Promises and Challenges of Symbolic Deobfuscation

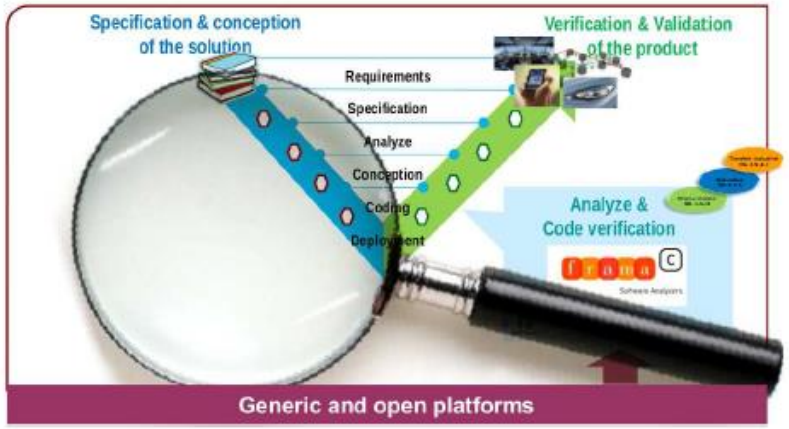
Sébastien Bardin
(CEA LIST)

Robin David, Jonathan Salwan, Adel Djoudi,
Richard Bonichon, Benjamin Farinier, Mathilde Ollivier, etc.

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

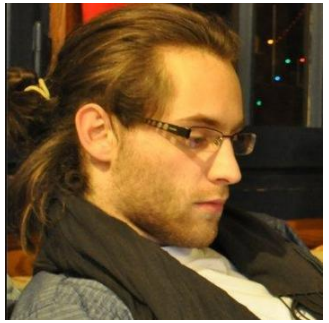
POLAR SSL

IN A NUTSHELL

- **MATE attacks and defenses is a hot topic**
  - *IP protection, malware comprehension*

- ***Symbolic deobfuscation as a game changer?***
  - *Many successful case-studies*
  - *Explore, Prove, Simplify*

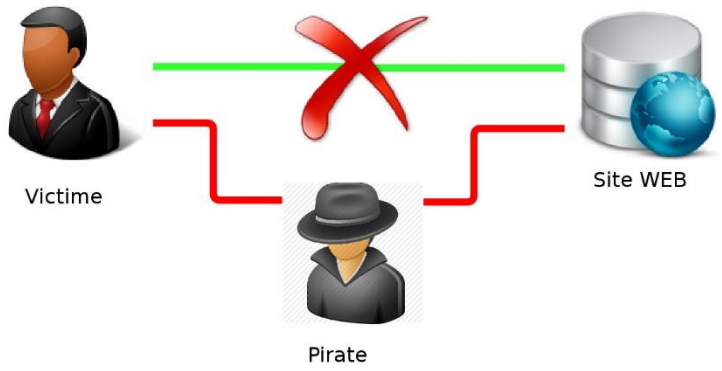
- ***This talk: a tour on symbolic deobfuscation***
  - *Present the approach, highlight successes and limits*
  - *[SANER 2016, FM 2016, BH Europe 2016, S&P 2017, DIMVA 2018]*



OUTLINE

- **Context: MATE and deobfuscation**
- **Back to the basic: binary-level semantic analysis**
- **Symbolic deobfuscation & achievements**
- **State of the defense**
- **Conclusion**

MATE is not MITM



- Steal pwd, keys, etc.

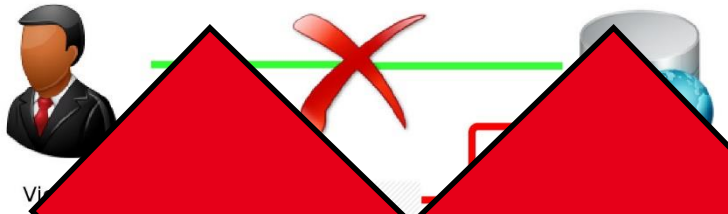
MITM: Man-In-The-Middle

Attacker is on the network

- **Observe messages**
- **Forge messages**

Known crypto solutions

MATE is not MITM



MITM … Idle

Attac… the …

- **Obse… messages**
- **Forge messages**

Known crypto solutions

MATE: Man-At-The-End

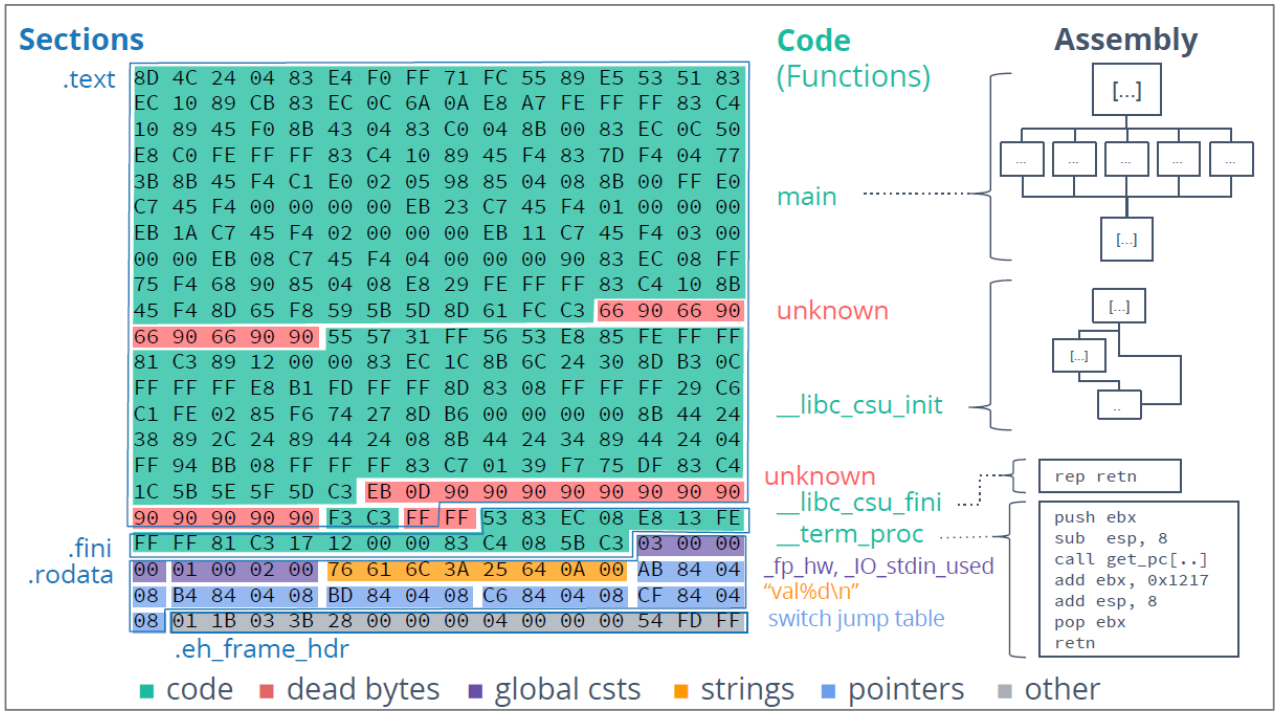
Attacker is *on the computer*

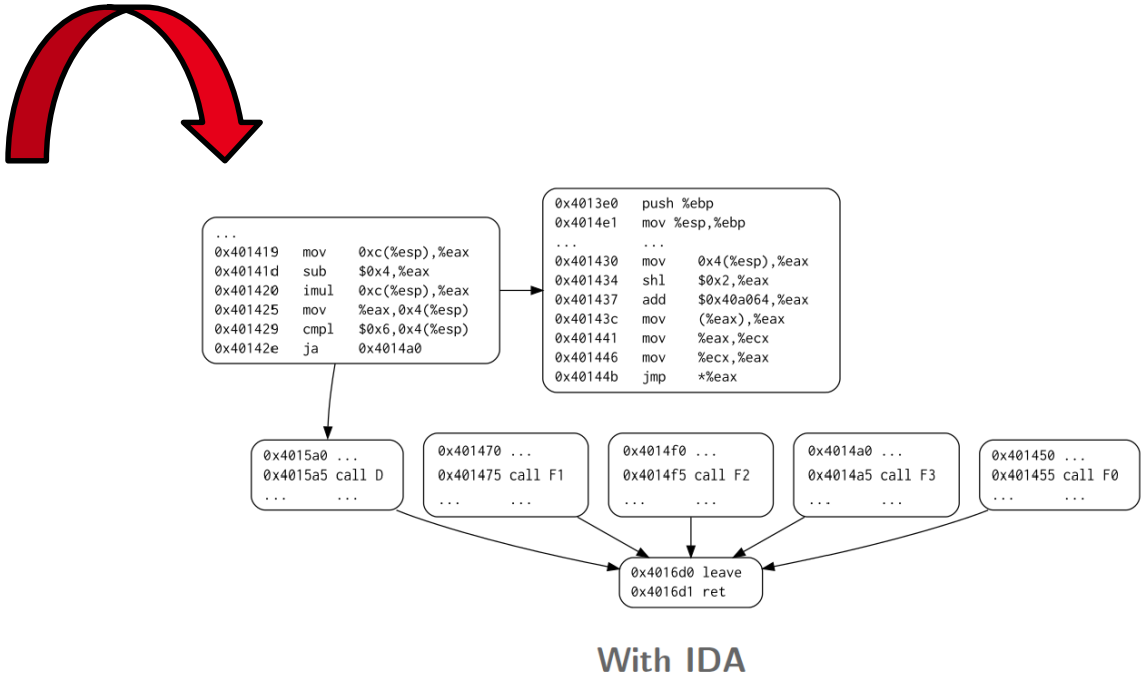
- **R/W the code**
- **Execute step by step**
- **Patch on-the-fly**

crypto or code analysis?

- Steal pwd, keys
- Tamper code
- Steal code

<*aparté*> NOT SO HARD FOR EXPERTS

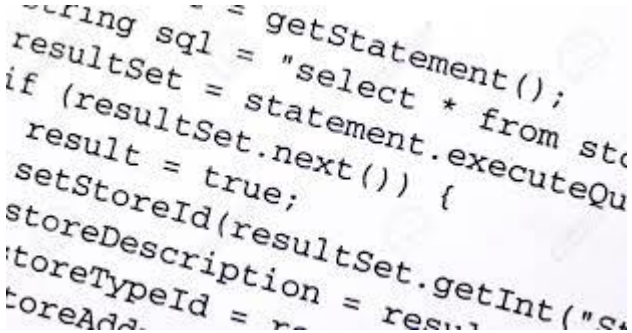


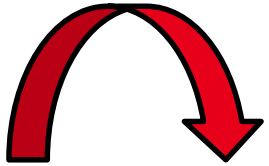


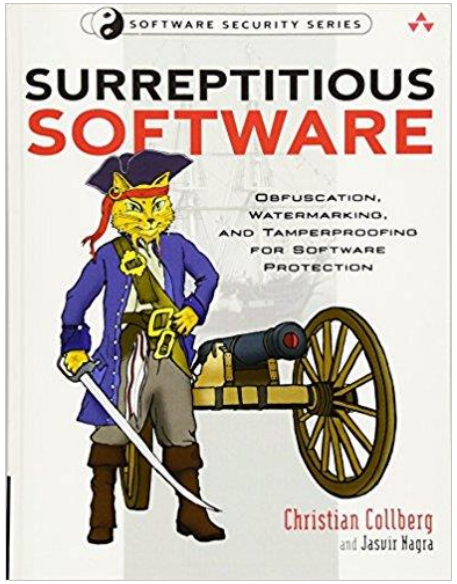
A SOLUTION: OBFUSCATION

State of the art

- **No usable math-proven solution**
- **Useful ad hoc solutions (strength?)**



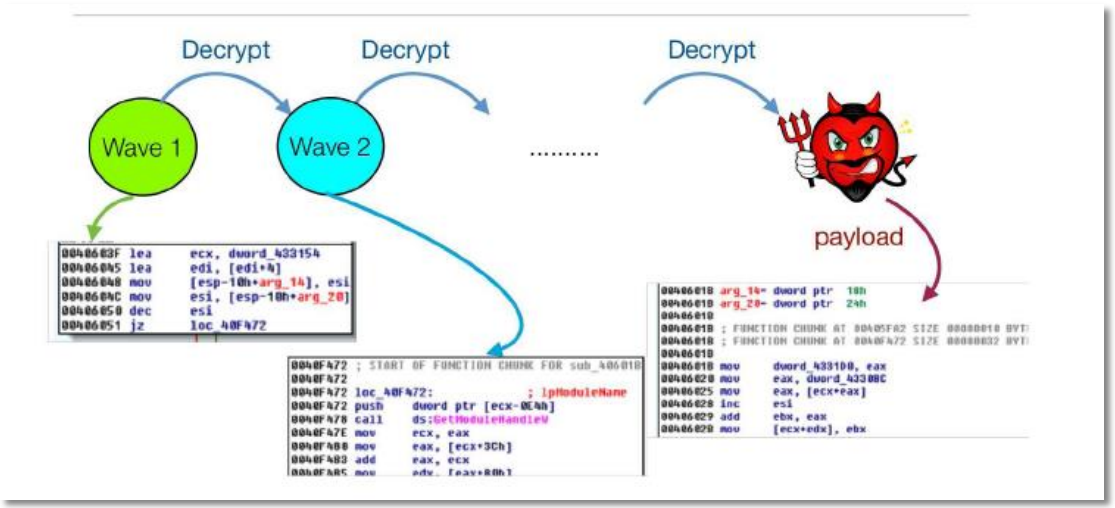




Transform P into P' such that

- **P' behaves like P**
- **P' roughly as efficient as P**
- **P' is very hard to understand**

OBFUSCATION IN PRACTICE



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

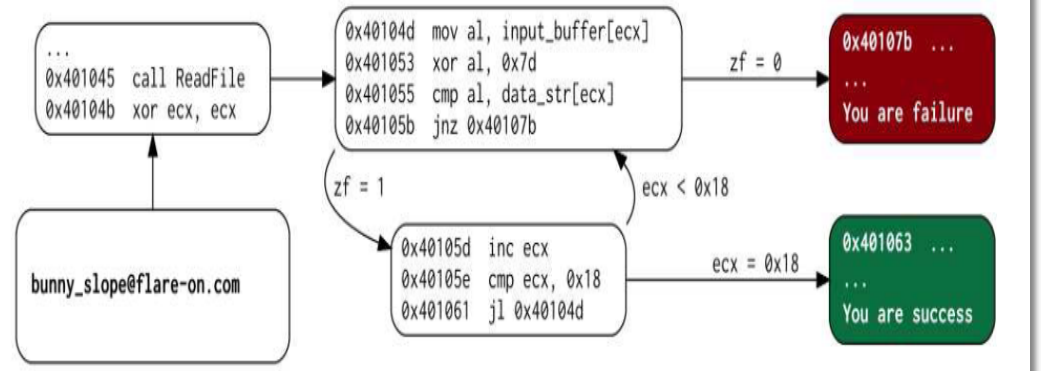
eg: $7y^2 - 1 \neq x^2$

(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>
```

- 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)

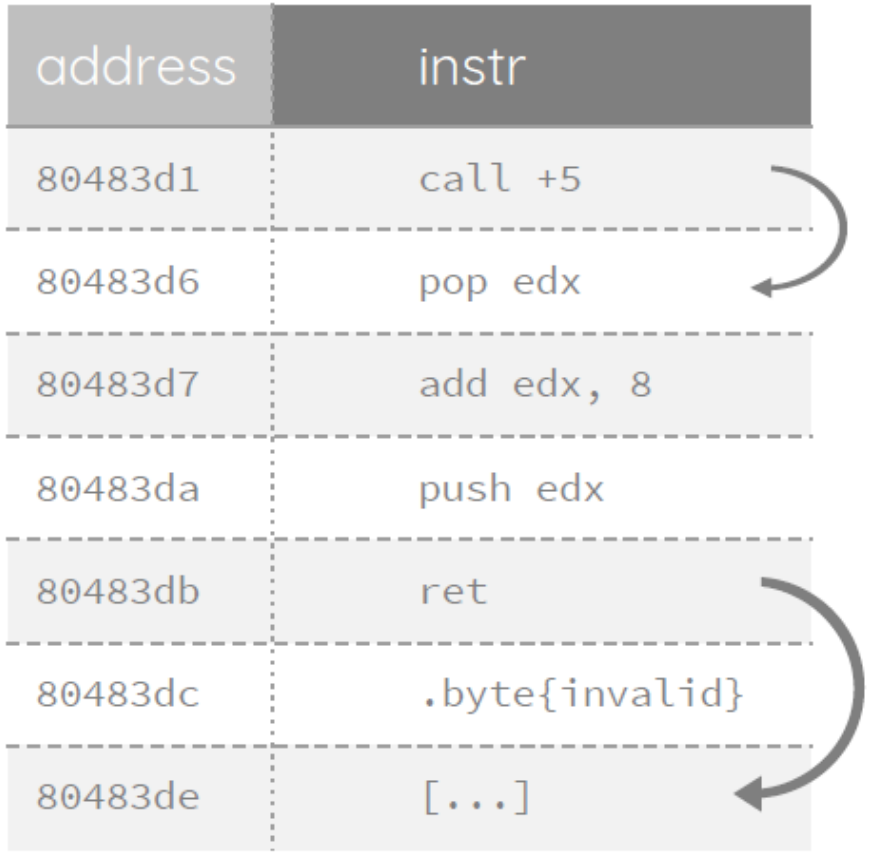
↧

```
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

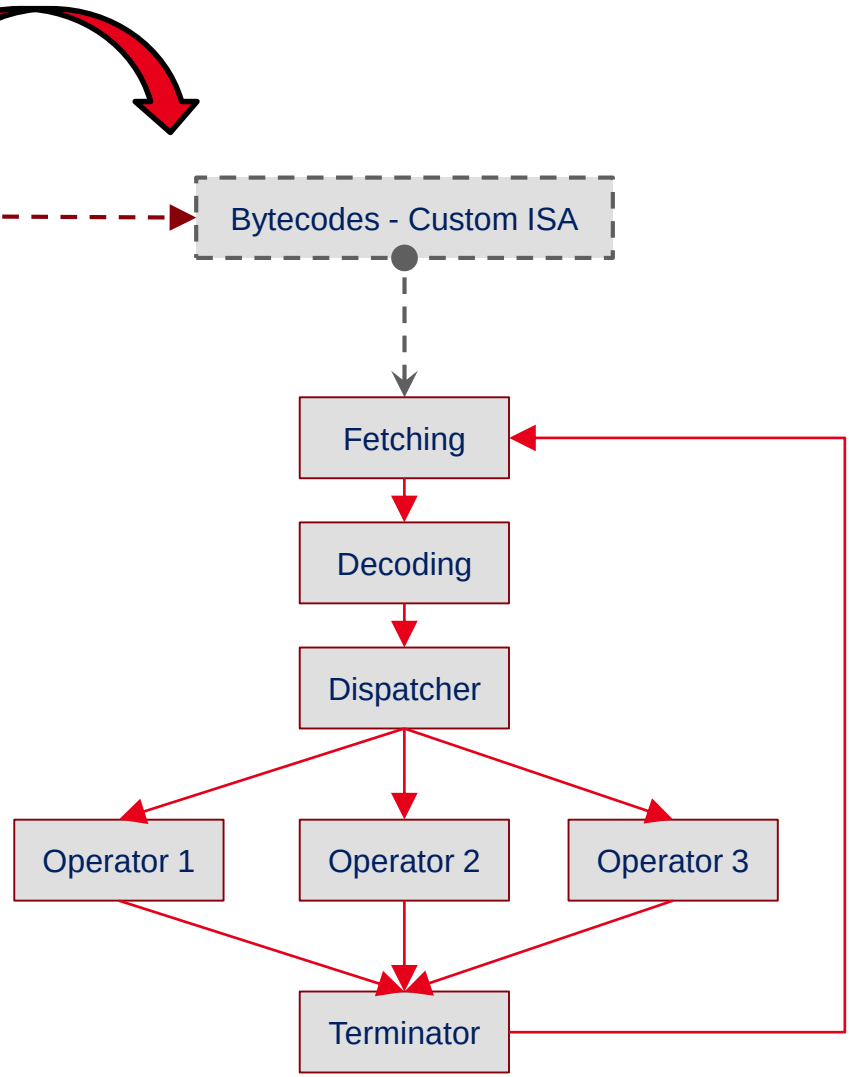
Alter the standard compilation scheme:
ret do not go back to call

- **hide** the real target
- return site is **spurious code**



EXAMPLE: VIRTUALIZATION

```
long secret(long x) {
  ......
            return x;
}
```

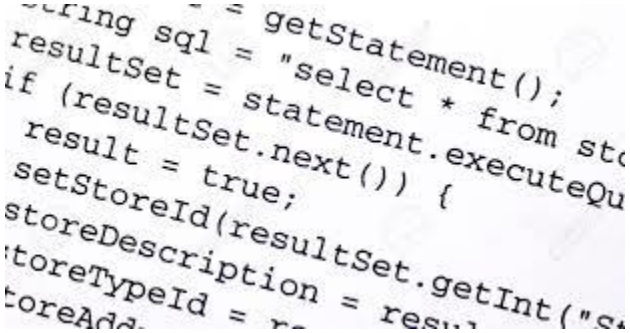


Turns code P into

- **a proprietary bytecode program + a homemade VM (runtime)**

- Easy to recover the VM structure
- But does not say anything about P

DEOBFUSCATION



- **Ideally, get P back from P’**
- **Or, get close enough**
- **Or, help understand P**

WHY WORKING ON DEOBFUSCATION?

Malware comprehension

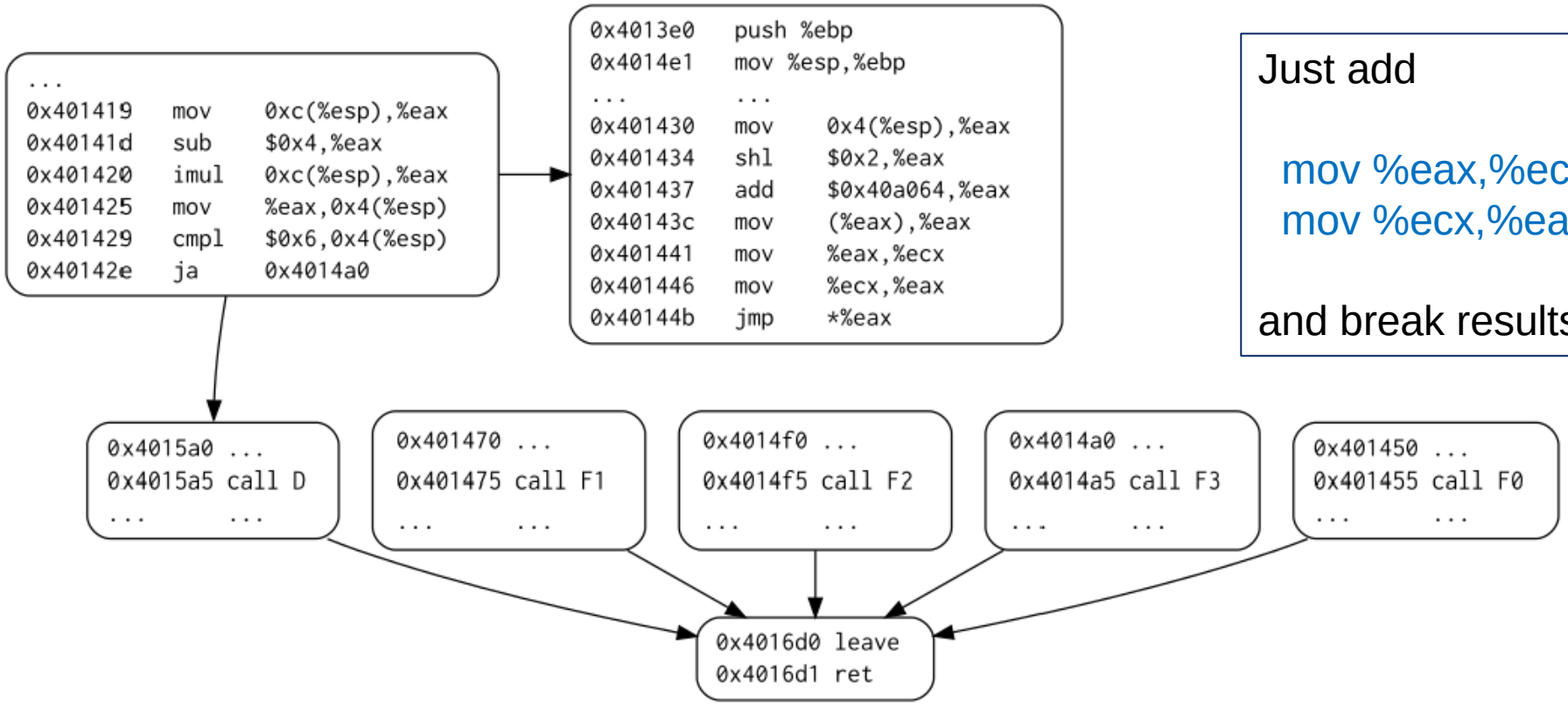
Protection evaluation

BUT … THIS IS HARD!!!

Obfuscation is automatic

➔Deobfuscation requires proper tooling

STATE-OF-THE-ART TOOLS ARE NOT ENOUGH



Just add

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

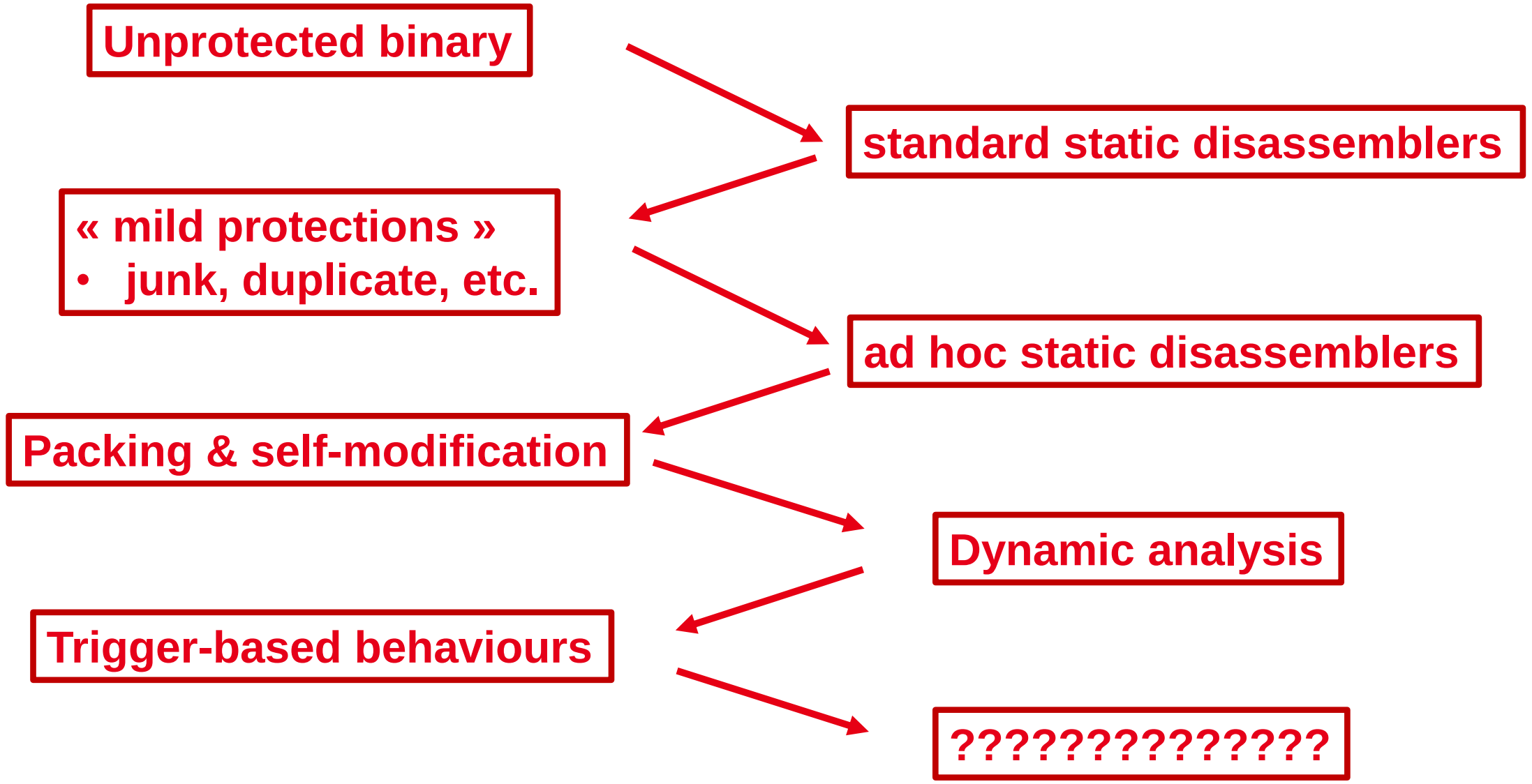
and break results

With IDA

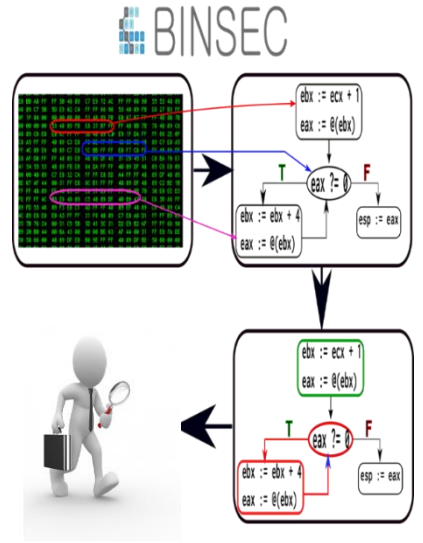
Static (syntactic)
- **too fragile (code variations)**

Dynamic
- **too incomplete (rare events)**

THE MATE ARM RACE



SOLUTION? SEMANTIC PROGRAM ANALYSIS



- ***From formal methods for safety-critical systems***
- ***Semantic = meaning of the program***

- ***Possibly well adapted***

Semantic preserved by obfuscation

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

- ***Symbolic deobfuscation***
  - *Explore, Prove & Simplify*

MAYBE A GOOD IDEA!

Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?

SEBASTIAN SCHRITTWIESER, St. Pölten University of Applied Sciences, Austria
STEFAN KATZENBEISSER, Technische Universität Darmstadt, Germany
JOHANNES KINDER, Royal Holloway, University of London, United Kingdom
GEORG MERZDOVNIK and EDGAR WEIPPL, SBA Research, Vienna, Austria

REALLY LOOKS LIKE A GREAT IDEA

A Generic Approach to Automatic Deobfuscation of Executable Code

Babak Yadegari Brian Johannesmeyer Benjamin Whitely Saumya Debray
Department of Computer Science
The University of Arizona
Tucson, AZ 85721
{babaky, bjohannesmeyer, whitely, debray}@cs.arizona.edu

Backward-Bounded DSE: Targeting Infeasibility Questions on Obfuscated Codes*

Sébastien Bardin
CEA, LIST,
91191 Gif-Sur-Yvette, France
sebastien.bardint@cea.fr

Robin David
CEA, LIST,
91191 Gif-Sur-Yvette, France
robin.david@cea.fr

Jean-Yves Marion
Université de Lorraine,
CNRS and Inria, LORIA, France
jean-yves.marion@loria.fr

Symbolic deobfuscation: from virtualized code back to the original*

Jonathan Salwan[1], Sébastien Bardin[2], and Marie-Laure Potet[3]

Code Obfuscation Against Symbolic Execution Attacks

Sebastian Banescu
Technische Universität München
banescu@in.tum.de

Christian Collberg
University of Arizona
collberg@gmail.com

Vijay Ganesh
University of Waterloo
vganesh@uwaterloo.ca

Zack Newsham
University of Waterloo
znewsham@uwaterloo.ca

Alexander Pretschner
Technische Universität München
pretschn@in.tum.de

REALLY LOOKS LIKE A GREAT IDEA

A Generic Approach to Automatic Deobfuscation of Executable Code

Babak Yadegari Brian Johannesmeyer Benjamin
Department of Computer Scien
The University of Arizona
Tucson, AZ 85721
{babaky, bjohannesmeyer, whitely, debray}@

Backward-Bounded DSE:
Targeting Infeasibility Questions
on Obfuscated Codes*

lin
, France
cea.fr

Robin David
CEA, LIST,
91191 Gif-Sur-Yvette, France
robin.david@cea.fr

Jean-Yves Marion
Université de Lorraine,
CNRS and Inria, LORIA, France
jean-yves.marion@loria.fr

QUESTIONS

- **How does it work?**
- **What does it achieve?**
- **How to counter it?**

Symbolic deobfuscation:
from virtualized code back to the original*

Jonathan Salwan[1], Sébastien Bardin[2], and Marie-Laure Potet[3]

scation Against Symbolic Execution Attacks

Sebastian Banescu
Technische Universität
München
banescu@in.tum.de

Christian Collberg
University of Arizona
collberg@gmail.com

Vijay Ganesh
University of Waterloo
vganesh@uwaterloo.ca

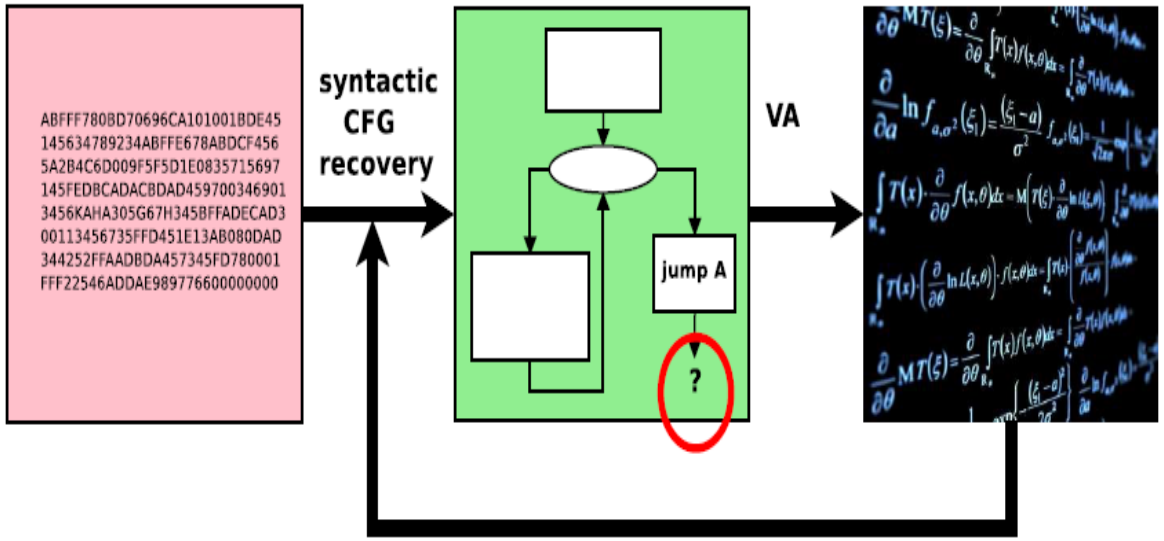
Zack Newsham
University of Waterloo
znewsham@uwaterloo.ca

Alexander Pretschner
Technische Universität
München
pretschn@in.tum.de

OUTLINE

- **Context: MATE and deobfuscation**
- **Back to the basic: binary-level semantic analysis**
- **Symbolic deobfuscation & achievements**
- **State of the defense**
- **Conclusion**

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



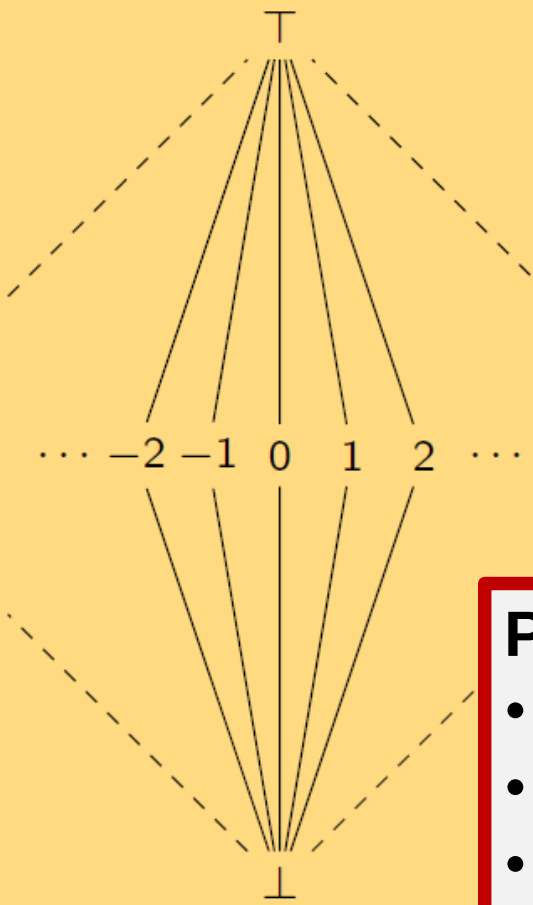
Framework : abstract interpretation

- notion of abstract domain

  $\bot, \top, \sqcup, \sqcap, \sqsubseteq$, eval$^{\#}$
- more or less precise domains

  . intervals, polyhedra, etc.
- fixpoint until stabilization

Generalize constant propagation



Problems

- Jump eax
- memory
- Bit resoning

WANTED

Robustness

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

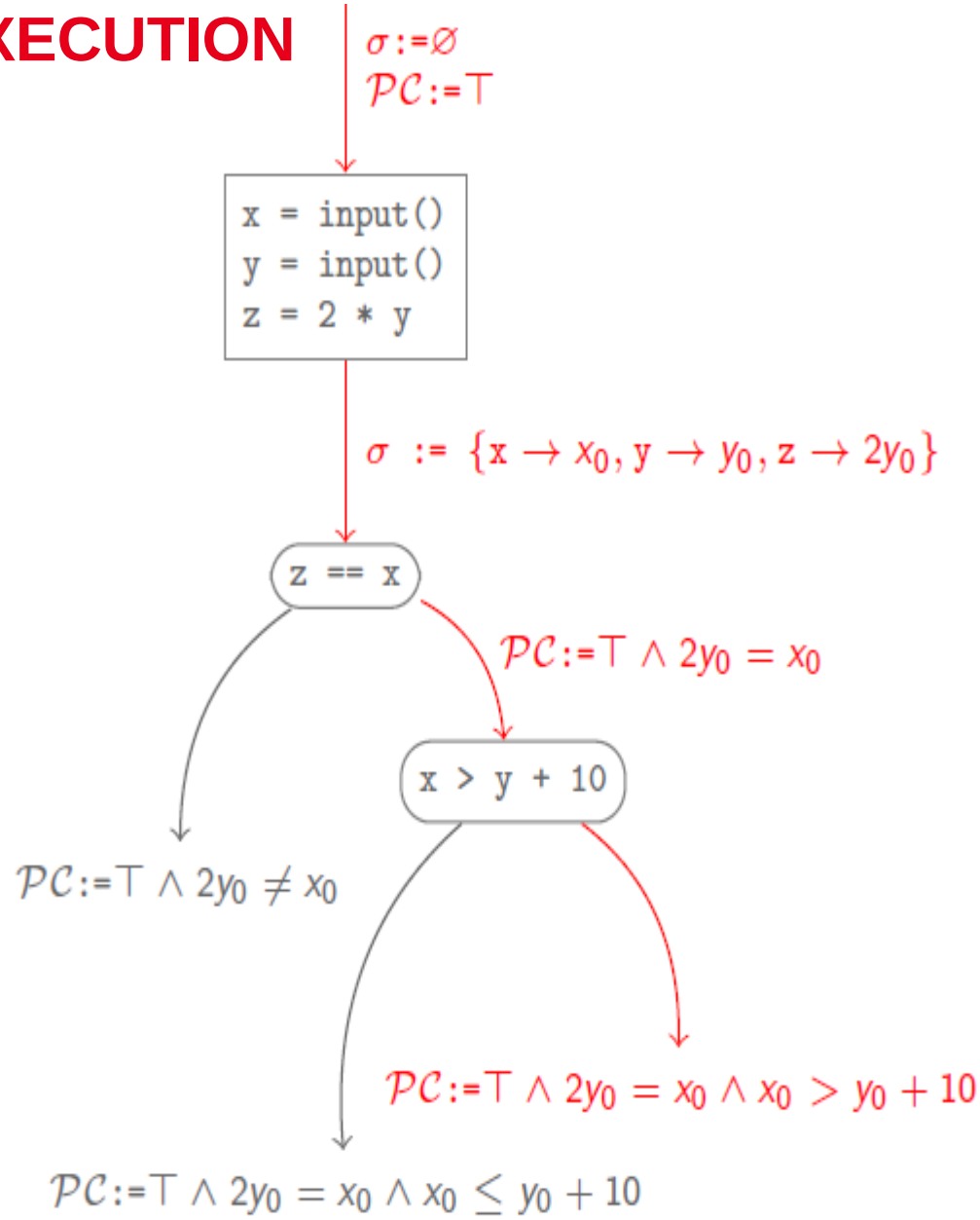
Precision

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

Scale

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;
}
```



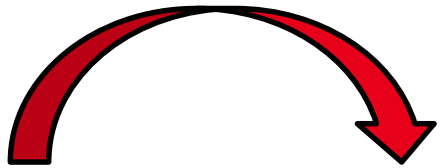
Given a path of a program

- Compute its « path predicate » f
- Solution of f ⇔ input following the path
- Solve it with powerful existing solvers

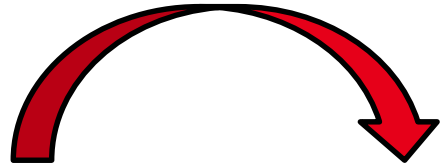
PATH PREDICATE COMPUTATION & RESOLUTION

Loc	Instruction
0	input(y,z)
1	w := y+1
2	x := w + 3
3	if (x < 2 * z) (branche True)
4	if (x < z) (branche False)

$$\text{let } W_1 \triangleq Y_0 + 1 \text{ in}$$
$$\text{let } X_2 \triangleq W_1 + 3 \text{ in}$$
$$X_2 < 2 \times Z_0 \wedge X_2 \geq Z_0$$



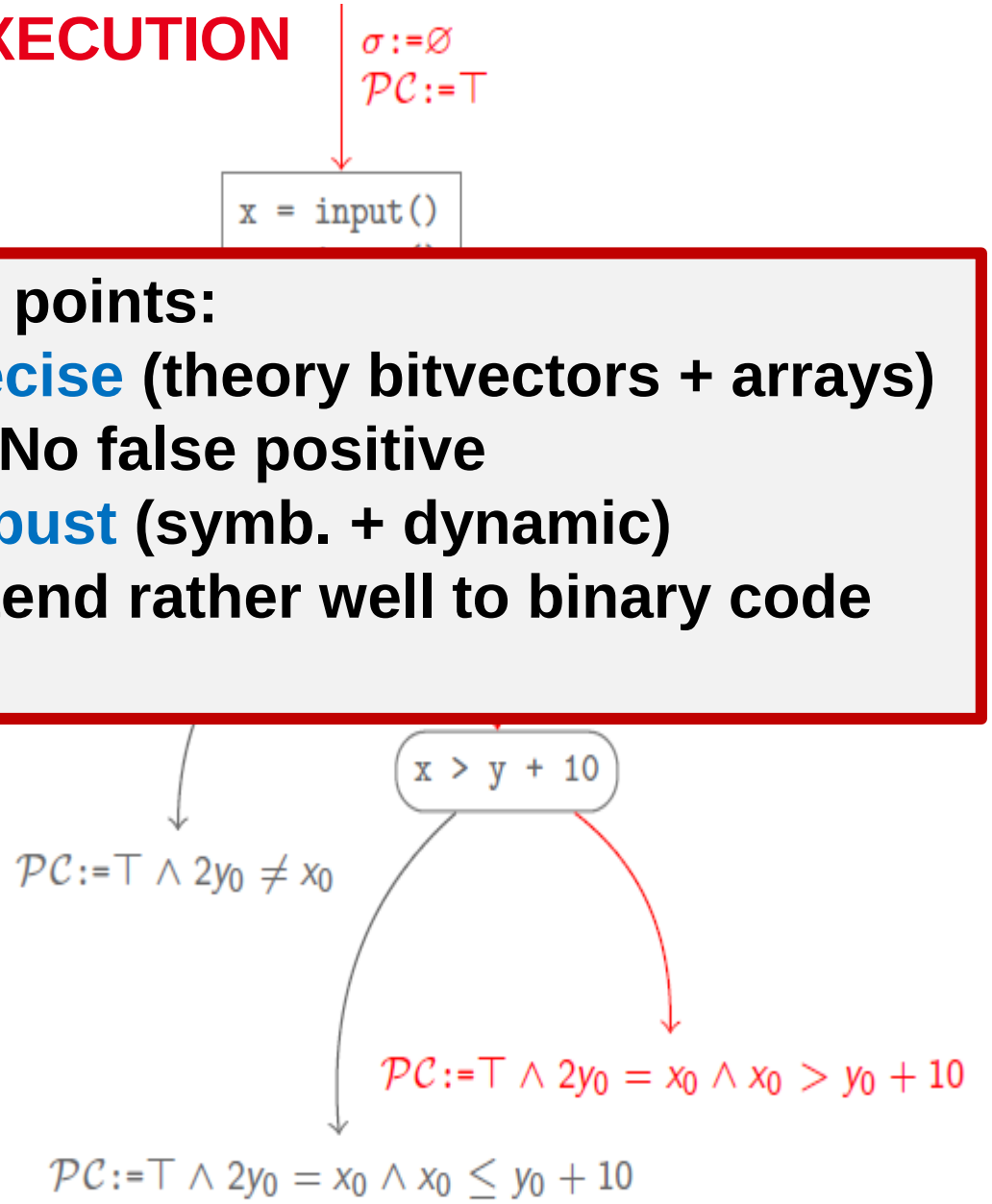
SMT Solver



Y0 = 0 /\ Z0=3

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;
}
```



Good points:

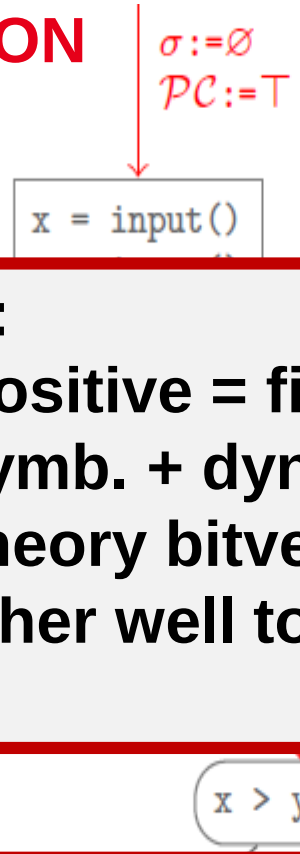
- **Precise** (theory bitvectors + arrays)
  - No false positive
- **Robust** (symb. + dynamic)
- Extend rather well to binary code

Given a path of a program

- Compute its « path predicate » f
- Solution of f ⇔ 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;
}
```



Good points:

- **No false positive = find real paths**
- **Robust (symb. + dynamic)**
- **Precise (theory bitvectors + arrays)**
- **Extend rather well to binary code**

Given a path of a program

- Compute its « path predicate » f
- Solution of f ⇔ input following the path
- Solve it with powerful existing solvers

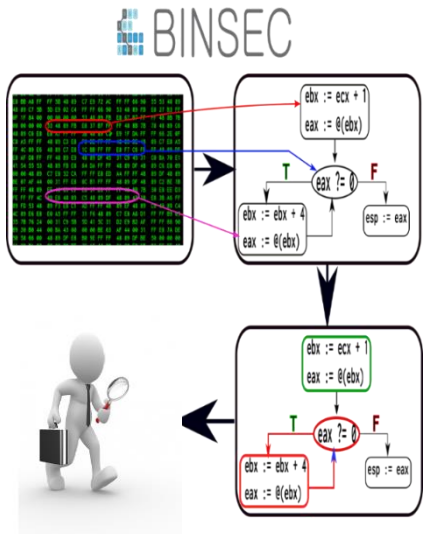
« concretization »

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

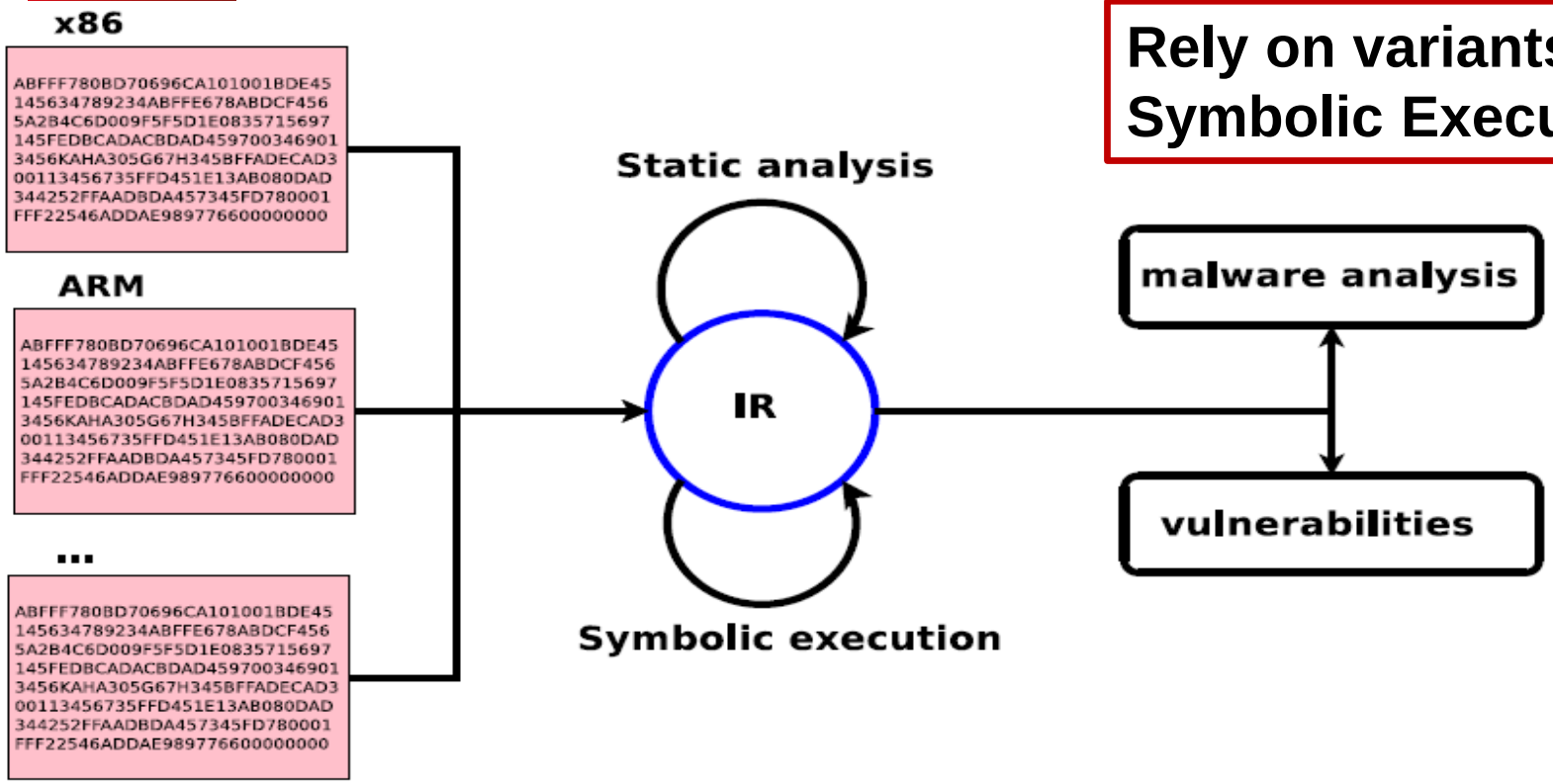
OUTLINE

- **Context: MATE and deobfuscation**
- **Back to the basic: binary-level semantic analysis**
- **Symbolic deobfuscation & achievements**
- **State of the defense**
- **Conclusion**

BINSEC: SYMBOLIC ANALYSIS for BINARY

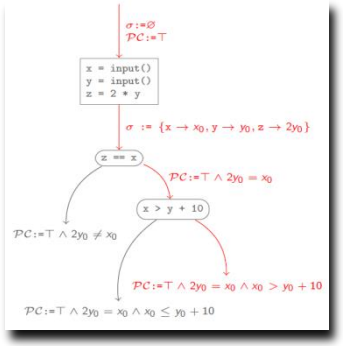


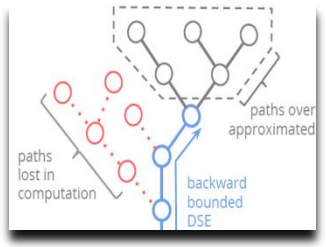
Rely on variants of Symbolic Execution

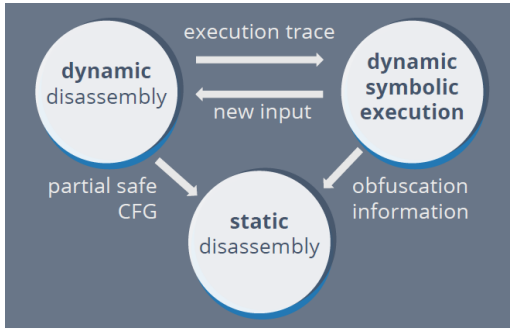


- Explore
- Prove
- Simplify

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







PART I: EXPLORE (standard SE) [SANER'16] [Yadegari et al, SP'15] (with Robin David)

Forward reasoning

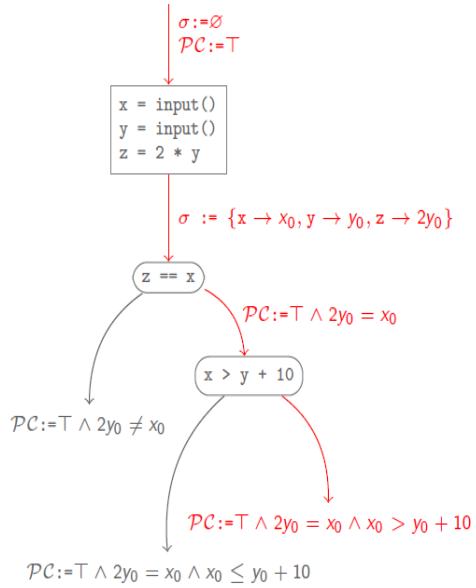
- Follows path
- Find new branch / jumps
- Standard DSE setting

Advantages

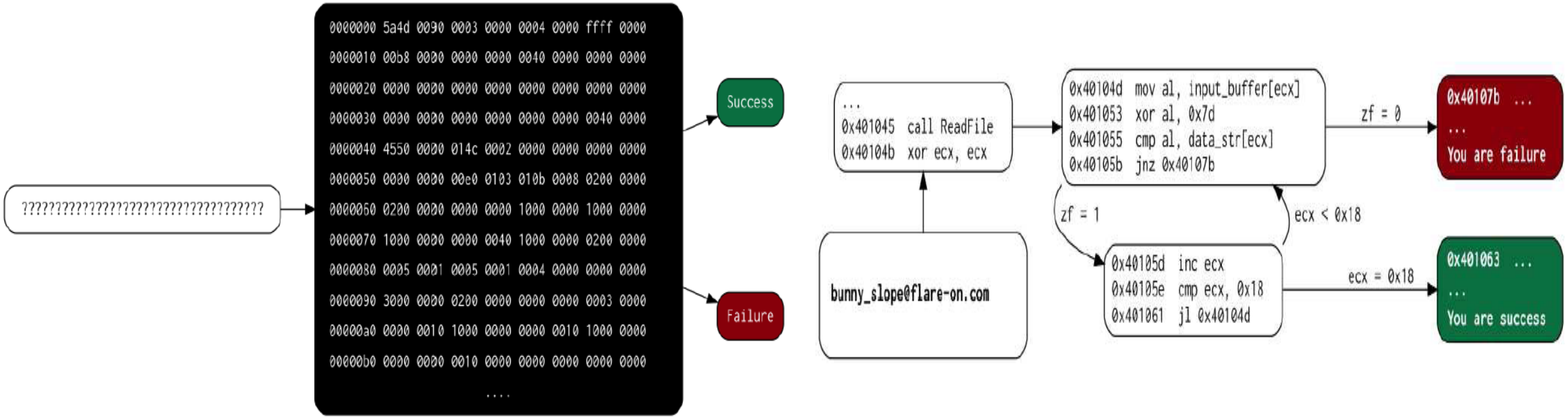
- Find new real paths
- Even rare paths

« dynamic analysis on steroids »

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



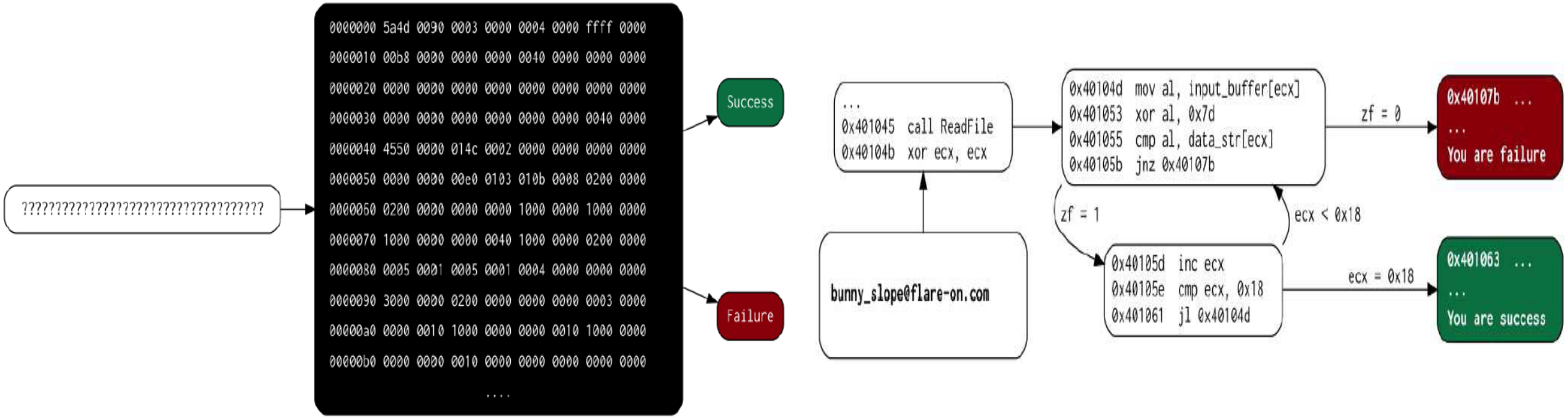
EXAMPLE: FIND THE GOOD PATH



Crackme challenges

- input == secret ↦ success
- input ≠ secret ↦ failure

EXAMPLE: FIND THE GOOD PATH

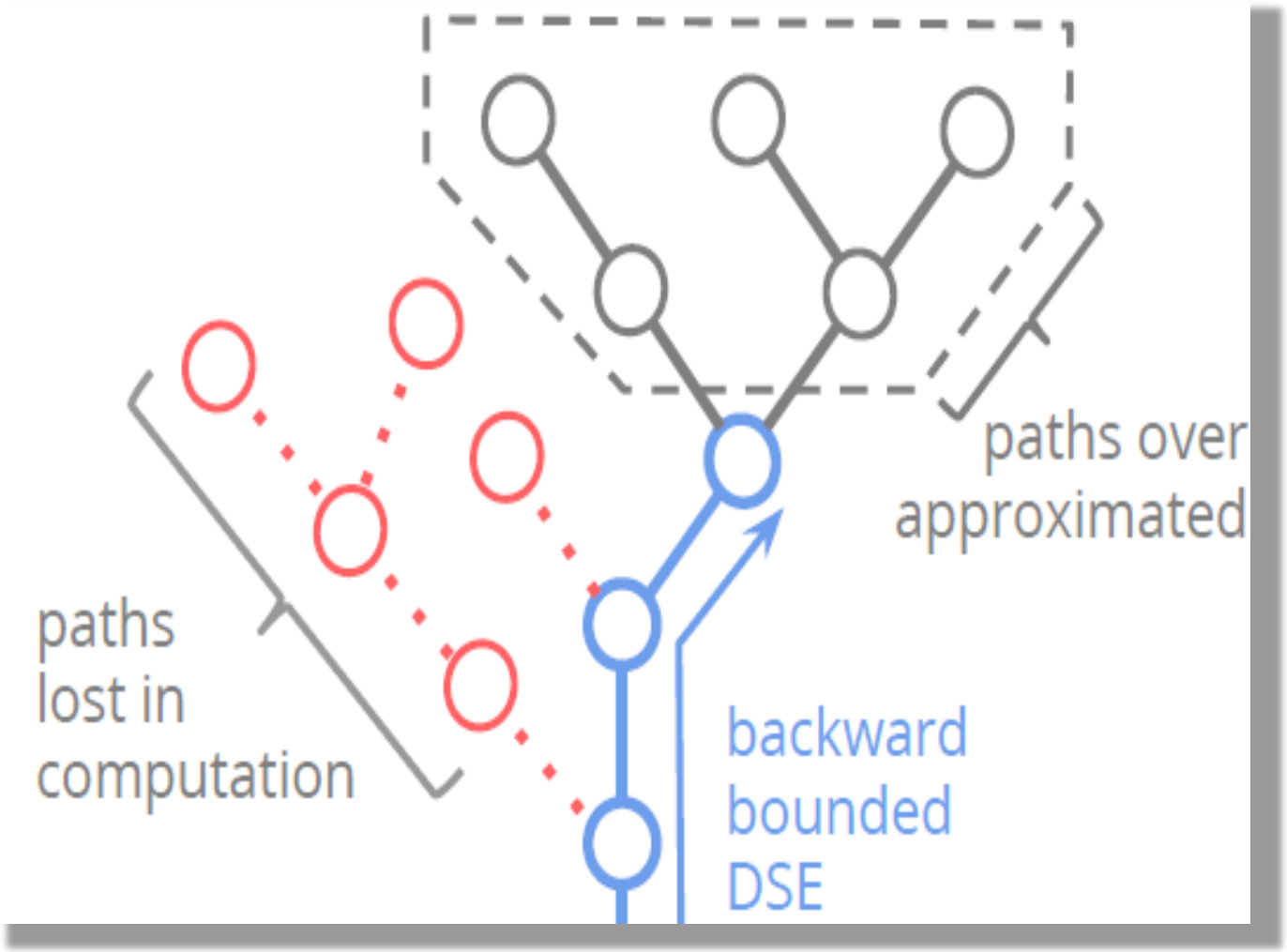


Beware: scale?

Crackme challenges

- input == secret ↦ success
- input ≠ secret ↦ failure

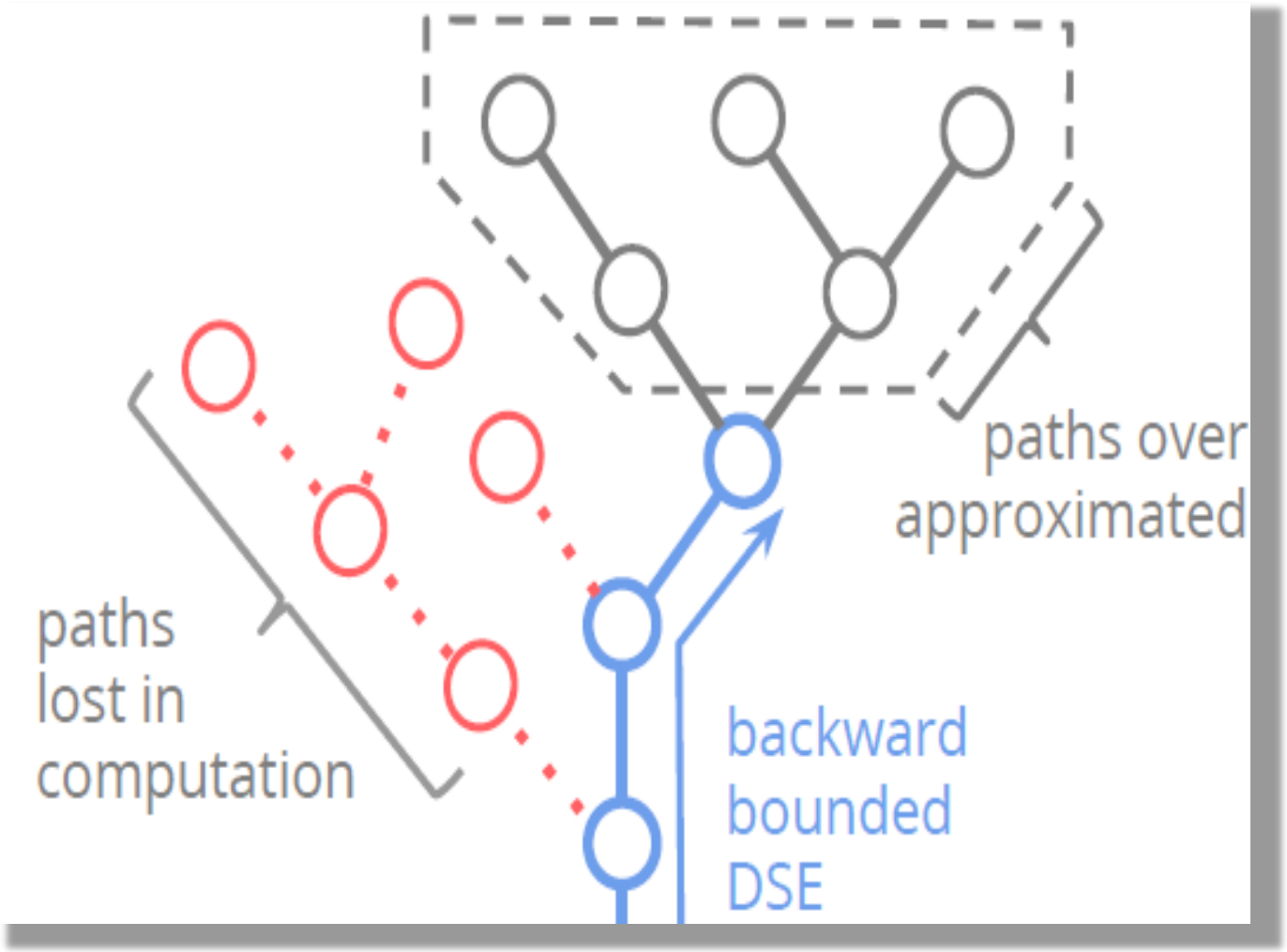
PART II: PROVE



Backward bounded SE

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

PART II: PROVE



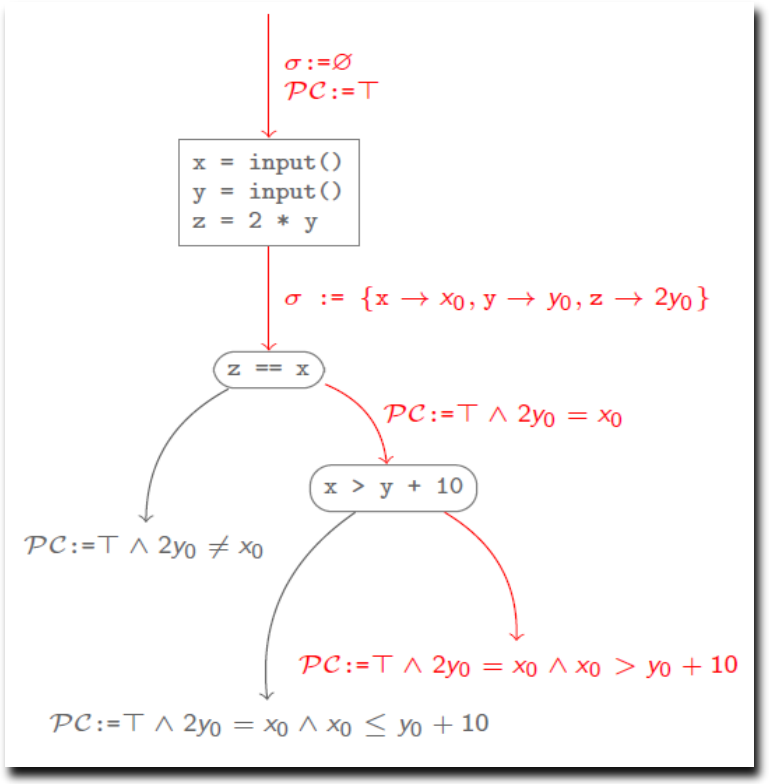
Backward bounded SE

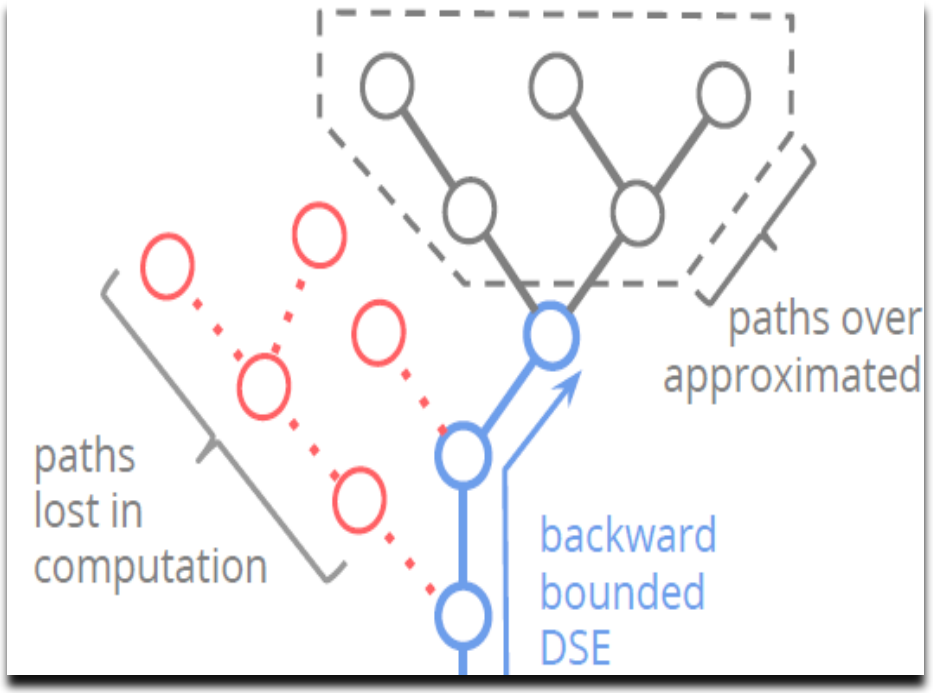
- Compute k-predecessors
- If the set is empty, no pred.
- Allows to prove things

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

  (low rate, controlled XPs)

BACKWARD SYMBOLIC EXECUTION





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

Explore & discover

- **Prove infeasible**

IN PRACTICE

- **Scalable switch target recovery**
- **Opaque predicate detection**
- **Call stack tampering**
- **High-level condition recovery**

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

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

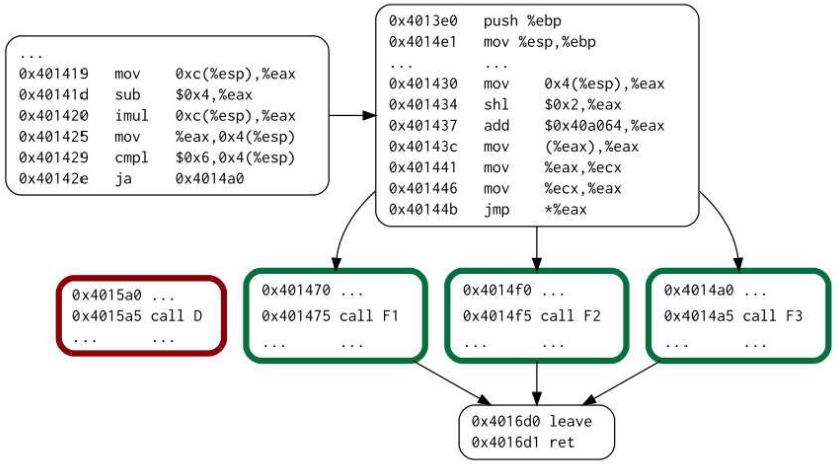
↧

```
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	[...]

```
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 l1
        X := 1
        goto l2
l1:     X := -1
l2:
```



With IDA + BINSEC

CASE-STUDY: PACKERS [BH EU'16] (with Robin David)

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				24	125	78
Expressor	635K	1	1				14	0
FSG v2.0	68k	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				14	0
TELock v0.51	406K	1	1				3	1
Upack v0.39	711K	1	1				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..)

OP in Armadillo

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

CST in ACProtect

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

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
```

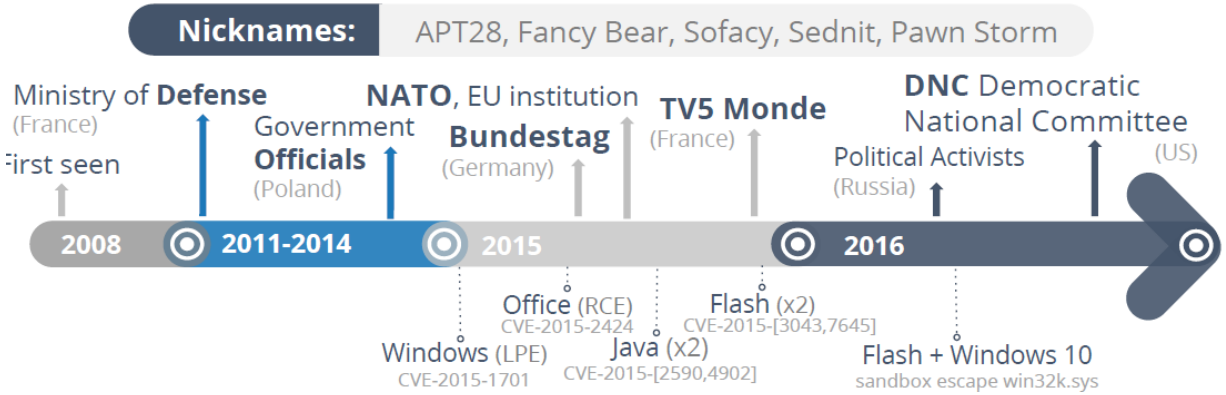
0x10040ff at runtime

ZF = 0 | ZF = 1

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

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

CASE-STUDY: THE XTUNNEL MALWARE [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	**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)

PART III: SIMPLIFY

Why? recover hidden simple expressions

- **Junk code, junk computations**
- **Opaque values**
- **Duplicate code**
- **Complex patterns (MBAs)**

Basic

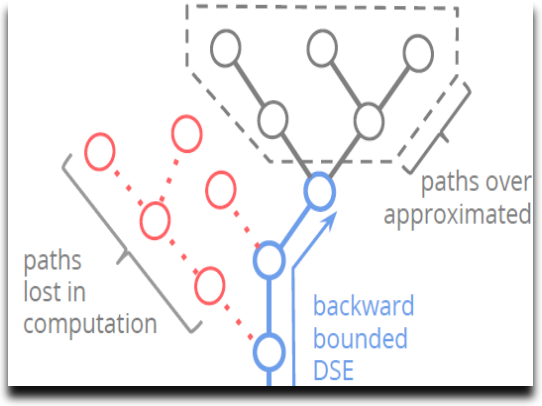
- **Tainting**
- **Slicing**

Symbolic reasoning a priori well adapted

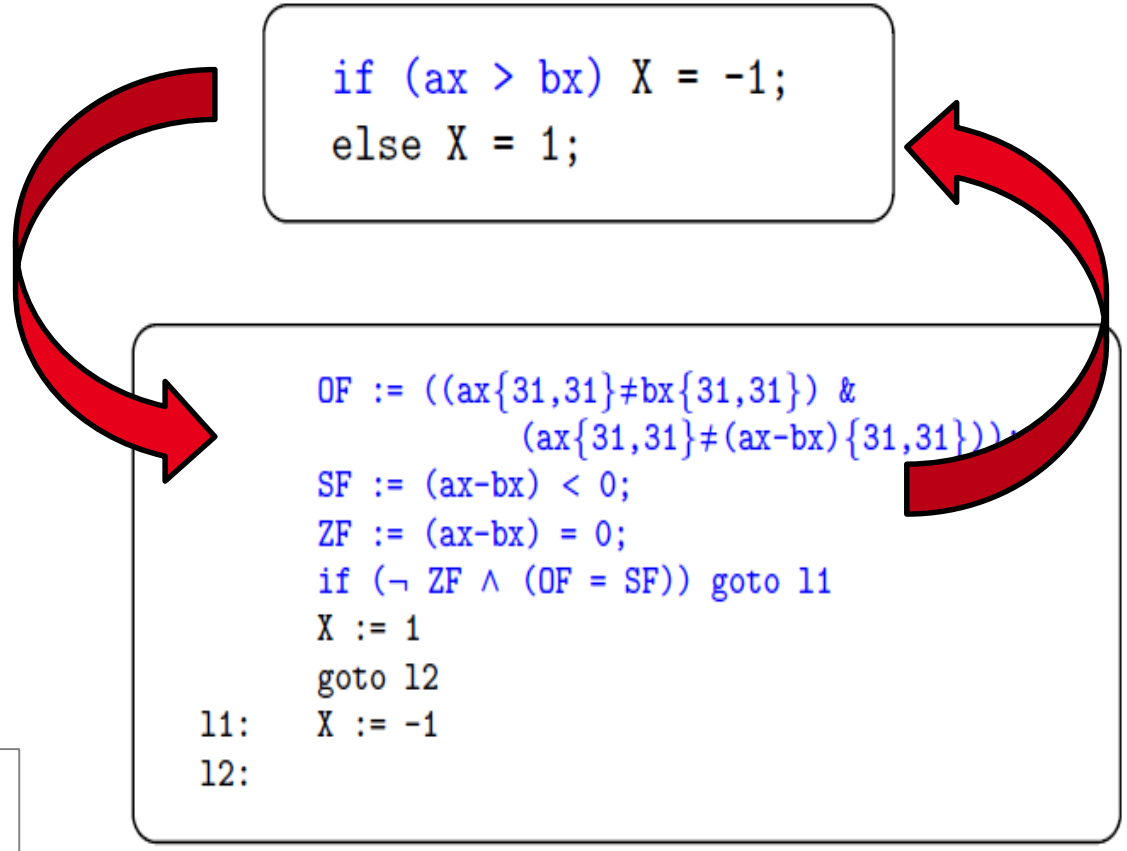
- **Normalization / rewrite rules: (a+b-a) → b**
- **Solver-based proof: check-valid(a+b-a = b)**

EXAMPLE: HIGH-LEVEL CONDITION RECOVERY [FM'16] (with Adel Djoudi)

	flag predicate	cmp x y predicate	sub x y predicate [2]	test x y predicate
ja, jnbe	$\neg CF \wedge \neg ZF$	$x >_u y$	$x' \neq 0$	$x\&y \neq 0$
jae, jnb, jnc	$\neg CF$	$x \geq_u y$	true	true
jb, jnae, jc	CF	$x <_u y$	$x' \neq 0$	false
jbe, jna	$CF \vee ZF$	$x \leq_u y$	true	$x\&y = 0$
je, jz	ZF	$x = y$	$x' = 0$	$x\&y = 0$
jne, jnz	$\neg ZF$	$x \neq y$	$x' \neq 0$	$x\&y \neq 0$
jg, jnle	$\neg ZF \wedge (OF = SF)$	$x > y$	$x' > 0$	$(x\&y \neq 0) \wedge (x \geq 0 \vee y \geq 0)$
jge, jnl	$(OF = SF)$	$x \geq y$	true	$(x \geq 0 \vee y \geq 0)$
jl, jnge	$(OF \neq SF)$	$x < y$	$x' < 0$	$(x < 0 \wedge y < 0)$
jle, jng	$ZF \vee (OF \neq SF)$	$x \leq y$	true	$(x\&y = 0) \vee (x < 0 \wedge y < 0)$



+ simplification inference
+ SMT solver



CASE-STUDY: DEVIRTUALIZATION [DIMVA'18] (with Jonathan Salwan)

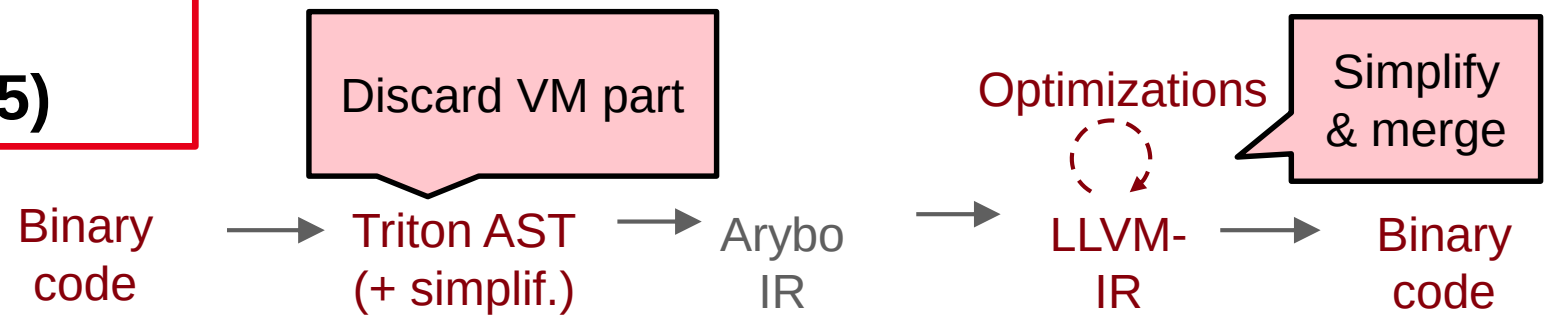
TIGRESS Challenge

- Original codes: hash-like functions
- Focus on challenges 0-4
- Only challenge 1 was solved

Challenge	Description	Number of binaries	Difficulty (1-10)	Script	Prize	Status
0000	One level of virtualization, random dispatch.	5	1	script	Certificate issued by DAPA	Solved
0001	One level of virtualization, superoperators, split instruction handlers.	5	2	script	Signed copy of Surreptitious Software.	Open
0002	One level of virtualization, bogus functions, implicit flow.	5	3	script	Signed copy of Surreptitious Software.	Open
0003	One level of virtualization, instruction handlers obfuscated with arithmetic encoding, virtualized function is split and the split parts merged.	5	2	script	Signed copy of Surreptitious Software.	Open
0004	Two levels of virtualization, implicit flow.	5	4	script	USD 100.00	Open
0005	One level of virtualization, one level of jitting, implicit flow.	5	4	script	USD 100.00	Open
0006	Two levels of jitting, implicit flow.	5	4	script	USD 100.00	Open

Solve challenges 0 - 4 (25 samples)

- very close to the original codes
- sometimes even smaller!
- very efficient (<1min on 20/25)



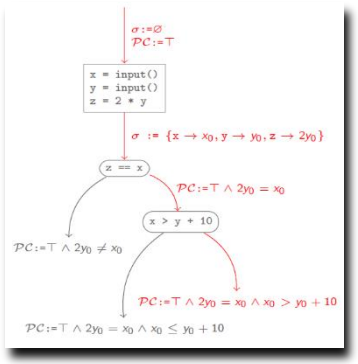
CASE-STUDY: DEVIRTUALIZATION (fun facts)

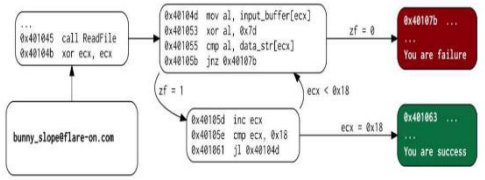
- **Duplicate opcodes: merged!**
- **Nested VMs**
  - **k=2: ok (laptop)**
  - **k=3: ok (cloud)**
- **Also tested vs each VM-option**

	Challenge-0	Challenge-1	Challenge-2	Challenge-3	Challenge-4
VM 0	3.85 seconds	9.20 seconds	3.27 seconds	4.26 seconds	1.58 seconds
VM 1	1.26 seconds	1.42 seconds	3.27 seconds	2.49 seconds	1.74 seconds
VM 2	6.58 seconds	2.02 seconds	2.63 seconds	4.85 seconds	3.82 seconds
VM 3	45.59 seconds	11.30 seconds	8.84 seconds	4.84 seconds	21.64 seconds
VM 4	361 seconds	315 seconds	588 seconds	8040 seconds	1680 seconds
	Few seconds to extract the equation and less than 200 MB of RAM used				
	Few minutes to extract the equation and ~4 GB of RAM used				
	Few minutes to extract the equation and ~5 GB of RAM used				
	Few minutes to extract the equation and ~9 GB of RAM used				
	Few minutes to extract the equation and ~21 GB of RAM used				
	Few hours to extract the equation and ~170 GB of RAM used				

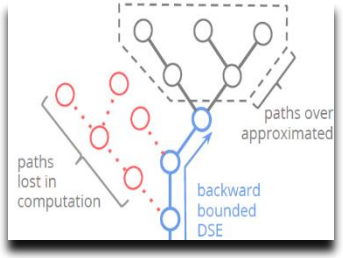
REMINDER: SYMBOLIC DEOBFUSCATION

- **EXPLORE**





- **PROVE**



- **SIMPLIFY**

ARE WE ALL DEAD?

A Generic Approach to Automatic Deobfuscation of Executable Code

Babak Yadegari Brian Johannesmeyer Benjamin Whitely Saumya Debray
Department of Computer Science
The University of Arizona
Tucson, AZ 85721
{babaky, bjohannesmeyer, whitely, debray}@cs.arizona.edu

Code Obfuscation Against Symbolic Execution Attacks

Sebastian Banescu
Technische Universität München
banescu@in.tum.de

Christian Collberg
University of Arizona
collberg@gmail.com

Vijay Ganesh
University of Waterloo
vganesh@uwaterloo.ca

Zack Newsham
University of Waterloo
znewsham@uwaterloo.ca

Alexander Pretschner
Technische Universität München
pretschn@in.tum.de

Backward-Bounded DSE: Targeting Infeasibility Questions on Obfuscated Codes*

Sébastien Bardin
CEA, LIST,
91191 Gif-Sur-Yvette, France
sebastien.bardint@cea.fr

Robin David
CEA, LIST,
91191 Gif-Sur-Yvette, France
robin.david@cea.fr

Jean-Yves Marion
Université de Lorraine,
CNRS and Inria, LORIA, France
jean-yves.marion@loria.fr

Symbolic deobfuscation: from virtualized code back to the original*

Jonathan Salwan[1], Sébastien Bardin[2], and Marie-Laure Potet[3]

Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?

SEBASTIAN SCHRITTWIESER, St. Pölten University of Applied Sciences, Austria
STEFAN KATZENBEISSER, Technische Universität Darmstadt, Germany
JOHANNES KINDER, Royal Holloway, University of London, United Kingdom
GEORG MERZDOVNIK and EDGAR WEIPPL, SBA Research, Vienna, Austria

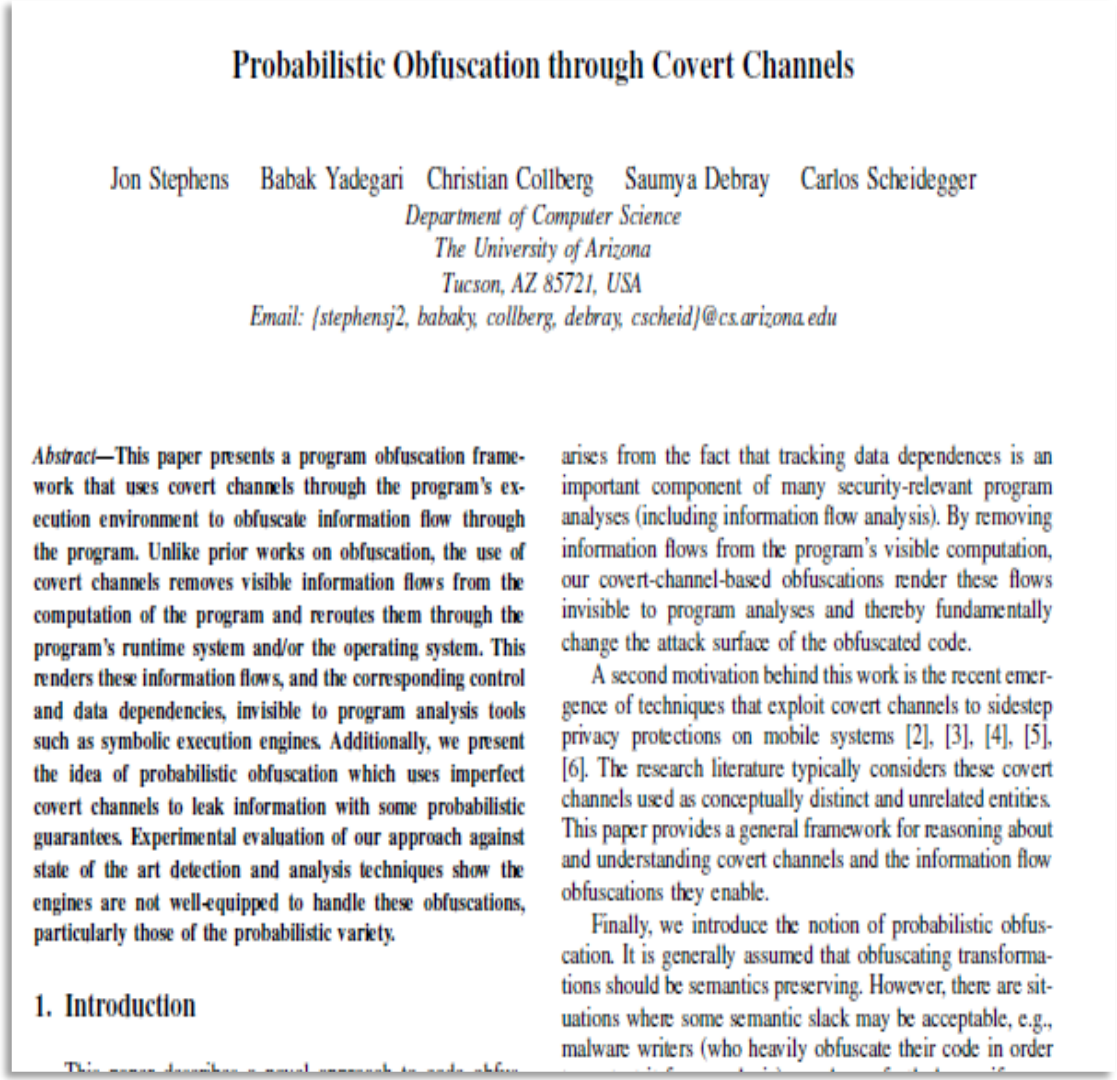
OUTLINE

- **Context: MATE and deobfuscation**
- **Back to the basic: binary-level semantic analysis**
- **Symbolic deobfuscation & achievements**
- **State of the defense**
- **Conclusion**

ANTI-DSE PROPOSALS ARE BLOOMING

- **Hard-to-solve predicates**
  - floats or array intensive formulas
  - mixed boolean arithmetic
  - crypto hash functions x = 42 ➔ hash(x) = 10580
- **Modelling defaults**
  - anti-dynamic, anti-taint, etc.
  - side-channels
- **Beware**
  - protections must be input-dependent, otherwise removed by standard optimizations
- **Hot topic, battle in progress**
  - Tradeoff between performance penalty vs protection?
  - Exact goal of the attacker?

SIDE CHANNELS vs DSE (Collberg et al., Euro S&P 2018)



Probabilistic Obfuscation through Covert Channels

Jon Stephens Babak Yadegari Christian Collberg Saumya Debray Carlos Scheidegger
Department of Computer Science
The University of Arizona
Tucson, AZ 85721, USA
Email: {stephensj2, babaky, collberg, debray, cscheid}@cs.arizona.edu

***Abstract*—This paper presents a program obfuscation framework that uses covert channels through the program's execution environment to obfuscate information flow through the program. Unlike prior works on obfuscation, the use of covert channels removes visible information flows from the computation of the program and reroutes them through the program's runtime system and/or the operating system. This renders these information flows, and the corresponding control and data dependencies, invisible to program analysis tools such as symbolic execution engines. Additionally, we present the idea of probabilistic obfuscation which uses imperfect covert channels to leak information with some probabilistic guarantees. Experimental evaluation of our approach against state of the art detection and analysis techniques show the engines are not well-equipped to handle these obfuscations, particularly those of the probabilistic variety.**

1. Introduction

arises from the fact that tracking data dependences is an important component of many security-relevant program analyses (including information flow analysis). By removing information flows from the program's visible computation, our covert-channel-based obfuscations render these flows invisible to program analyses and thereby fundamentally change the attack surface of the obfuscated code.

A second motivation behind this work is the recent emergence of techniques that exploit covert channels to sidestep privacy protections on mobile systems [2], [3], [4], [5], [6]. The research literature typically considers these covert channels used as conceptually distinct and unrelated entities. This paper provides a general framework for reasoning about and understanding covert channels and the information flow obfuscations they enable.

Finally, we introduce the notion of probabilistic obfuscation. It is generally assumed that obfuscating transformations should be semantics preserving. However, there are situations where some semantic slack may be acceptable, e.g., malware writers (who heavily obfuscate their code in order

- **Idea = use side channels for communicating information**
- **Rational: DSE does not take the physical world into account, get confused**
- **Example: concurrent threads writing x, one slow and one fast**
- **Blinds DSE, but « probabilistic correctness» only**

HARD-TO-SOLVE PREDICATES

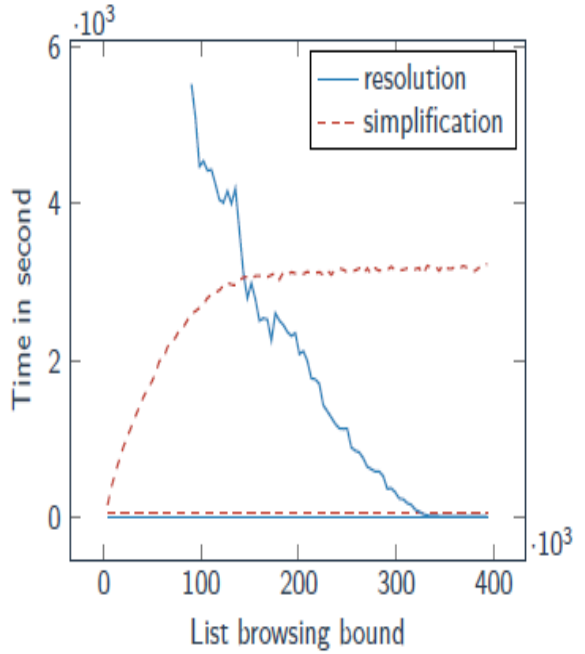
- **Beware: the solver guys are incredible!**
- **Floats ➔ solutions start to emerge (SMTCOMP, Colibri@CEA)**
- **Arrays ➔ progress, see next slide**
- **Mixed Boolean Arithmetic ➔**
  - some partial solutions [N. Eyrolles]
  - effects of MBA hard to predict in advance
  - depend on setting: solve vs simplify
- **Crypto hash functions ➔ highly powerful (but take care of Marine M.)**
  - cost? stealth?
  - becomes a WB issue ➔ finds the key

Tuning the solver: array formula simplification [LPAR 2018] with Benjamin Farinier

- **Makes the difference!**

no block cypher	Z3		#select all arrays	#select 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

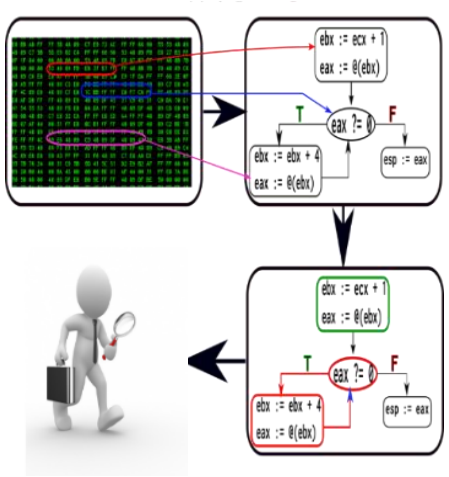
Using list representation

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

OUTLINE

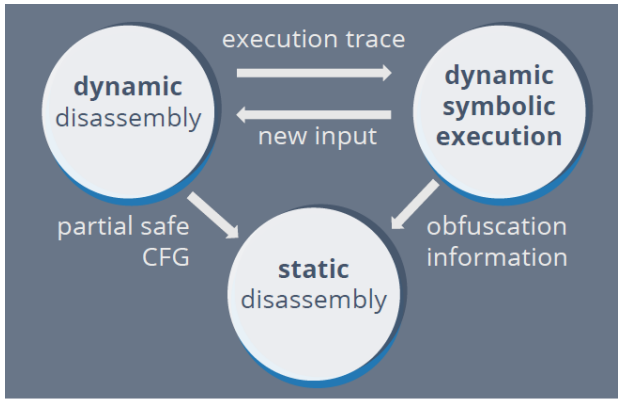
- Context: MATE and deobfuscation
- Back to the basic: binary-level semantic analysis
- Symbolic deobfuscation & achievements
- State of the defense
- Conclusion

CONCLUSION & TAKE AWAY



- **A tour on the advantages and limits of symbolic deobfuscation**

- **Symbolic deobfuscation complements existing approaches!**
  - *Well-adapted – semantics is invariant by obfuscation*
  - *Explore, prove, simplify*
  - ➔ **defenders have to take it into account!**



- **The arm race is still on**
  - *Anti-symbolic and anti-anti-symbolic methods*
  - *Open the way to fruitful combinations (attack & defense)*

- **Still many rooms to explore**
  - *Deobfuscation for malware detection*
  - *Tradeoffs power – detection – ressources*

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