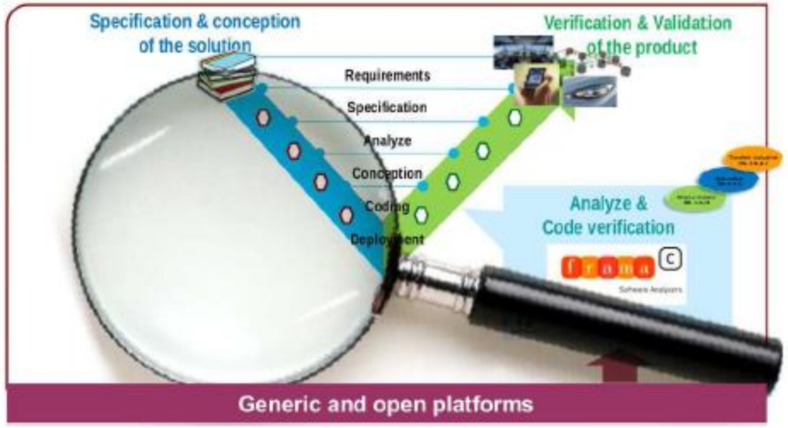
DEOBFUSCATION: SEMANTIC ANALYSIS TO THE RESCUE

Sébastien Bardin (CEA LIST)
Robin David (CEA LIST, QuarksLab)
Jean-Yves Marion (LORIA)

ABOUT MY LAB @CEA [Paris-Saclay, 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



frama C Software Analyzers

GATeL

PathCrawler

UNISIM

POLAR SSL

IN A NUTSHELL

- **Challenge: malware *deobfuscation***
- **Standard techniques (dynamic, syntactic) not enough**

- ***Semantic methods can help* [obfuscation preserves semantic]**
  - Yet, need to be strongly adapted (robustness, precision, efficiency)
- ***A tour on how symbolic methods can help***
  - *Explore and discover*
  - *Prove infeasibility*   *[S&P 2017]*
  - *Simplify* (not covered here)

OUTLINE

- **Context**
  - *Malware comprehension*
  - *Semantic analysis*
- **The hard journey from source to binary**
  - *Explore & Discover*
  - *Prove infeasibility*
- **A few case-studies**
- **Conclusion**

CONTEXT: MALWARE COMPREHENSION

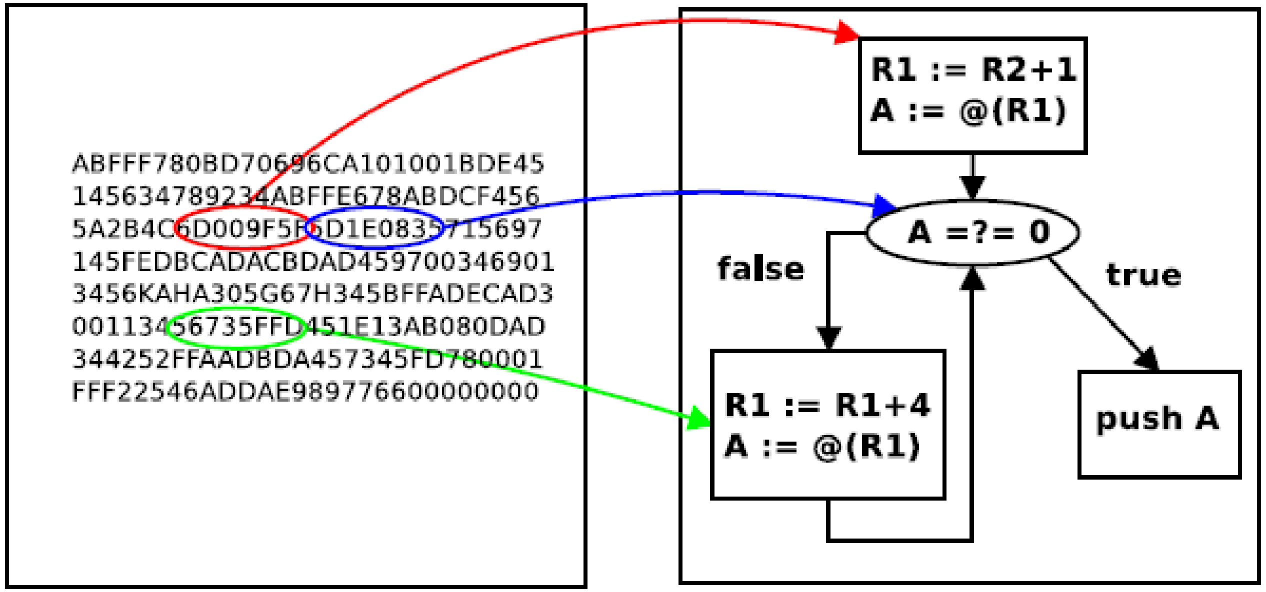
APT: highly sophisticated attacks
- **Targeted malware**
- **Written by experts**
- Attack: 0-days
- Defense: stealth, **obfuscation**
- **Sponsored by states or mafia**

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**

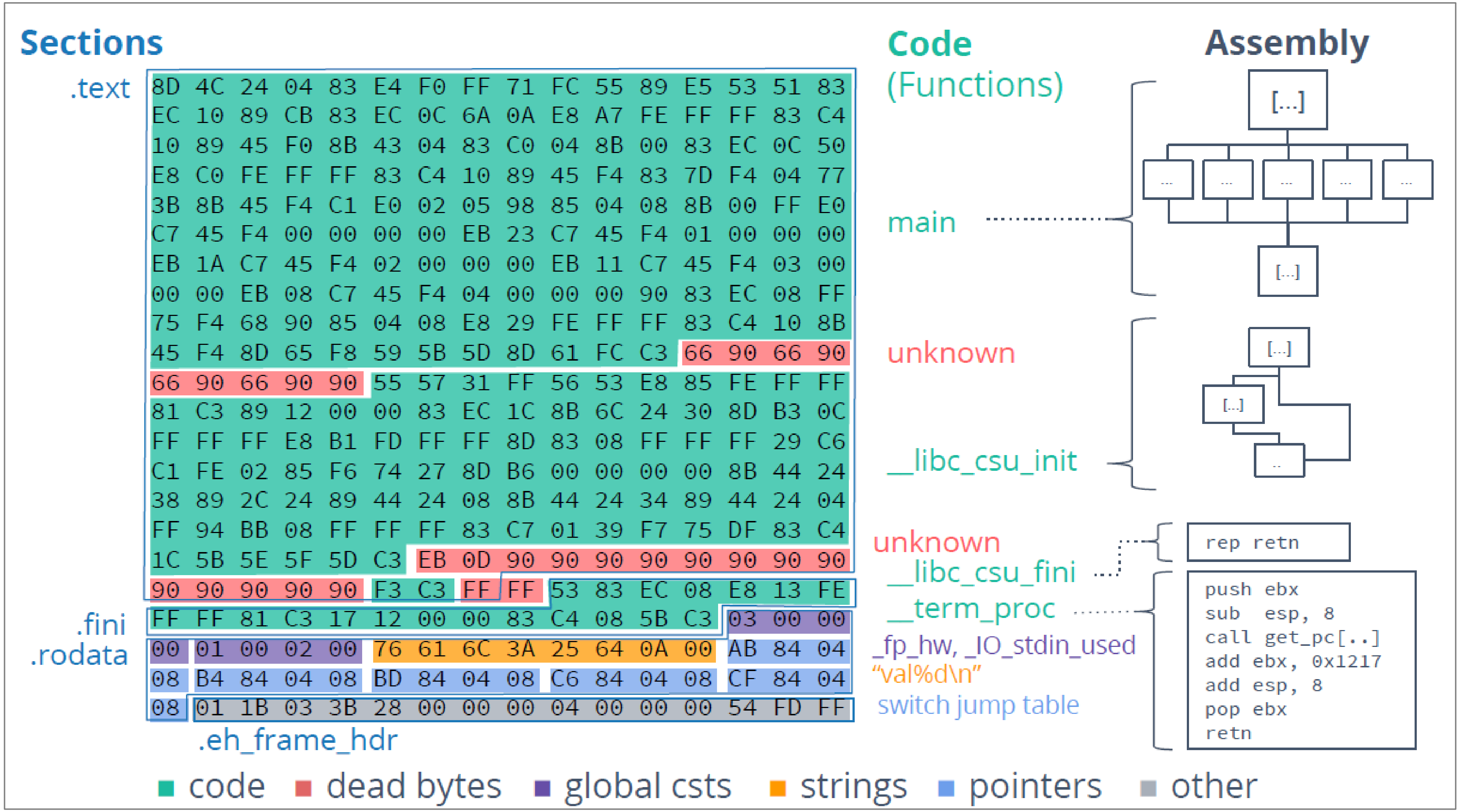
CHALLENGE: CORRECT DISASSEMBLY



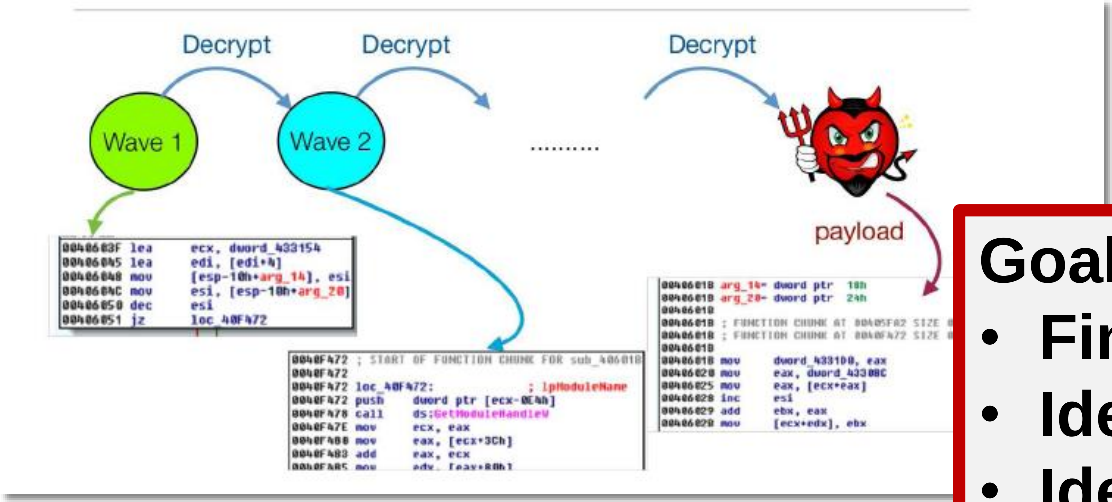
Basic reverse problem
- **aka model recovery**
- **aka CFG recovery**

CAN BE TRICKY!

- code – data
- dynamic jumps (jmp eax)



REVERSE CAN BECOME A NIGHTMARE (OBFUSCATION)



Goal: help malware comprehension

- **Find real parts of the code**
- **Identify and simplify protections**
- **Ideal = revert protections**

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

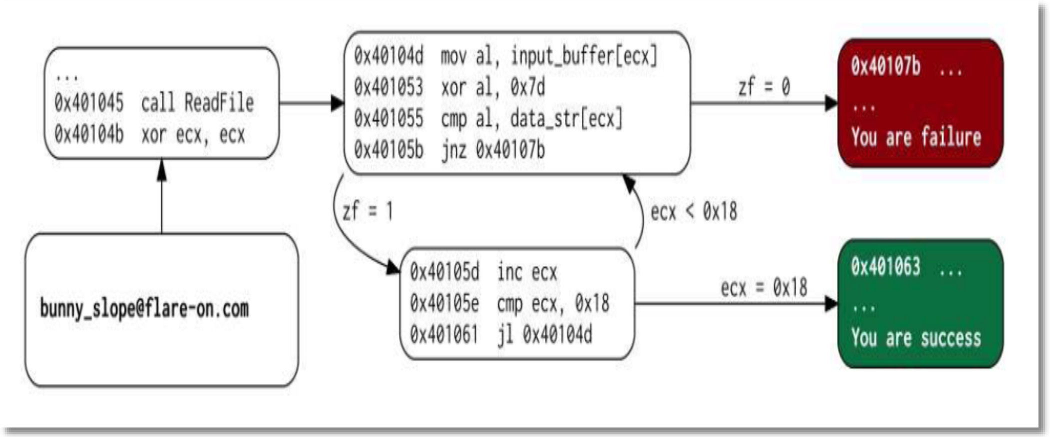
(for any value of x, y in modular arithmetic)

↧

```
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>
```

Obfuscation: make a code hard to reverse

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



EXAMPLE: OPAQUE PREDICATE

Constant-value predicates

(always true, always false)

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

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>
```

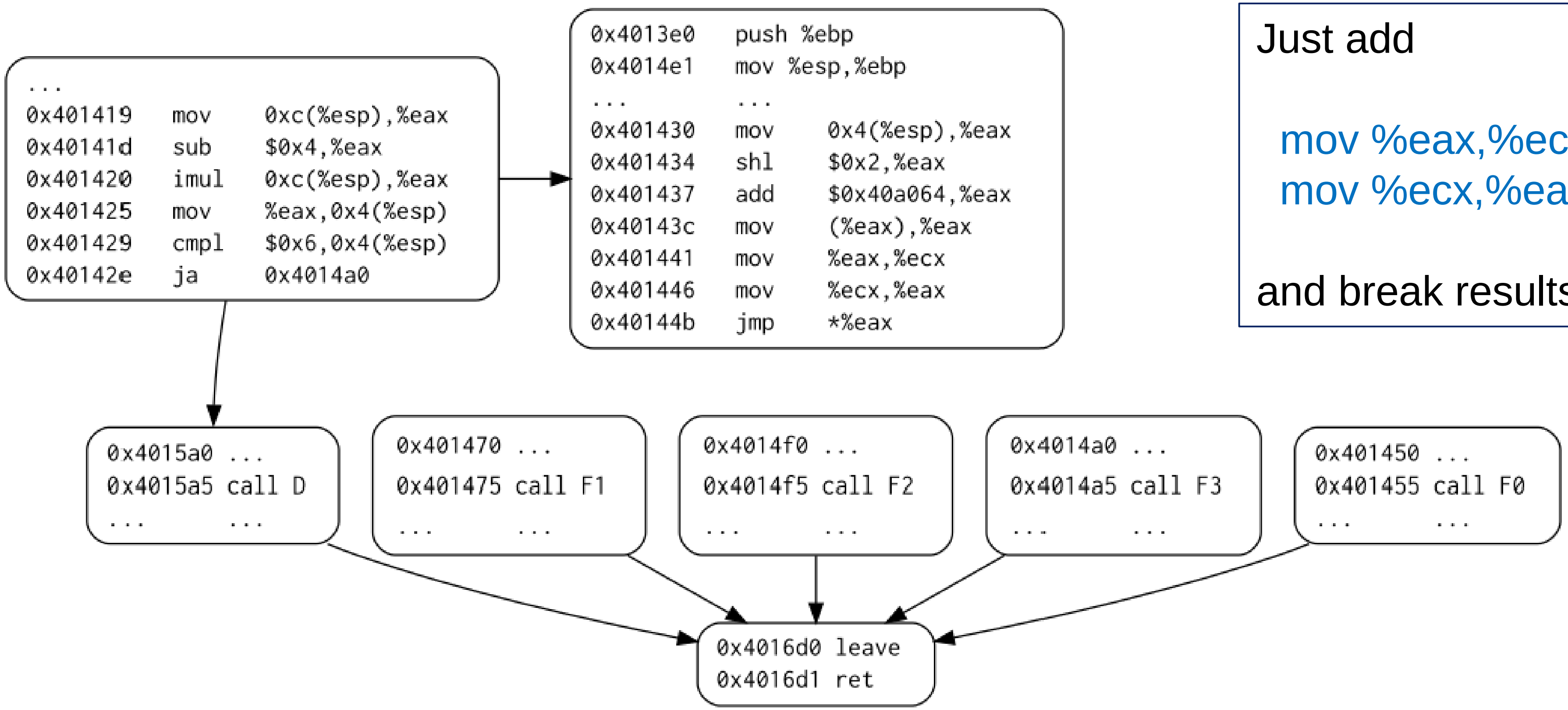
EXAMPLE: STACK TAMPERING

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	[...]

STATE-OF-THE-ART TOOLS ARE NOT ENOUGH



With IDA

Just add

mov %eax,%ecx
mov %ecx,%eax

and break results

- **Static (syntactic): too fragile**
- **Dynamic: too incomplete**

THE SITUATION

- Malware deobfuscation is necessary
- Malware deobfuscation is highly challenging
- Standard tools are not enough – experts need better help!

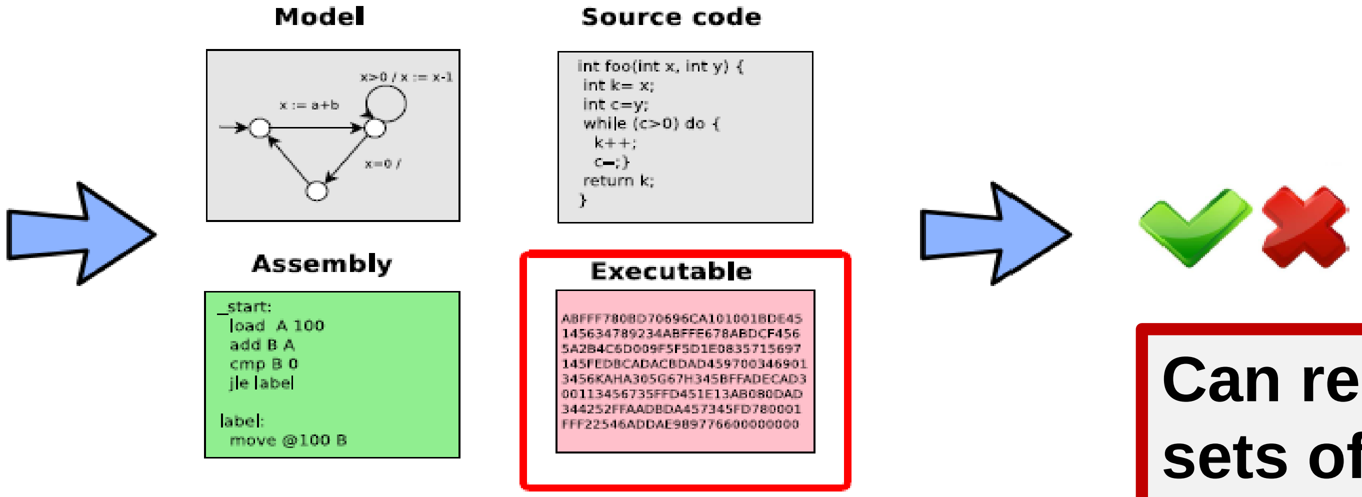
- Static (syntactic): too fragile
- Dynamic: too incomplete

SOLUTION? BINARY-LEVEL SEMANTIC ANALYSIS

Semantic preserved by obfuscation

Semantic tools help make sense of binary

- Develop the next generation of binary-level tools !
- motto : leverage formal methods from safety critical systems



Can reason about sets of executions
- **find rare events**
- **prove infeasibility**

Advantages

- more robust than syntactic
- more thorough than dynamic

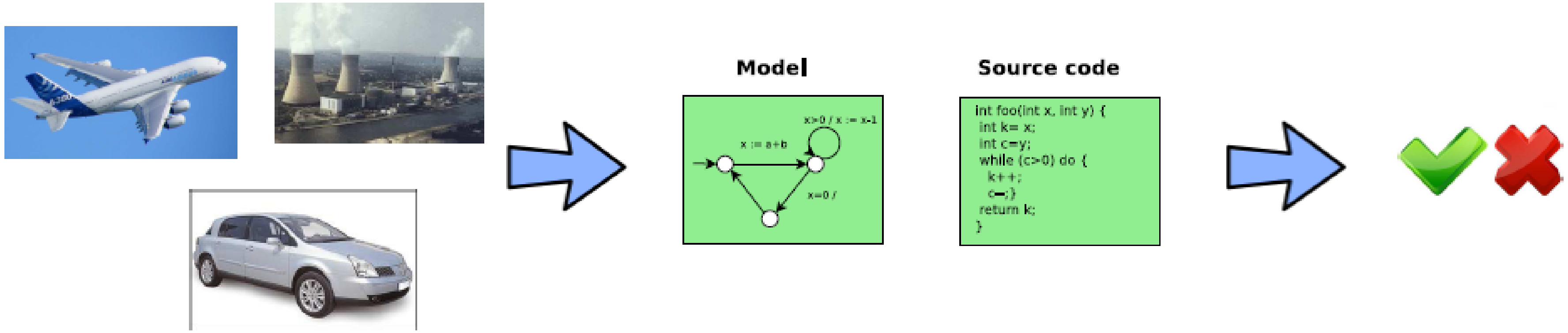
Challenges

- source-level ↦ binary-level
- safety ↦ security
- many (complex) architectures

<*En aparté*> 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
- φ : property to be checked
- $\models$: algorithmic check

Kind of properties

- absence of runtime error
- pre/post-conditions
- temporal properties

<*En aparté*> A DREAM COME TRUE … IN CERTAIN DOMAINS

Industrial reality in some key areas, especially safety-critical domains

- hardware, aeronautics [airbus], railroad [metro 14], smartcards, drivers [Windows], certified compilers [CompCert] and OS [Sel4], etc.

Ex : Airbus

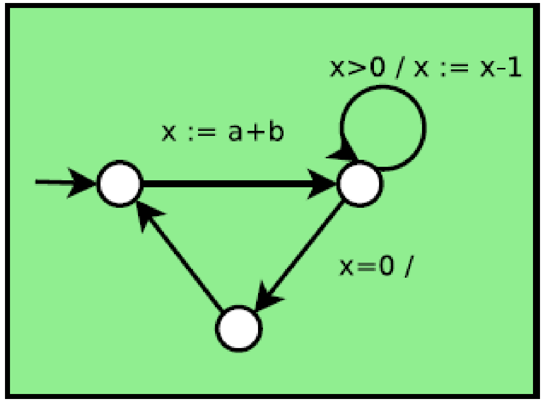
Verification of

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

* : by CEA DILS/LSL

NOW: BINARY-LEVEL ANALYSIS & OBFUSCATION

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 HARD JOURNEY FROM SOURCE TO BINARY

Low-level semantics of data

- machine arithmetic, bit-level operations, untyped memory
- ▶ difficult for any state-of-the-art formal technique

Low-level semantics of control

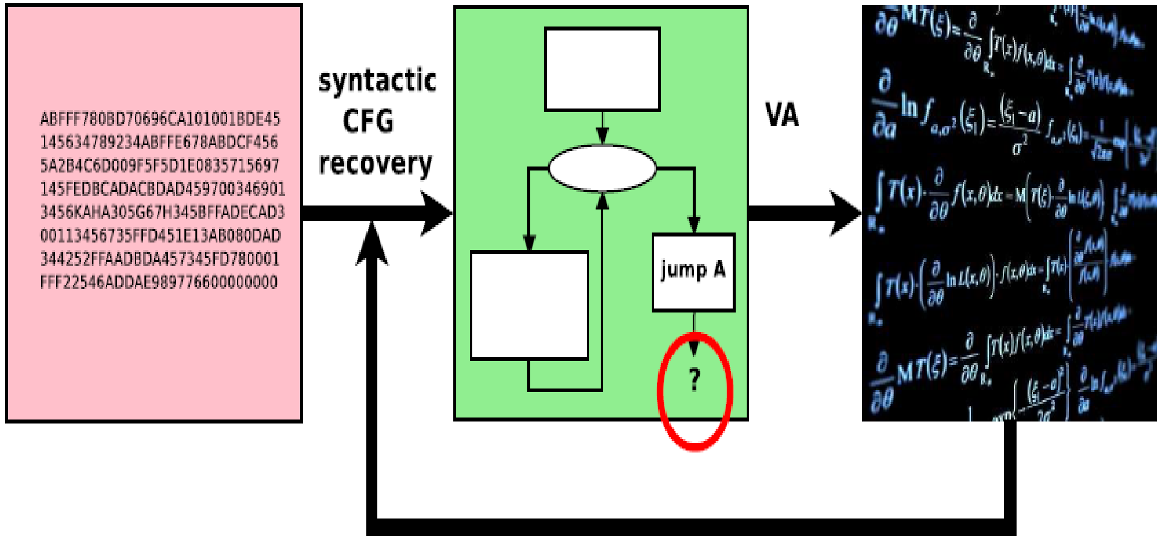
- no distinction data / instructions, dynamic jumps (`jmp eax`)
- no (easy) syntactic recovery of Control-Flow Graph (CFG)
- ▶ violate an implicit prerequisite for most formal techniques

Diversity of architectures and instruction sets

- support for many instructions, modelling issues
- ▶ tedious, time consuming and error prone

Wanted
- robustness
- precision
- scale

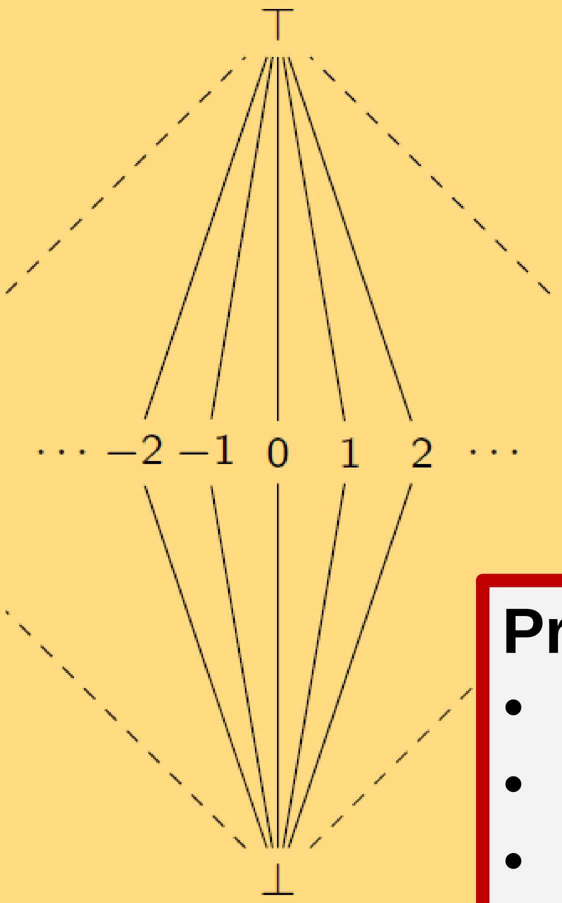
<En aparté> STATIC SEMANTIC ANALYSIS IS VERY VERY HARD ON BINARY CODE



Framework : abstract interpretation

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

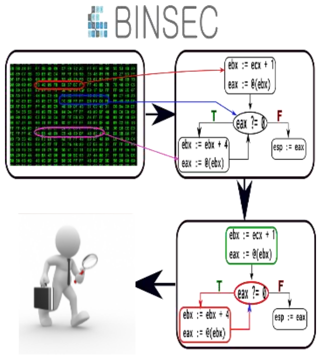
Generalize constant propagation

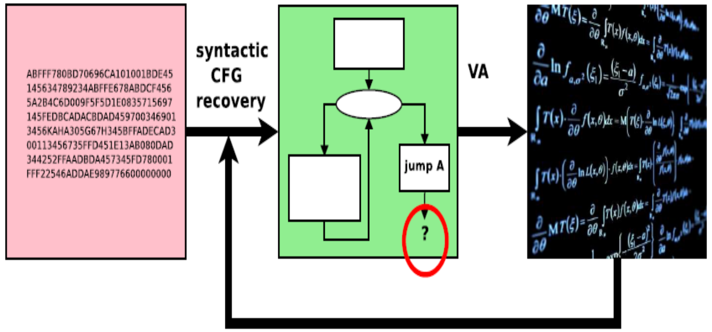


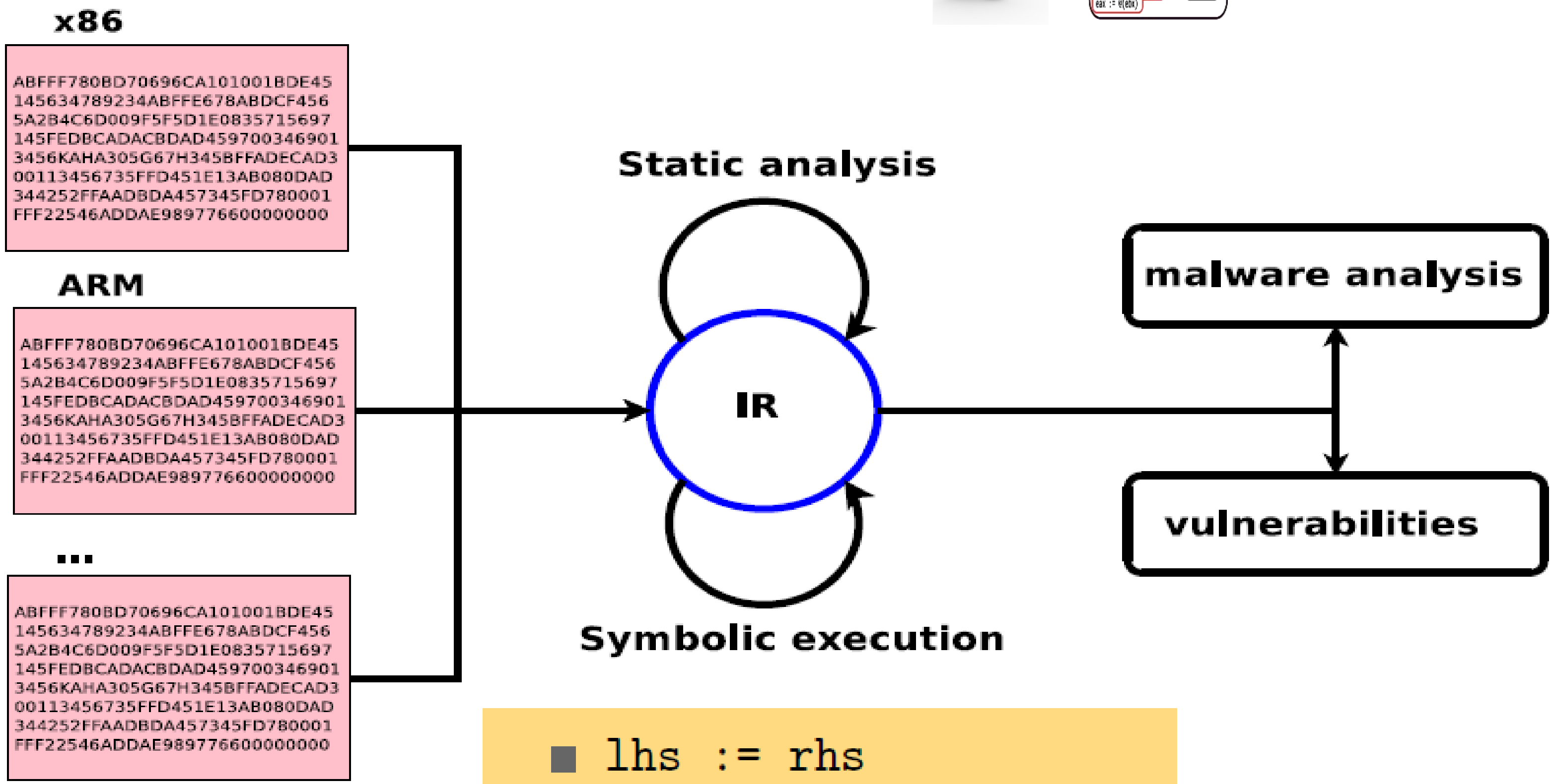
Problems

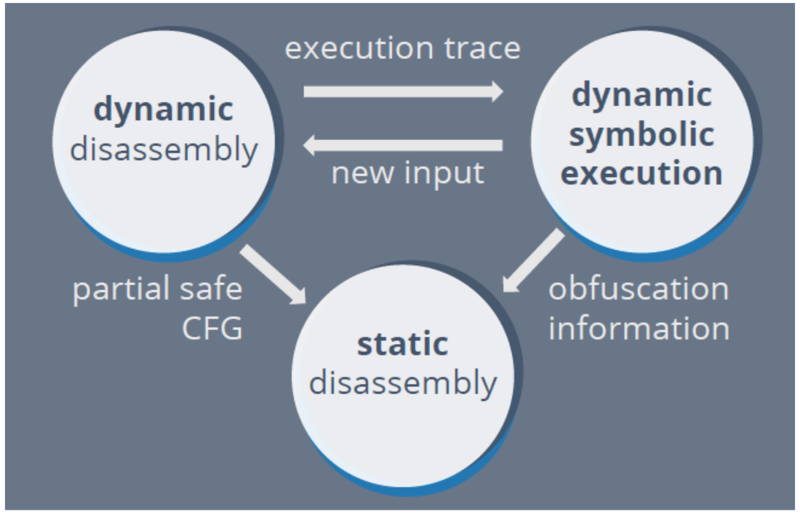
- Jump eax
- memory
- Bit resoning

OUR APPROACH: BINSEC

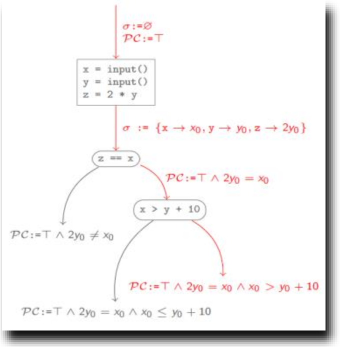


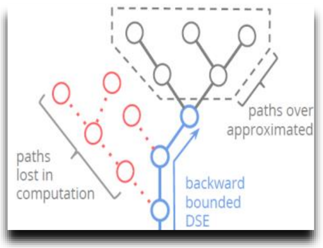






- lhs := rhs
- goto addr, goto expr
- ite(cond)? goto addr :
- assume, assert, nondet





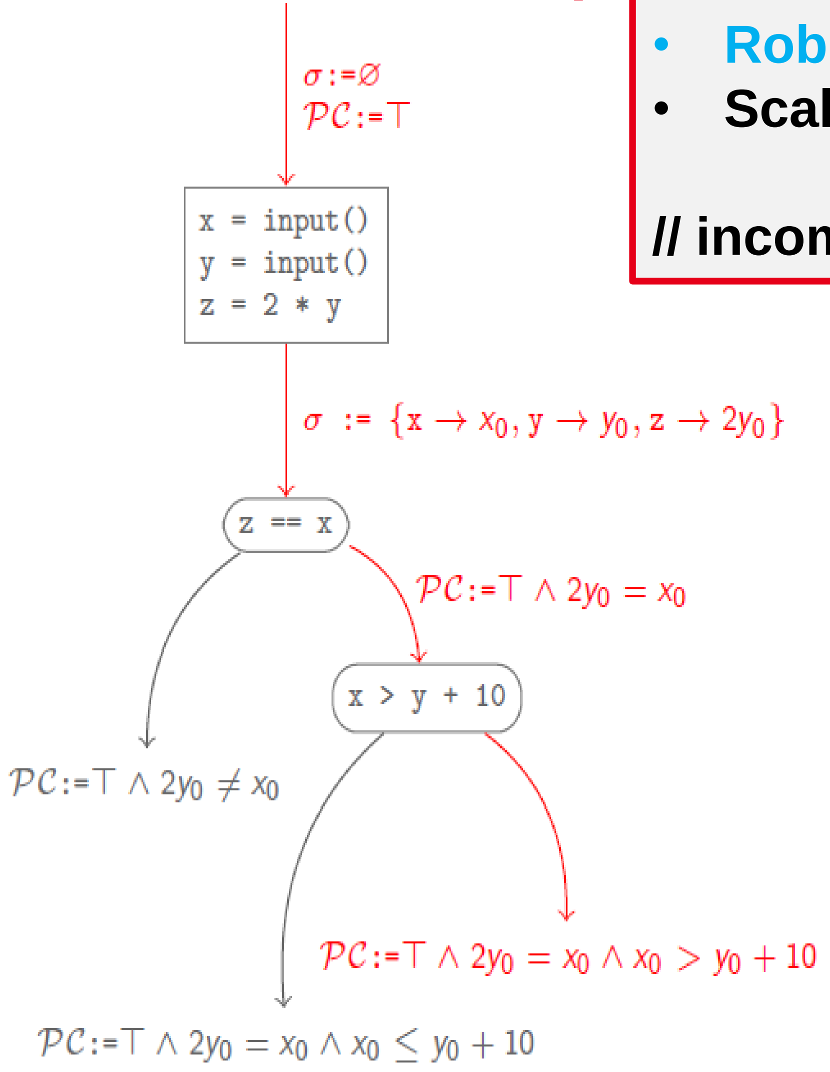
KEY: DYNAMIC SYMBOLIC EXECUTION (DSE, Godefroid 2005)

Perfect for intensive testing
- **Correct**
- **No false alarm**
- **Robust**
- **Scale in some ways**

// incomplete

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

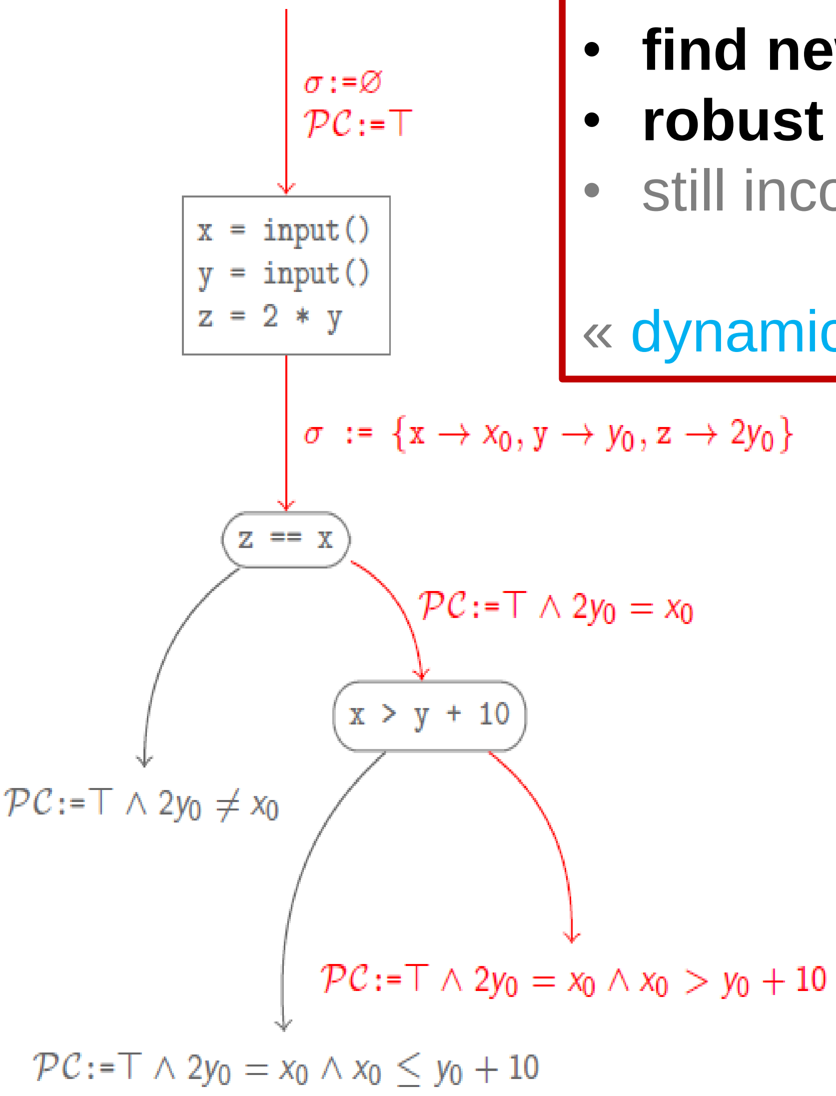
- given a path of the program
- automatically find input that follows the path
- then, iterate over all paths



DYNAMIC SYMBOLIC EXECUTION CAN HELP (Debray, Kruegel, …)

```
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

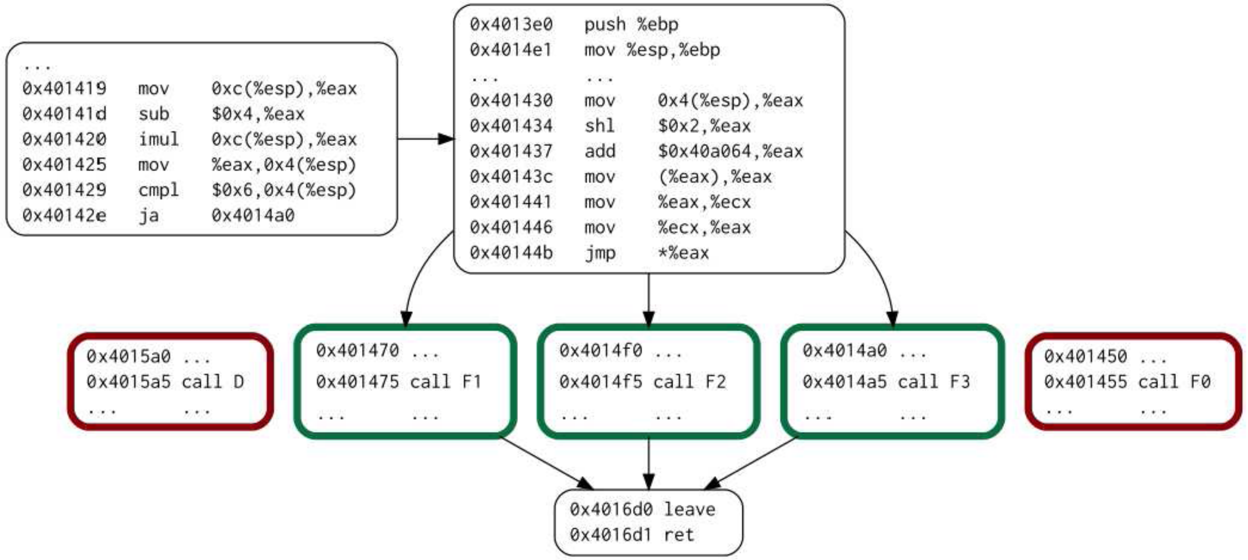


For deobfuscation

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

« dynamic analysis on steroids »

IN PRACTICE

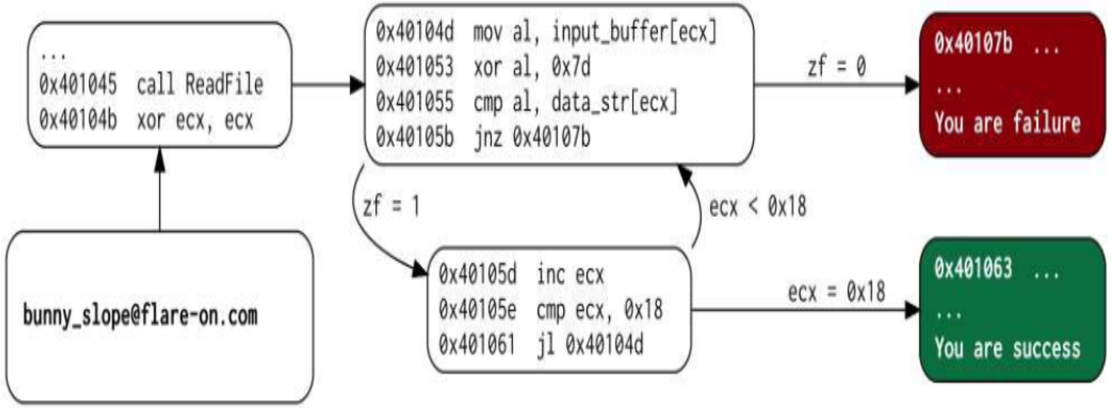


With IDA + BINSEC

```
cmp eax ebx
cmc
jae ...
```

CF := (eax$<_u$ebx)
CF := ¬CF
if (¬CF) goto ...

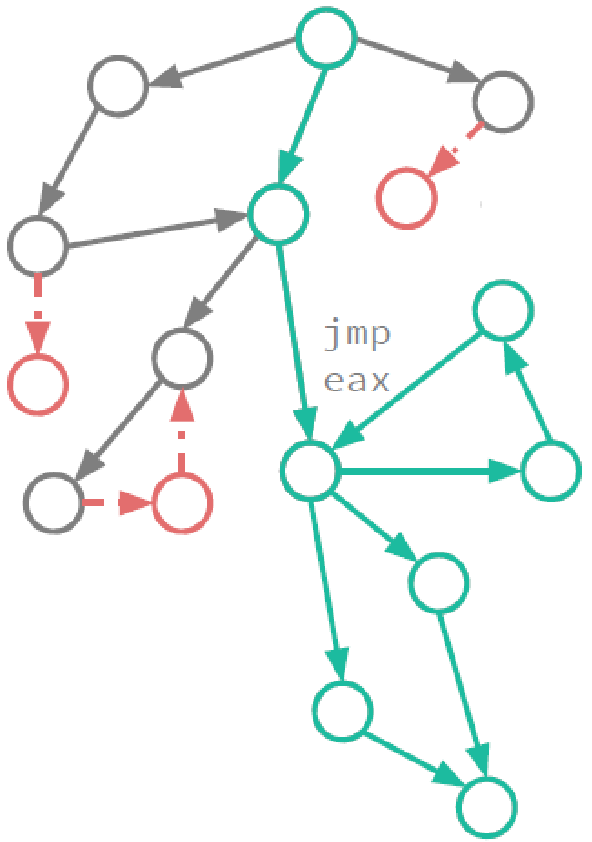
Can recover useful semantic information
- **More precise disassembly**
- **Exact semantic of instructions**
- **Input of interest**
- **…**



YET … WHAT ABOUT INFEASIBILITY QUESTIONS?

Prove that something is always true (resp. false)

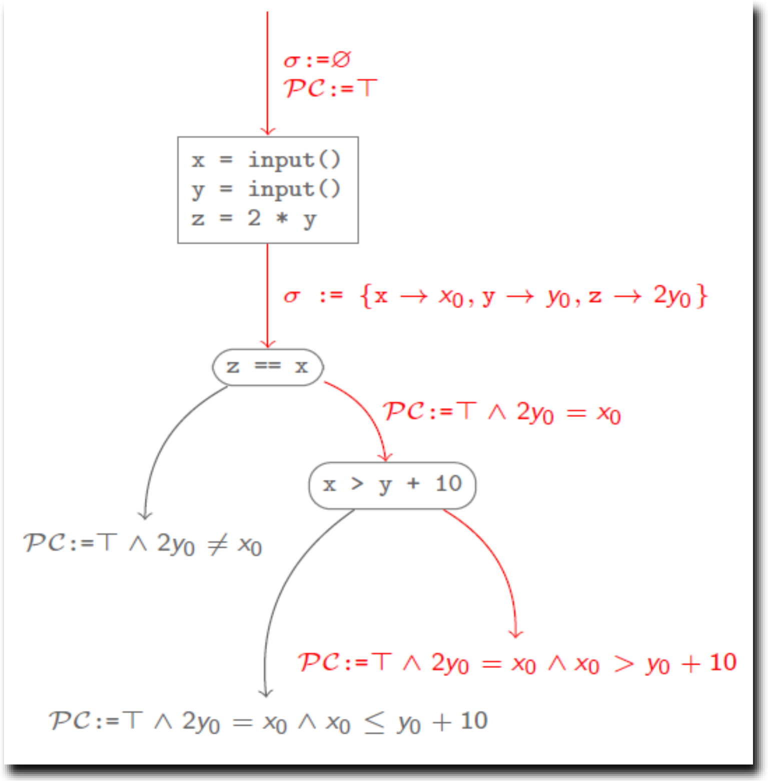
Many such issues in reverse
- **is a branch dead?**
- **does the ret always return to the call?**
- **have i found all targets of a dynamic jump?**

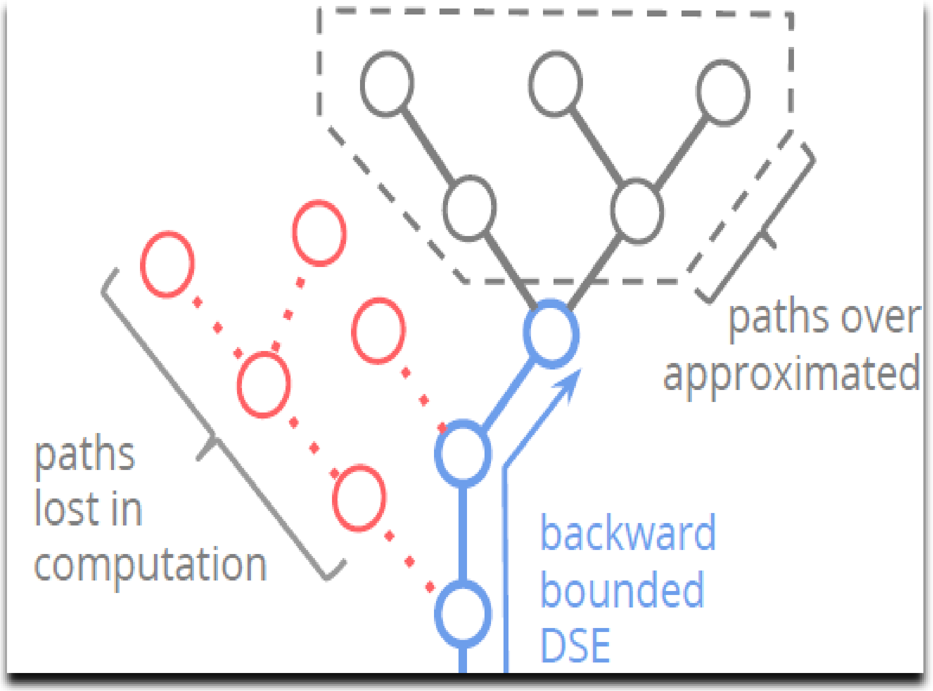


And more
- **does this malicious ret always go there?**
- **does this expression always evaluate to 15?**
- **does this self-modification always write this opcode?**
- **does this self-modification always rewrite this instr.?**
- **…**

Not addressed by DSE
- **Cannot enumerate all paths**

FORWARD & BACKWARD SYMBOLIC EXECUTION





	(forward) DSE	bb-DSE
feasibility queries	🟢	🔴
infeasibility queries	🔴	🟢
scale	🟠	🟢

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)		Genuine (5183)		TO	Error rate (FP+FN)/Tot (%)	Time (s)	avg/query (s)
	ok	miss (FN)	ok	miss (FP)				
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			runtime violation		
	`#ret` †	proved genuine	proved a/d	`#ret` †	proved a/d	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	[illegible]	[illegible]	[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: PACKERS (fun facts)

Several of the tricks detected by the analysis

OP in ACProtect		
1018f7a	js	0x1018f92
1018f7c	jns	0x1018f92

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

CST in ACProtect		
1004328	call	0x1004318
1004318	add	[esp], 9
100431c	ret	

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	0x10043bb at runtime
10043b0	jnz	0x10043ba
Enter SMC Layer 1		
10043ba	push	0x10011d7
10043bf	ret	

OP (decoy) in ASPack

```
10040fe: mov bl, 0x0
10041c0: cmp bl, 0x1
1004103: jnz 0x1004163
```

0x10040ff at runtime

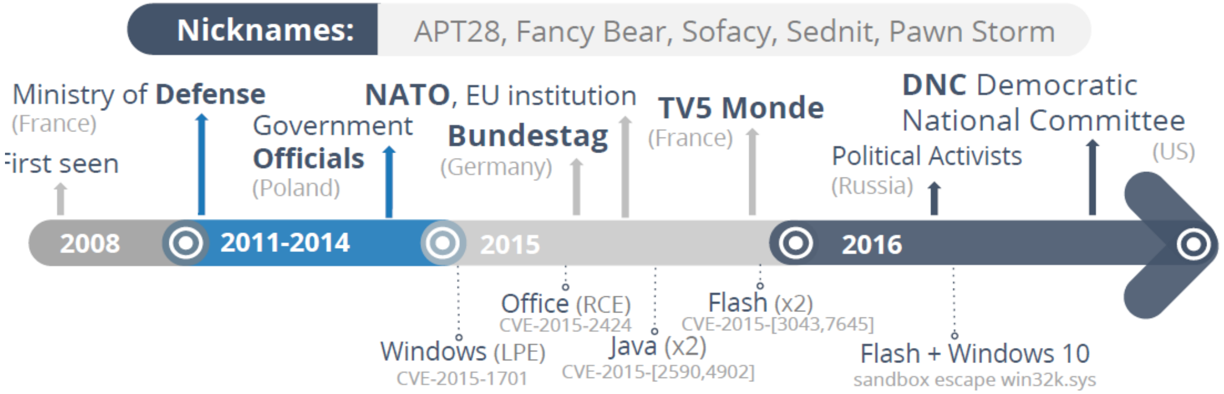
ZF = 0:

```
1004163: jmp 0x100416d
[...]
```

ZF = 1:

```
1004105: inc [ebp+0xec]
[...]
```

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

CASE-STUDY: THE XTUNNEL MALWARE (fun facts)

- **Protection seems to rely only on opaque predicates**
- **Only two families of opaque predicates** $7y^2 - 1 \neq x^2$ $\frac{2}{x^2 + 1} \neq y^2 + 3$
- **Yet, quite sophisticated**
  - original OPs
  - interleaving between payload and OP computation
  - sharing among OP computations
  - possibly long dependencies chains (avg 8.7, upto 230)

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

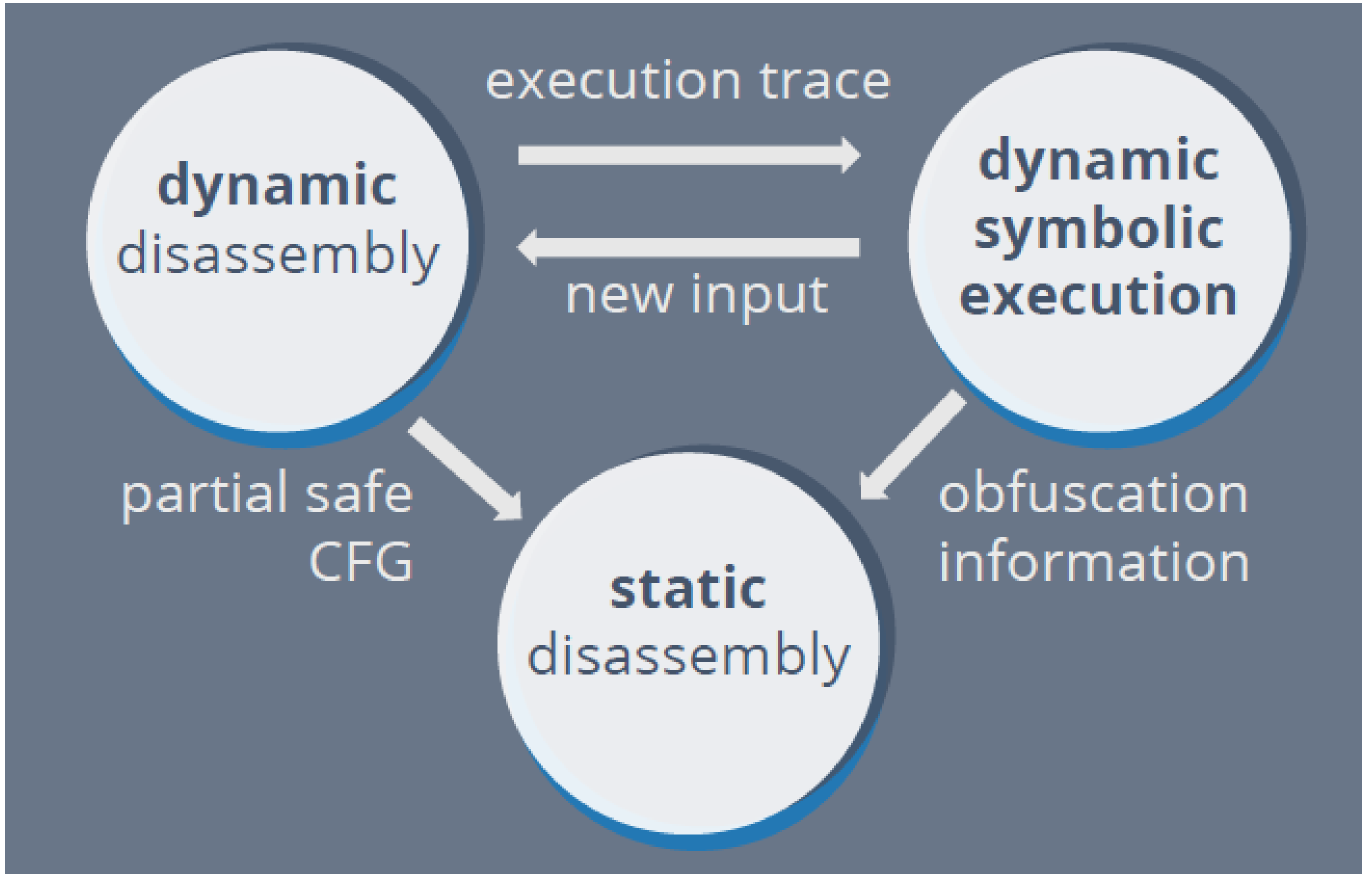
- **Anti-dynamic tricks (fool initial dynamic recovery)**
  - Can use the appropriate mitigations
  - Note: some tricks can be circumvent by symbolic reasoning

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

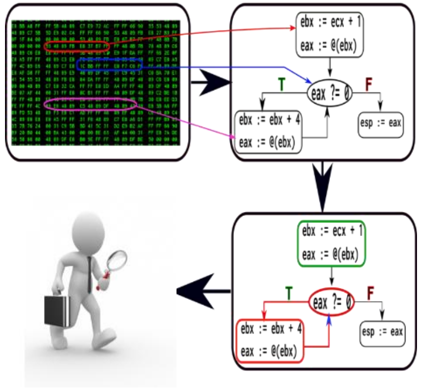
SUMMARY

	Feasibility	Infeasibility	Efficient	Robust
Static syntactic	x	--	OK	x
Dynamic	--	x	OK	OK
DSE	OK	x	x	OK
BB-DSE	x	OK	OK	OK

FUTURE DIRECTION: SPARSE DISASSEMBLY



CONCLUSION & TAKE AWAY

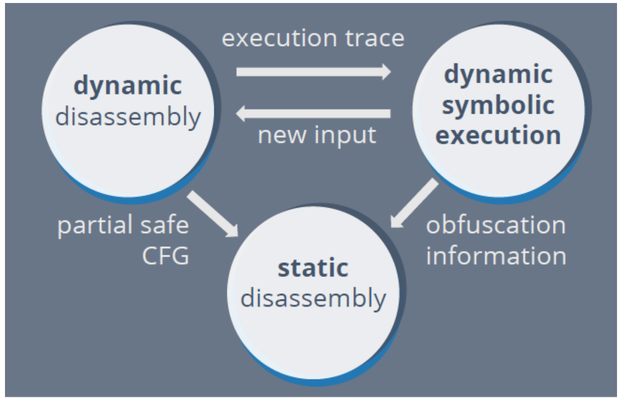


- **A tour on the advantages of symbolic methods for deobfuscation**

- **Semantic analysis complements existing approaches**
  - Explore, prove infeasible, simplify
  - Open the way to fruitful combinations

- **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

- **Next Step**
  - Combines with user and learning!
  - Anti-anti-DSE



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