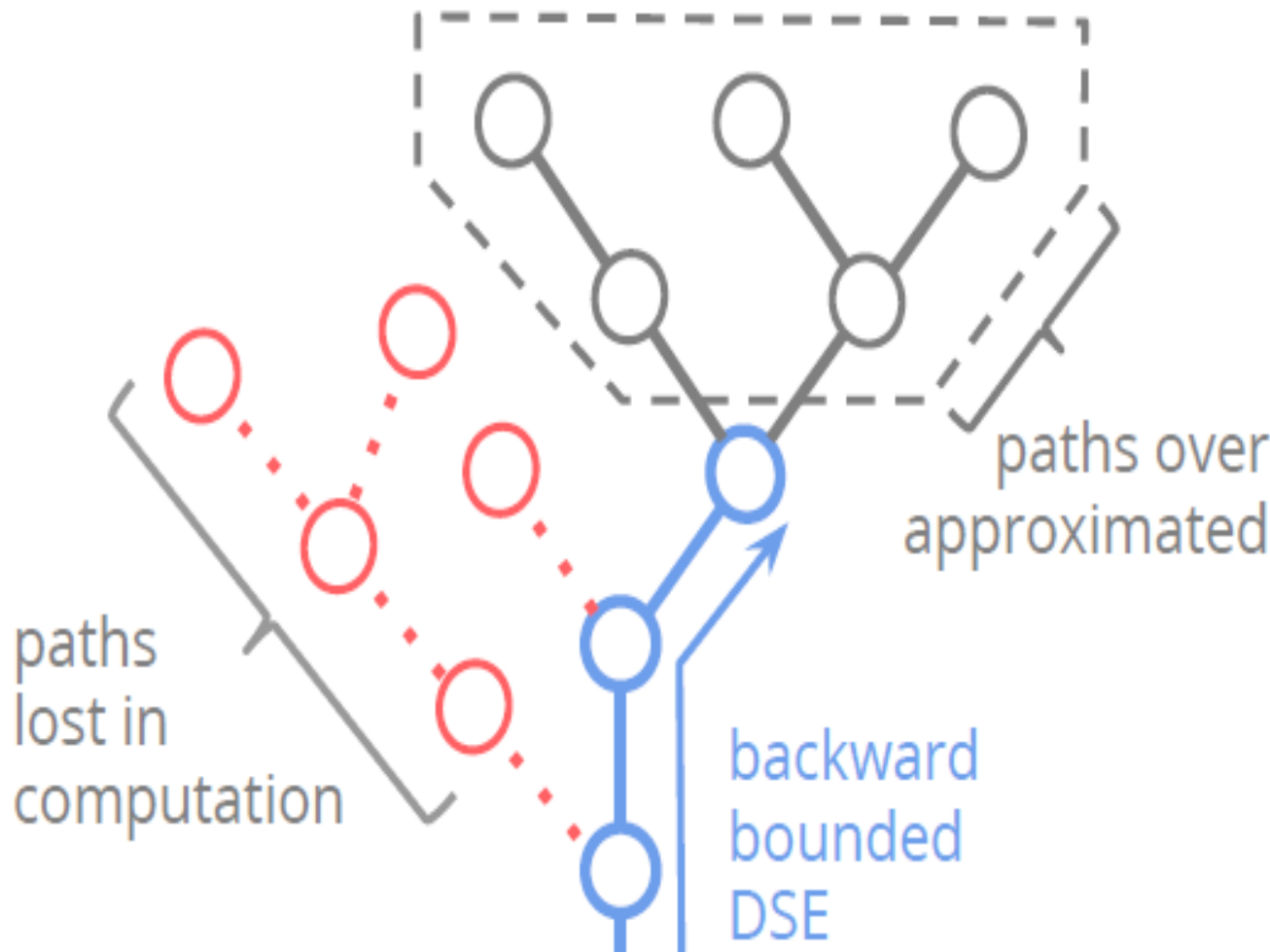
All jump targets are found

## Case 2: BINARY-LEVEL PROOF

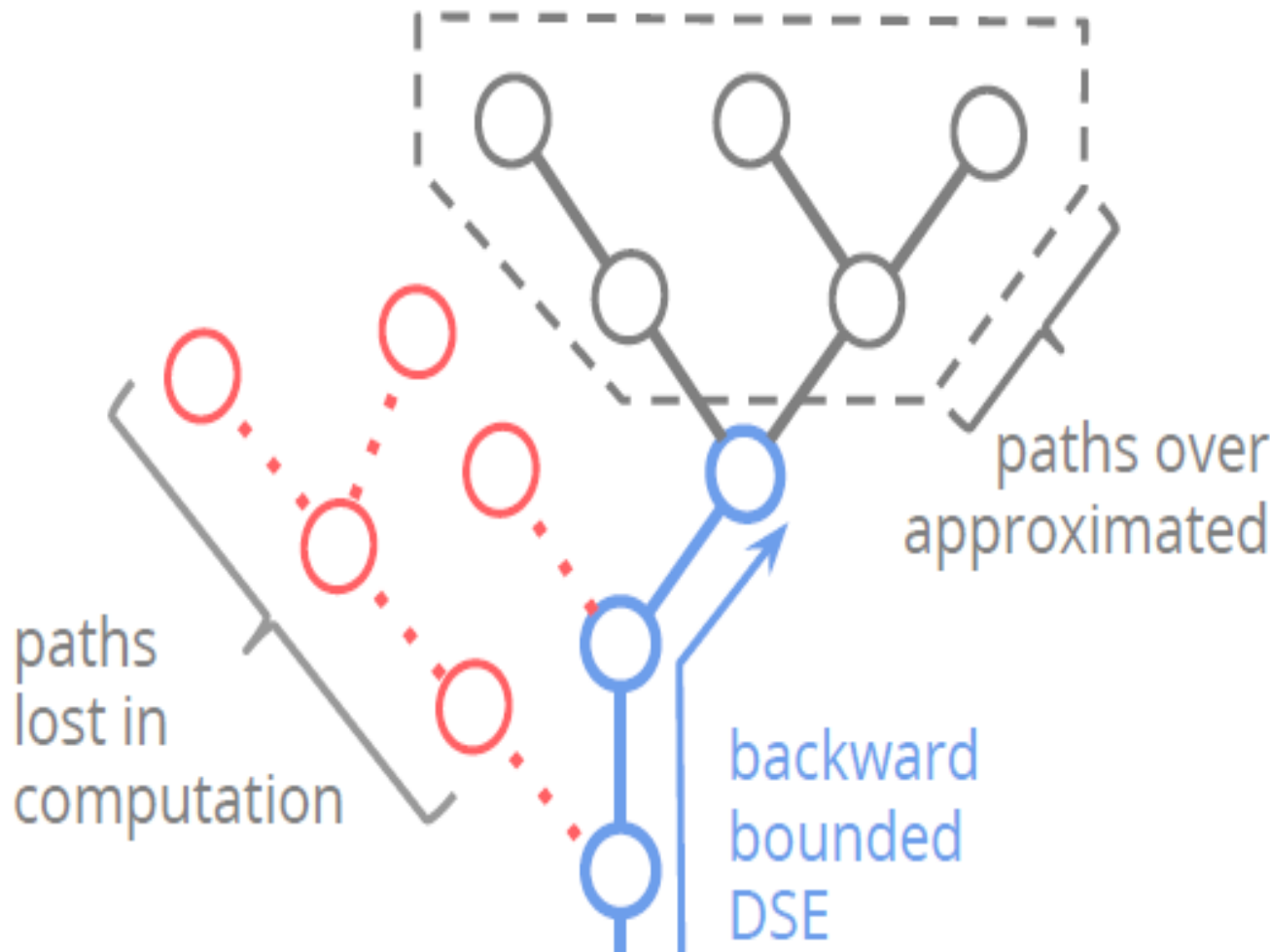


### Backward bounded SE

- Compute k-predecessors
- If the set is empty, no pred.
- Allows to **prove** things



- **False Negative:  $k$  too small**
  - *Missed proofs*
- **False Positive: CFG incomplete**
  - *Wrong proofs ?!*

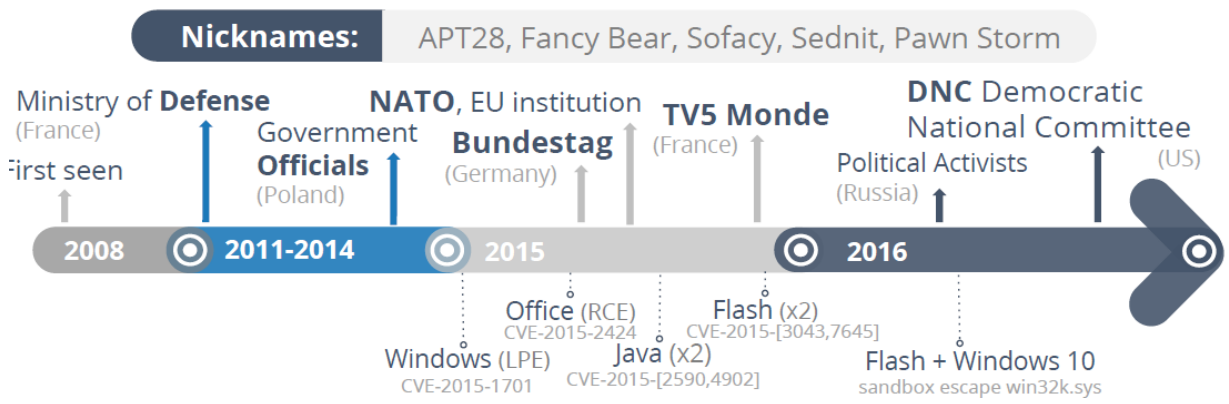


- **False Negative:  $k$  too small**
  - *Missed proofs*
- **False Positive: CFG incomplete**
  - *Wrong proofs*

- Low rate of wrong proofs
- Controlled XPs

# CASE-STUDY: THE XTUNNEL MALWARE [BH EU 16, S&P'17]

## -- part of DNC hack (with Robin David)



**Two heavily obfuscated samples**

- Many opaque predicates

**Goal: detect & remove protections**

- Identify 45% of code as spurious
- Fully automatic, < 3h



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

## CASE 3: finely tuning the technology

- SMT solvers are powerful weapons
- But (binary-level) security problems are terrific beasts
- **Need to adapt them!**

### Two examples

- Scalability [LPAR 2018, Benjamin Farinier]
- Robustness [CAV 2018, Benjamin Farinier]



# Example 1: scalability (reasoning about memory)

## Array theory

- Necessary to model memory

ROW rule may  
introduce case-splits

## Hard for solvers

- Case-splits

- Reading in  $a$  at index  $i \in \mathcal{I}$  :  $\text{select } a \ i$
- Writing in  $a$  an element  $e \in \mathcal{E}$  at index  $i \in \mathcal{I}$  :  $\text{store } a \ i \ e$

$\text{select} : \text{Array } \mathcal{I} \ \mathcal{E} \rightarrow \mathcal{I} \rightarrow \mathcal{E}$

$\text{store} : \text{Array } \mathcal{I} \ \mathcal{E} \rightarrow \mathcal{I} \rightarrow \mathcal{E} \rightarrow \text{Array } \mathcal{I} \ \mathcal{E}$

$\forall a \ i \ e. \text{select} (\text{store } a \ i \ e) \ i = e$

$\forall a \ i \ j \ e. (i \neq j) \Rightarrow \text{select} (\text{store } a \ i \ e) \ j = \text{select } a \ j$

Prevalent in software analysis

- Modelling memory
- Abstracting data structure  
(map, queue, stack...)

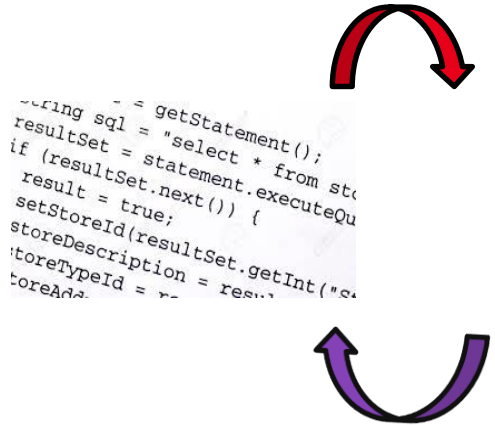
Hard to solve

- NP-complete
- ROW may require case-splits



# Not pure theory!

Reverse of a ASPACK-protected code



```

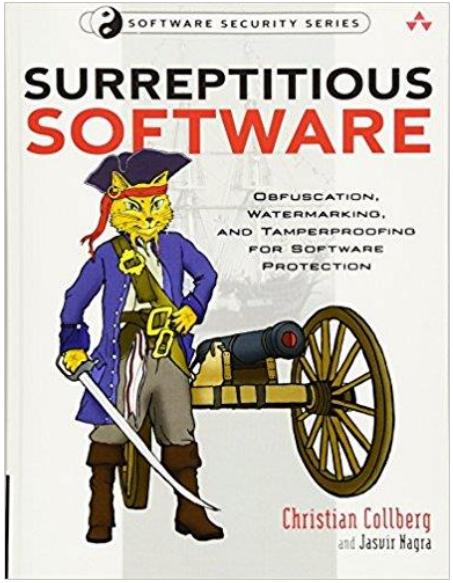
lists($NDckZAMTQGqlyz)}{$mTuzXWErlmR->set_sensitive(false);}if($!j=llGLMCMbMm1+1){$HuePILknsBRY
BOLKUYFWI=1}{if($OorGLlhtemPke=""){$KlZFFVKHdyZBp;switch($OorGLlhtemPke){case1:$KlZFFVKHdy
um;$OorGLlhtemPke;}}functionobdRelgeOymhh($pghtWakLgMD7d){global$Image;$ORMLLenz;global$D3YFgcl
P;$screen_height;$RedHlBLAcDrpxKc[1]*;$RedHlBLAcDrpxKc[0];}else{.$oeyScFrZMcQP-$screen_height;$RedHlBL
"ru","2","1","was";$EQAvvKCKPCIMV=$sqlite_query($MERFSVesYem,"SELECTlageFROMlageWHEREld=0");i
"ru","2","1","was","q");for($!=$!;$!<=8;$!+){$BwVwchPFTcEd-$OorGLlhtemPke[$!]."M";$+;+if($
KtSulohm")} {$!$mZyBrmtyjInBo}=newGetRadLibOuton(nul1,"",0);$LWUWbMVKtSulohm={$!$mZyBrmtyjInBo};}elst
gQL($Image_file){$pghtWakLgMD7d=$Image_file;$OorGLlhtemPke=arr('1a','mo','ro','1a','mm','1a','nu
dlig{$!B=BLAZPmFPZYU,$gbeycQSLKbFFm,$mVMEtGLGhrVCSjz,$zCjwzGcWUleM)}{.$!$mYlMpfTFAQEL=Imagegetfbb
1[1]*.$!tchPUnQVedz-$!$mYlMpfTFAQEL[0]*.$!$mYlMpfTFAQEL[1]*.$!$mYlMpfTFAQEL[2]*.$!$mYlMpfTFAQEL[3]*.$!$m
cfcQ;$zrBrcMxVPUjMBo["h">$KHeVgncDaxz7Rr;$zrBrcMxVPUjMBo["w">$YUgexVWLdAGd];return$rBrcMxVPUjMBo;
WlcadZyrtz-$zrBrcMxVPUjMBo[1];if($gbeycQSLKbFFm=0){$LMEPLLskpTlv=18;}else{$!$LMEPLLskpTlv=8;}$!$LME
UWNTLTLvEgth=Imagey($mABmHCoyglt1)/2-Imagey($mLVSpumZuhU)/2;if($mVMEtGLGhrVCSjz="u"){.$!$mVMEtGLG
upmZuhU)/2;}if($DudgKydkWkCBZ="r"){.$!$ogbPKcrLTQz2=Imagey($mABmHCoyglt1)-Imagey($mLVSpumZuhU
QkVQhlp["g");$oovG5j5YMSMEjt=$!$QudgKydkWkCBZ["b");}if($LubboQuohpBom="height"){.$!$QudgKydkWkCBZ
DwK=255;}if($oovG5j5YMSMEjt>127){$oovG5j5YMSMEjt=18;}else{$oovG5j5YMSMEjt=255;}if($!$TReBQZDZF
EufmKZIGI-$NDckZAMTQGqlyz;$!$BrcPmFPZYU=$getImagesze;$oovG5j5YMSMEjt=255;$!$oovG5j5YMSMEjt=$!$TReBQZDZF
($mVQcZ25QKkMx=Imagey($mABmHCoyglt1)/4*9*9*$oovG5j5YMSMEjt)/($mVQcZ25QKkMx=Imagey($mABmHCoyglt1)
uhU)-$HDXowuyPofrFk;if($mVMEtGLGhrVCSjz="o"){.$!$UvMEtGLGhrVCSjz=Imagey($mABmHCoyglt1)/2-Imagey($mLV
($mABmHCoyglt1)/2-Imagey($mLVSpumZuhU)/2);}$!$UvMEtGLGhrVCSjz=Imagey($mABmHCoyglt1)/2-Imagey($mLV
->set_text(""));}$!$FrSLsBvBzD0-$GLGRAL;$!$FrSLsBvBzD0->set_text("");}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk
MNTLTLvEgth=Imagegetfbb1[1]*.$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk
XIGBmFdvbomK."WHEREld=0");functionEoVgEqkqkLs($zBwGSKDdGhW,$JfCRfmlBvDmP,$JfCzscrSMT3Dp
PLlSkpDTlv->set_text();if($mVMEtGLGhrVCSjz="o"){.$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk;}$!$HDXowuyPofrFk

```

Huge formula obtained by  
dynamic symbolic execution

293 000 select

24 hours of resolution!



- Obsidium
- JD Pack
- WinUpack
- Expressor
- PE Lock
- PE Compact
- Armadillo
- Packman
- EP Protector
- ACProtect
- TELockSVK
- Yoda's Crypter
- Mew
- Neolite
- UPX MoleBox
- FSGUpack
- Crypter
- Yoda's Protector
- ASPack
- BoxedApp
- Petite
- nPack
- PE Spin
- Enigma
- Setisoft
- Themida
- RLPack
- Mystic
- VMProtect

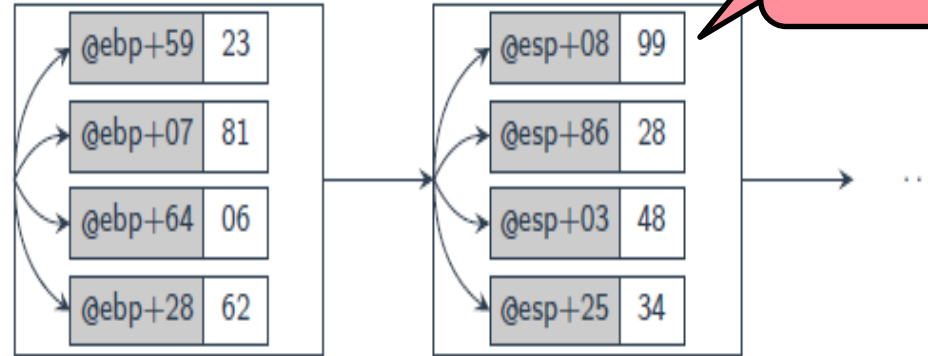
## Example 1: scalability (reasoning about memory)

- **Our goal: dedicated formula preprocessing to remove « RoW »**
  - Problem 1: standard « list »-representation for logical arrays induces a quadratic time preprocessing → prohibitive
  - Problem 2: need to cheaply but precisely reason about index equalities



# Example 1: scalability (reasoning about memory)

- Dedicated data structure (list-map)
- Tuned for base+offset access
- Linear complexity



- Scale
- Good when only few bases

Propagate “variable+constant” terms

- If  $y \triangleq z + 1$  then  $x \triangleq y + 2 \rightsquigarrow x \triangleq z + 3$
- Together with associativity, commutativity...

- Reduce the number of bases

- Prove disequalities between different bases

Associate to every indices  $i$  an abstract domain  $i^\#$

- If  $i^\# \sqcap j^\# = \perp$  then  $(a[i] \leftarrow e)[j] = a[j]$
- Integrated in the list-map representation

# Tuning the solver: array formula simplification [LPAR 2018]

## with Benjamin Farinier

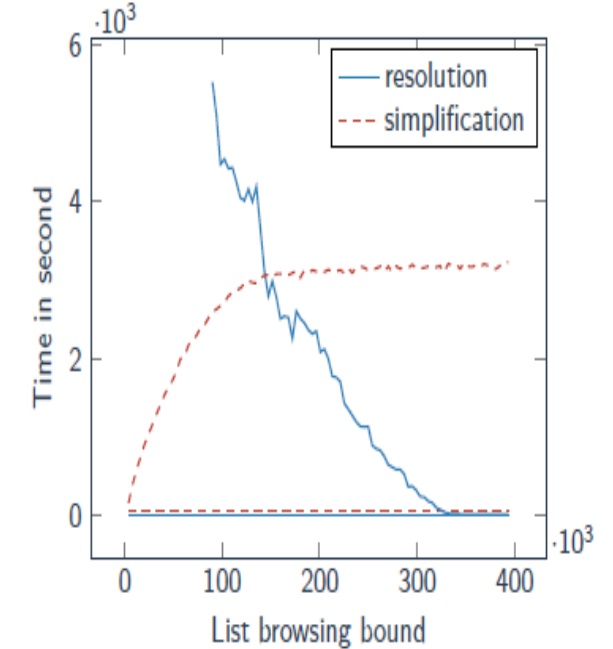
- **Makes the difference!**

no block cypher	#select		
	Z3	all arrays	non initial
no simplification	0 606.7	1 448 301	1 448 001
list-16	0 501.0	1 075 358	1 052 786
list-256	0 371.9	807 778	762 673
map	0 370.5	807 778	762 673
LMBN	0 46.0	65 788	5 044

- Huge formula obtained by dynamic symbolic execution
- 293 000 select
- 24 hours of resolution !

### Using LMBN

- #select reduced to 2 467
- 14 sec for resolution
- 61 sec for preprocessing



### Using list representation

- Same result with a bound of 385 024 and beyond...
- ...but 53 min preprocessing

## Example 2: robust symbolic execution

- **Standard symbolic reasoning** may produce **false positive**

What?!!

Safety is not security ...

- **for example here:**
  - SE will try to solve  $a * x + b > 0$
  - May return  $a = -100, b = 10, x = 0$
- **Problem: x is not controlled by the user**
  - If x change, possibly not a solution anymore
  - Example:  $(a = -100, b = 10, x = 1)$

```
int main () {  
    int a = input ();  
    int b = input ();  
  
    int x = rand ();  
  
    if (a * x + b > 0) {  
        analyze_me();  
    }  
    else {  
        ...  
    }  
}
```



## Example 2: robust symbolic execution

- Standard symbolic reasoning may produce **false positive**

- Actually, need to solve

$$\forall x. ax + b > 0$$

- Quantified formula
- SMT solvers bad for that

```
int main () {  
    int a = input ();  
    int b = input ();  
  
    int x = rand ();  
  
    if (a * x + b > 0) {  
        analyze_me();  
    }  
    else {  
        ...  
    }  
}
```

## Our solution: reduce quantified formula to the quantifier-free case

- Approximation
- But reuse the whole SMT machinery

### Key insights:

- independence conditions
- formula strengthening

- Quantified reachability condition

①  $\forall x. ax + b > 0$

- Taint variable constraint

②  $a^* \wedge b^* \wedge \neg x^*$  ( $a^*, b^*, x^*$ : fresh boolean variables)

- Independence condition

③  $((a^* \wedge x^*) \vee (a^* \wedge a = 0) \vee (x^* \wedge x = 0)) \wedge b^*$

④  $((\top \wedge \perp) \vee (\top \wedge a = 0) \vee (\perp \wedge x = 0)) \wedge \top$

⑤  $a = 0$

- Quantifier-free reachability condition

⑥  $(ax + b > 0) \wedge (a = 0)$

# Example: robustness and quantification

	SE classic		
	SAT correct	SAT incorrect	UNSAT or UNKNOWN
Boolector	12	11	1
CVC4	7	9	8
Yices	7	11	6
Z3	12	12	0

	SE robust		
	SAT correct	SAT incorrect	UNSAT or UNKNOWN
Boolector	N/A	N/A	24
CVC4	5	0	19
Yices	N/A	N/A	24
Z3	7	0	17

	SE robust + elim.		
	SAT correct	SAT incorrect	UNSAT or UNKNOWN
Boolector	12	0	12
CVC4	7	0	17
Yices	7	0	17
Z3	12	0	12

- set of crackme challenge
- compare true and false positive



## Example: robustness and quantification

### Back to 28: GRUB2 Authentication Bypass

- Original version: press Backspace 28 times to get a rescue shell
- Case study: same vulnerable code turned into a crackme challenge

- SE classic:  
incorrect solution
- SE robust:  
solvers TIMEOUT

- SE robust + elim.:  
correct solution in 80s
- SE robust + elim. + simpl.:  
correct solution in 30s



- The success of formal methods for safety
- Why binary-level security analysis?
- The hard journey from source-level safety to binary-level security
- Our approach
- **Conclusion**



# WHERE ARE WE?

- Explore
- Prove
- Simplify



Find a needle in the heap!



Obsidium  
JD Pack  
WinUpack  
Expressor PE Compact  
Armadillo  
Packman  
EP Protector  
ACProtect  
TELockSVK  
Yoda's Crypter  
New  
Neolite  
UPX MoleBox  
FSG Upack  
Crypter  
ASPack  
SolvedApp  
Petite  
nPack PE Spin  
Enigma  
Themida  
RLPack  
Mystic VMProtect

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

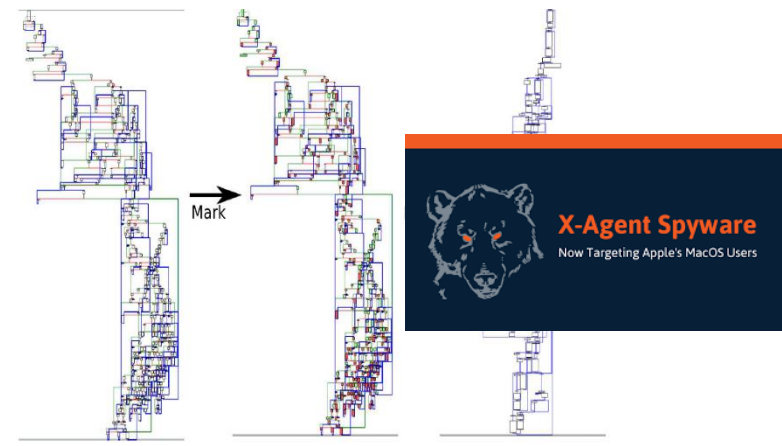


## Find (new) CVEs

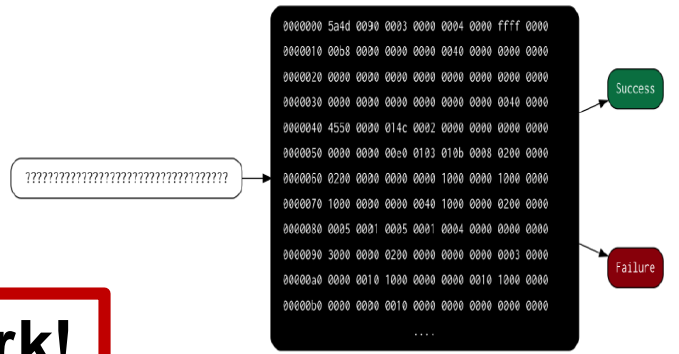
## Revert protections

### GRUB2 CVE 2015-8370

Elevation of privilege  
Information disclosure  
Denial of service



- Semantic approaches can work!



- **Robustness & precision are essential**
  - DSE is a good starting point
  - dedicated robust and precise (but not sound) static analysis are feasible
- **Can be adapted beyond the basic reachability case**
  - variants
  - combination with other techniques
- **Loss of guarantees**
  - Accept ... But control!
  - Look for « correct enough » solutions
- **Finely tune the technology**
  - Tools for safety are not fully adequate for security

# SECURITY IS NOT SAFETY

## • Source-level SAFETY

- Model: High-level language
- Properties: safety
- Algorithm: full correctness, possible help from user

- strong incentive to human assistance
- spec., parameter tuning, etc.

## • Binary-level SECURITY

- Model: **binary-level code, possibly adversarial, + attacker**
- Properties: safety, **k-safety, « bugs vs vulnerabilities »**
- Algorithm: **robust & precise enough**, fully automated

- no human assistance
- low tolerance to false positive



# SECURITY IS NOT SAFETY

## • Source-level SAFETY

- Model: High-level language
- Properties: safety
- Algorithm: full correctness, possible help from user

- strong incentive to human assistance
- spec., parameter tuning, etc.

## • Binary-level SECURITY

- Model: **binary-level code**, possibly adversarial, + attacker
- Properties: **safety**, **k-safety**, « bugs vs vulnerabilities » **[robust solutions]**
- Algorithm: **robust & precise enough**, fully automated

- no human assistance
- low tolerance to false positive



# CONCLUSION & TAKE AWAY



- **Binary-level security analysis**
  - Many applications, many challenges
  - Current syntactic and dynamic methods are not enough
- **Formal methods can help ... but must be strongly adapted**
  - [Complement existing approaches]
  - Need **robustness** and scalability!
  - Acceptable to lose both correctness & completeness – in a **controlled** way
  - Much better if **specifically tuned** for the problem at hand
  - **New challenges and variations, many things to do!**
- **Thanks for your attention!**



---

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](http://www-list.cea.fr)

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