Toward
Security-oriented Program Analysis

Sébastien Bardin
(CEA LIST, University Paris Saclay)

Context: A Urgent Need for Automated Security Analysis

- **Shortage of security experts VS growing demand in expertise**
  - example: European certification schemes for IoT

- **Even for experts, very hard to take everything into account**
  - Software can be huge and extremely complex
  - Errors may arise in many different layers
  - Many different classes of attacks

- **Focus on code-level security**
- **Implementation flaws / attacks**

IN A NUTSHELL

- **Formal methods and program analysis very successful in *safety***
- **First applications to *security* do exist, still pretty much *safety in disguise***
- **Question: *how does code-level security differ from code-level safety?***
- **Challenge: *how to move from safety-oriented code analysis to security-oriented code analysis***
- ***This talk: our experience on adapting source-level safety analysis to the case of binary-level security***

OUTLINE

- **Looking back: the success of formal methods for safety**
- **Safety Is Not Security**
- **From source-level safety to binary-level security: some examples**
- **Conclusion**

ABOUT FORMAL METHODS AND CODE ANALYSIS

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

Key concepts : $M \models \varphi$

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

Success in (regulated) safety-critical domains

ABOUT FORMAL METHODS AND CODE ANALYSIS

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

- Reason about the meaning of programs

Key concepts : $M \models \varphi$

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

- Reason about infinite sets of behaviours

Success in (regulated) safety-critical domains

A DREAM COME TRUE … IN CERTAIN DOMAINS

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

WAIT ??!!! Verification is undecidable

Cannot have analysis that
- Terminates
- Is perfectly precise

On all programs

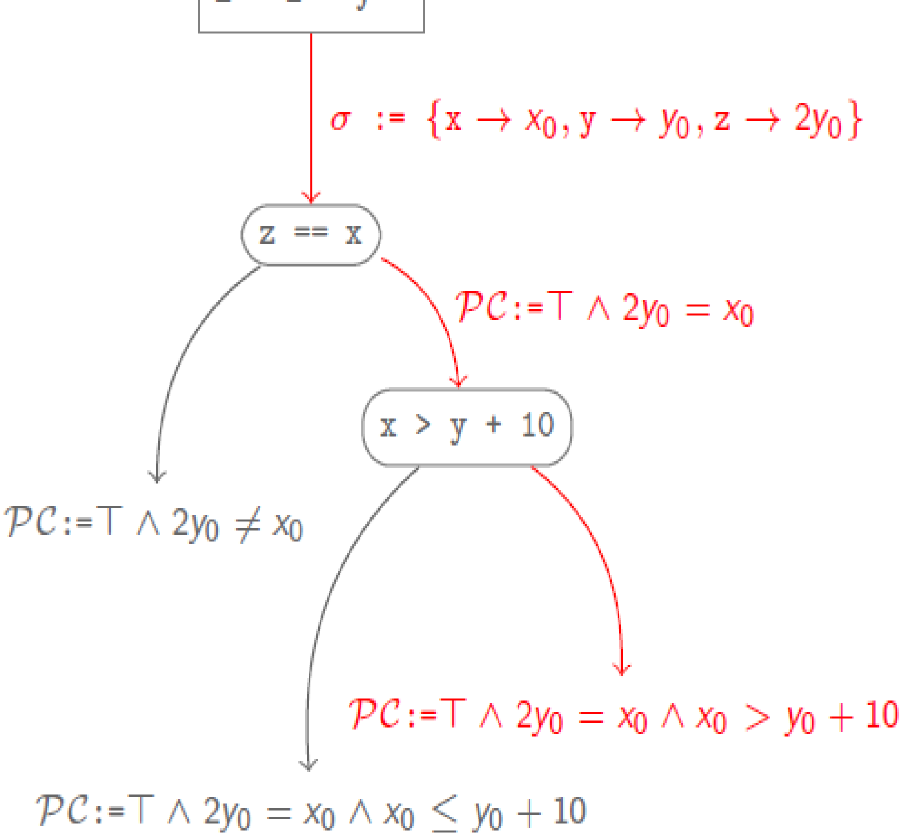
Answers
- Forget perfect precision: proofs only
- Forget perfect precision: bugs only
- Forget termination
- Focus only on « interesting » programs
- Put a human in the loop

SYMBOLIC EXECUTION (Godefroid 2005, Sen 2005, Cadar 2005, …)

Find real bugs

Bounded verification

```
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

OUTLINE

- Looking back: the success of formal methods for safety

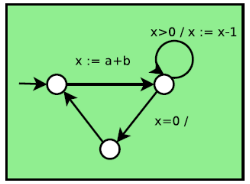
- Safety Is Not Security

- From source-level safety to binary-level security: some examples

- Conclusion

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
```

- Binary code

- Attacker

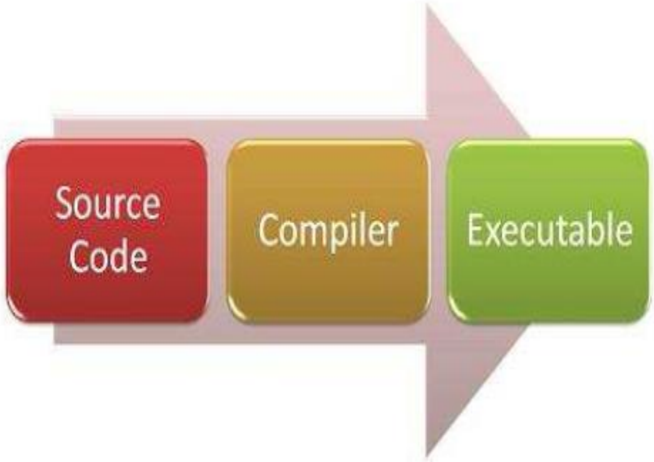
- Properties

OUTLINE

- **Looking back: the success of formal methods for safety**
- **Safety Is Not Security**
  - Going down to binary
  - Adversarial setting
  - « True security » properties
- **From source-level safety to binary-level security: some examples**
- **Conclusion**

WHY?

Post-compilation



No source code

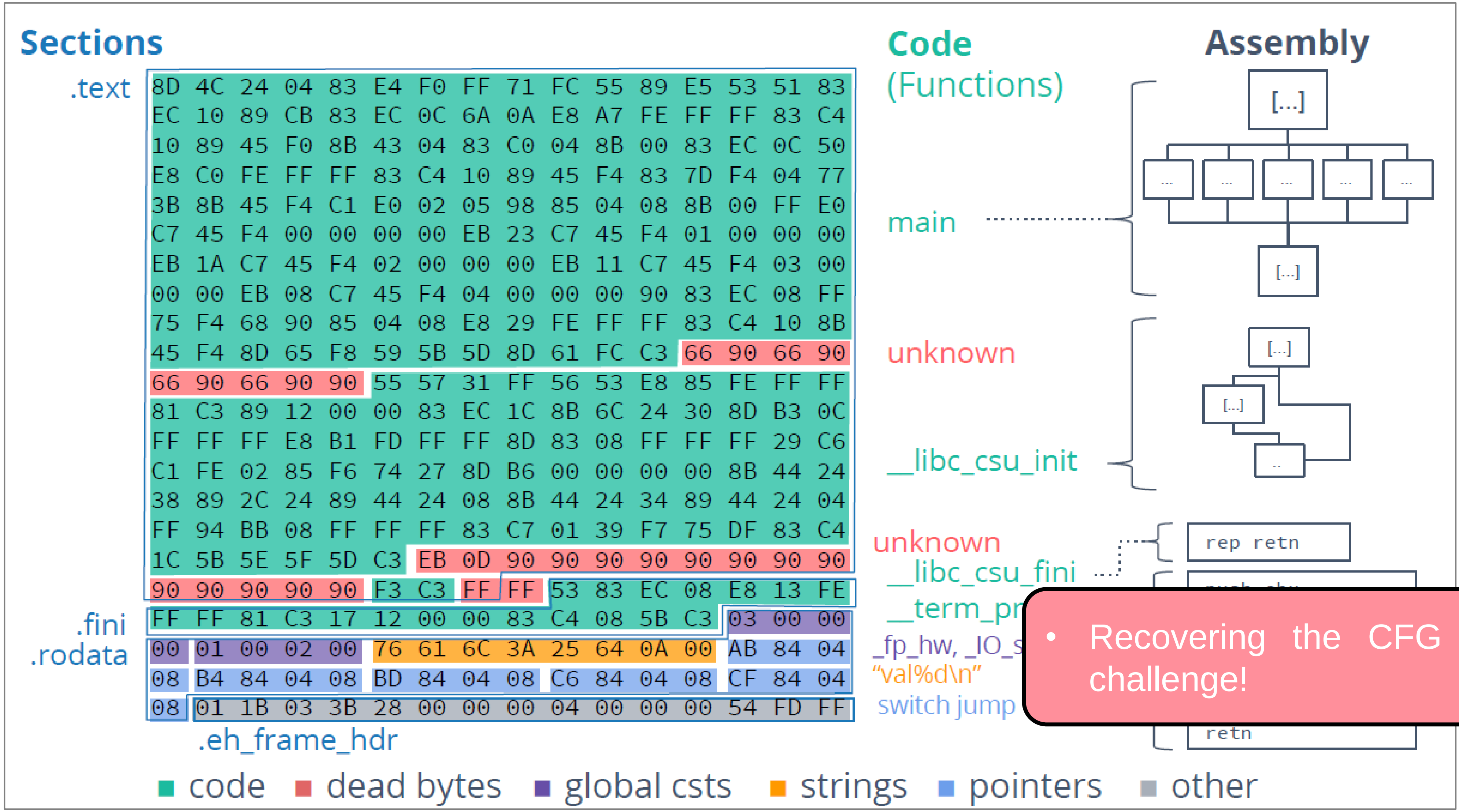
Malware comprehension

Protection evaluation

CHALLENGE: BINARY CODE LACKS STRUCTURE

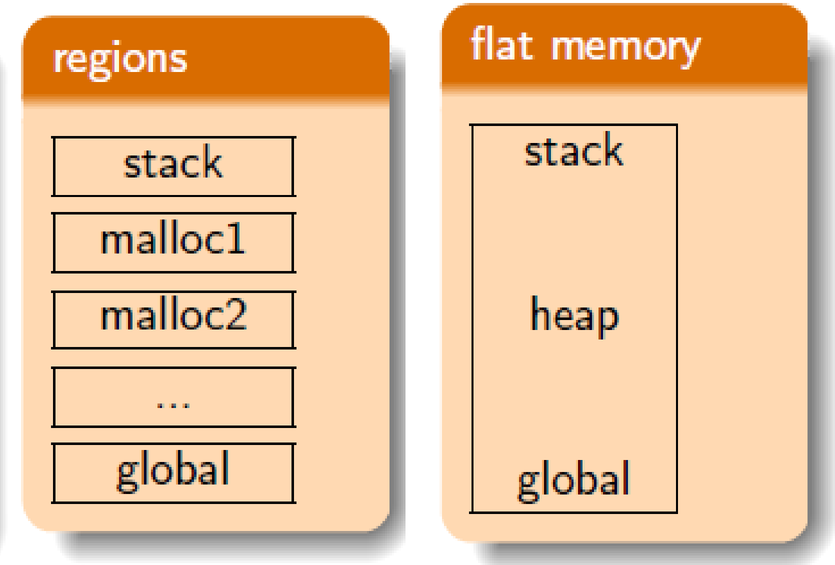
DISASSEMBLY IS ALREADY TRICKY!

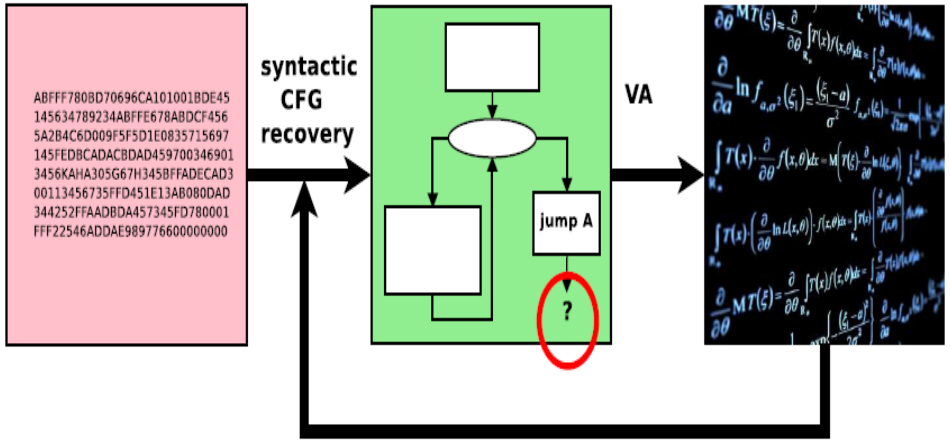
- code – data ??
- dynamic jumps (jmp eax)



- Recovering the CFG is already a challenge!

BINARY CODE SEMANTIC LACKS STRUCTURE






```
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:
```

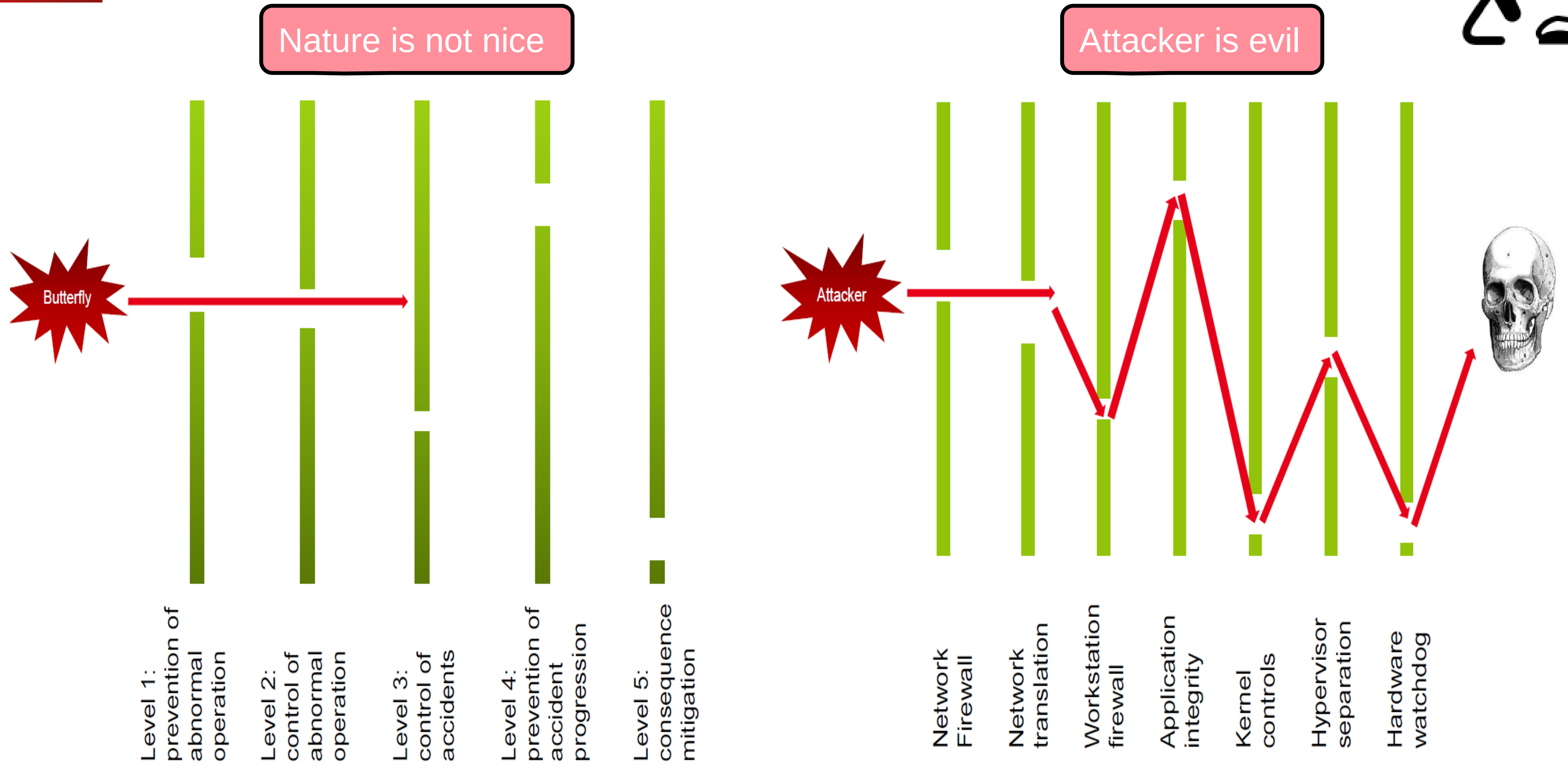
Problems

- Jump eax
- Untyped memory
- Bit-level resoning

OUTLINE

- **Looking back: the success of formal methods for safety**
- **Safety Is Not Security**
  - Going down to binary
  - Adversarial setting
  - « True security » properties
- **From source-level safety to binary-level security: some examples**
- **Conclusion**

CHALLENGE: ATTACKER



ATTACKER in Standard Program Analysis

- We are reasoning worst case: seems very powerful!

ATTACKER in Standard Program Analysis

- **We are reasoning worst case: seems very powerful!**

- **Still, our current attacker plays the rules: respects the program interface**
  - Can craft very smart input, but only through expected input sources

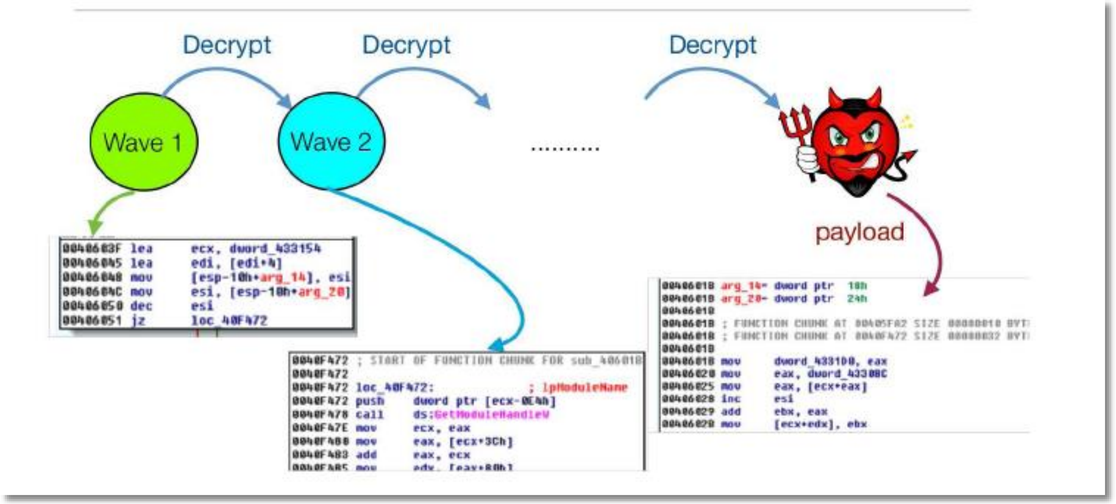
ATTACKER in Standard Program Analysis

- **We are reasoning worst case: seems very powerful!**

- **Still, our current attackers play the rules: respects the program interface**
  - Can craft very smart input, but only through expected input sources

- **What about someone who do not play the rules?**
  - Side channel attacks
  - Hardware fault injection

ADVERSARIAL BINARY CODE



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

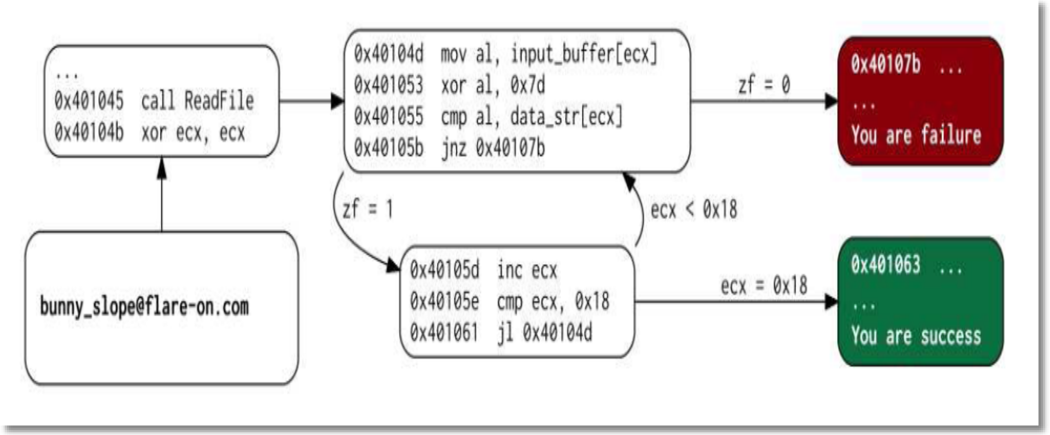
eg: $7y^2 - 1 \neq x^2$

(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**
- **…**



OUTLINE

- Looking back: the success of formal methods for safety

- Safety Is Not Security
  - Going down to binary
  - Adversarial setting
  - « True security » properties

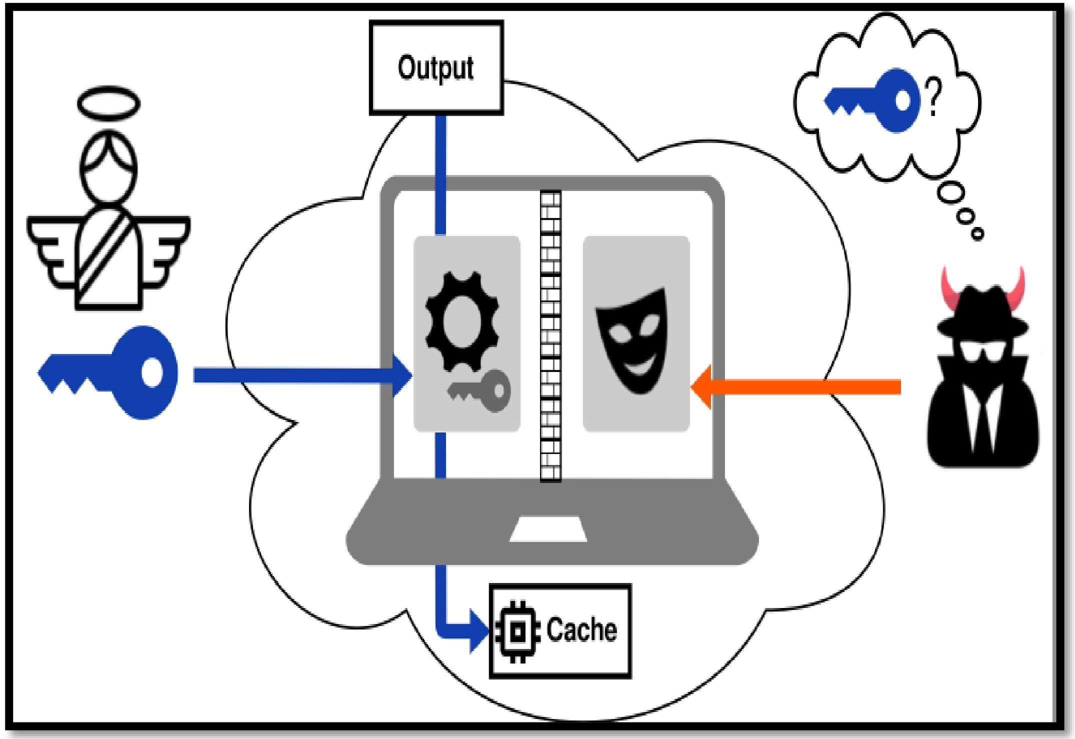
- From source-level safety to binary-level security: some examples

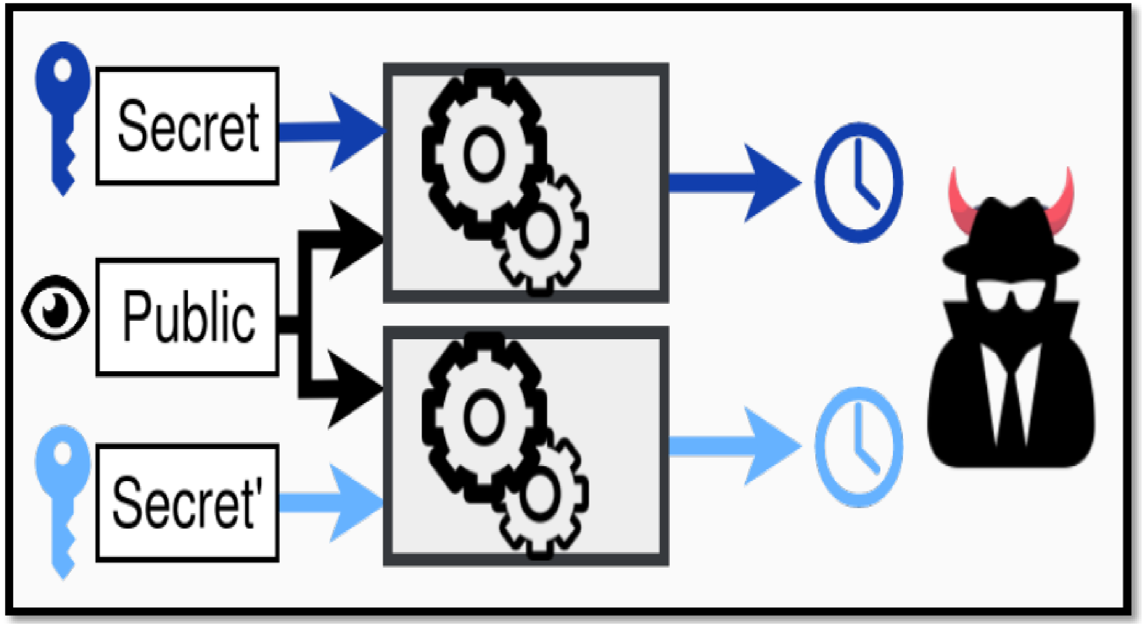
- Conclusion

EXAMPLE

Information leakage

Properties over pairs of executions





TRUE SECURITY PROPERTIES

- **Hyper-properties**
  - Non-interference, timing attacks, secret erasure
- **From bugs to exploits**
- **Quantitative reasoning**
  - Leaking 1 bit is not that important …
- …

OUTLINE

- Looking back: the success of formal methods for safety
- Safety Is Not Security
- **From source-level safety to binary-level security: some examples**
- Conclusion

OUTLINE

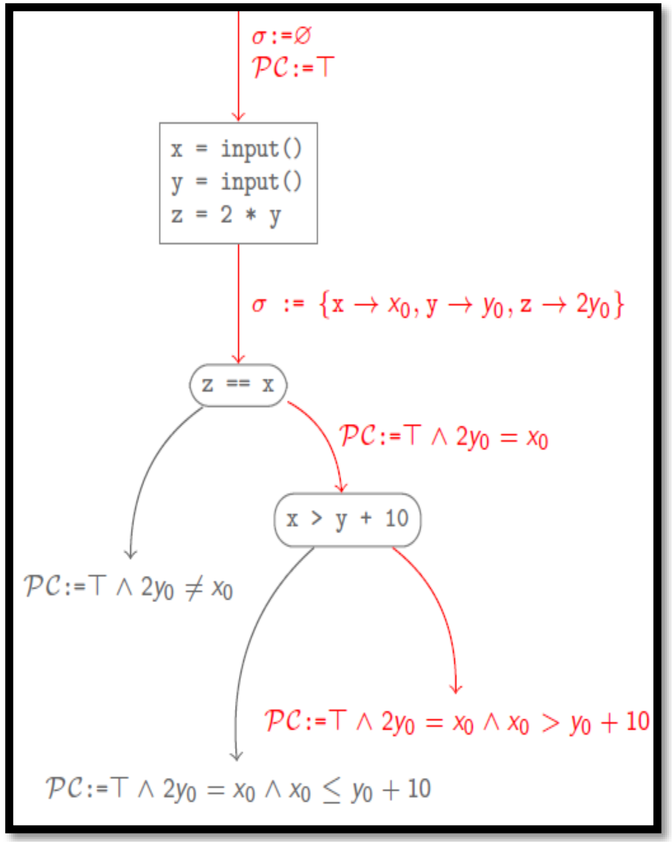
- **Looking back: the success of formal methods for safety**

- **Safety Is Not Security**

- **From source-level safety to binary-level security: some examples**
  - Vulnerability finding and exploit generation
  - Side channel attacks
  - Fault injection
  - Reverse of adversarial code

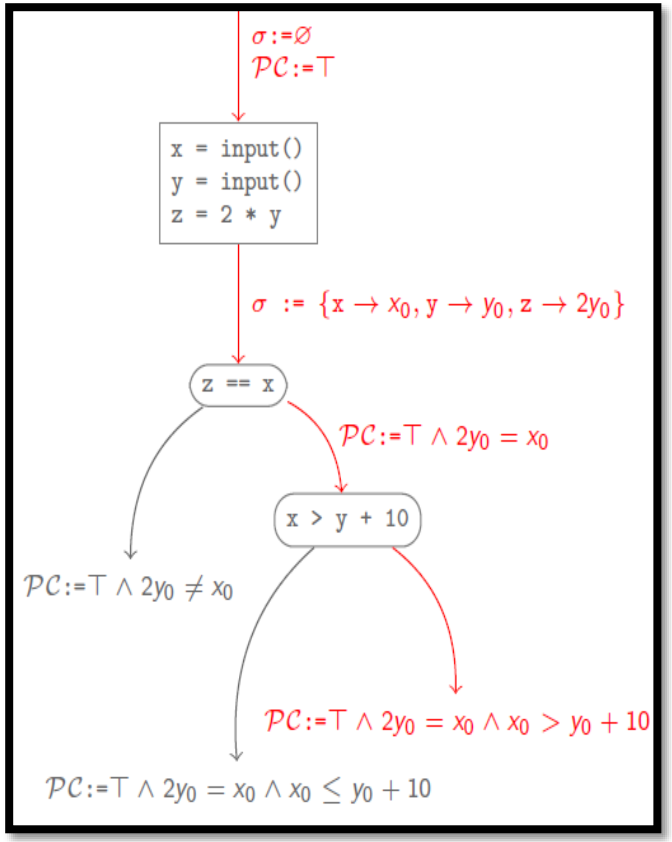
- **Conclusion**

Path coverage with symbolic execution



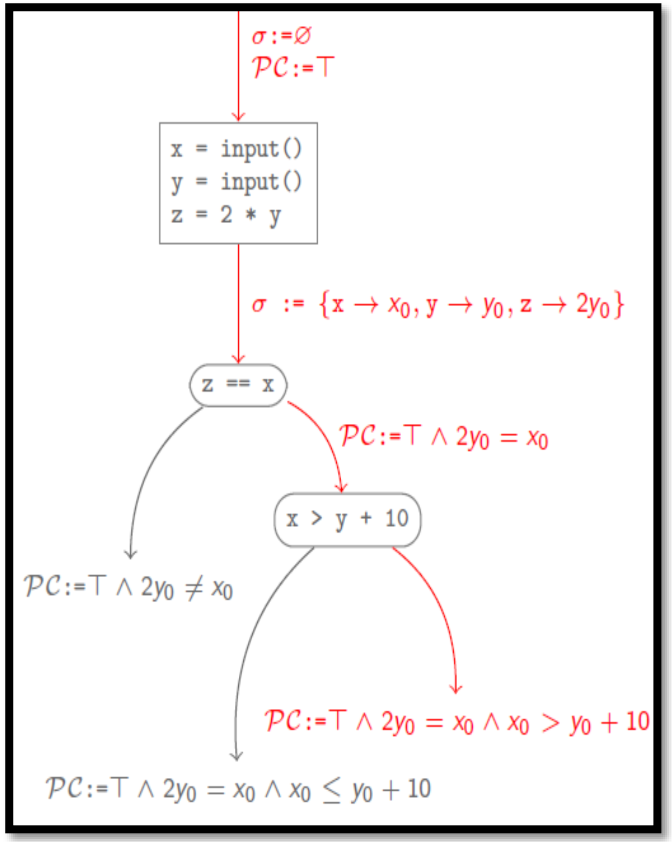
- Intensive path exploration

Vulnerability finding with symbolic execution



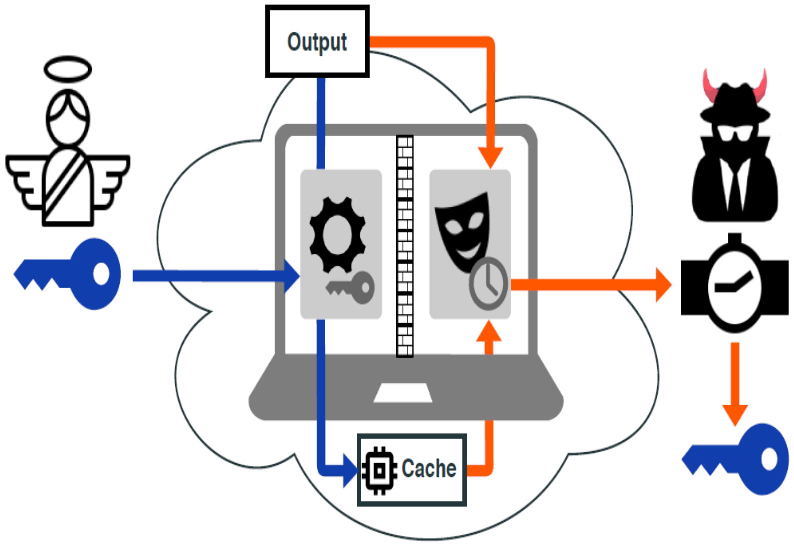
- Intensive path exploration
- Target critical bugs

Exploit finding with symbolic execution [Heelan 2012, Brumley 2014]



- Intensive path exploration
- Target critical bugs
- Directly create simple exploits

SECURING CRYPTO-PRIMITIVES -- [S&P 2020] (Lesly-Ann Daniel)



		#Instr static	#Instr unrol.	Time	CT source	Status		Comment
utility	ct-select	735	767	.29	Y	21×✗	21	1 new ✗
	ct-sort	3600	7513	13.3	Y	18×✗	44	2 new ✗
BearSSL	aes_big	375	873	1574	N	✗	32	-
	des_tab	365	10421	9.4	N	✗	8	-
OpenSSL	tls-remove-pad-lucky13	950	11372	2574	N	✗	5	-
Total		6025	30946	4172	-	**42** ×✗	110	-

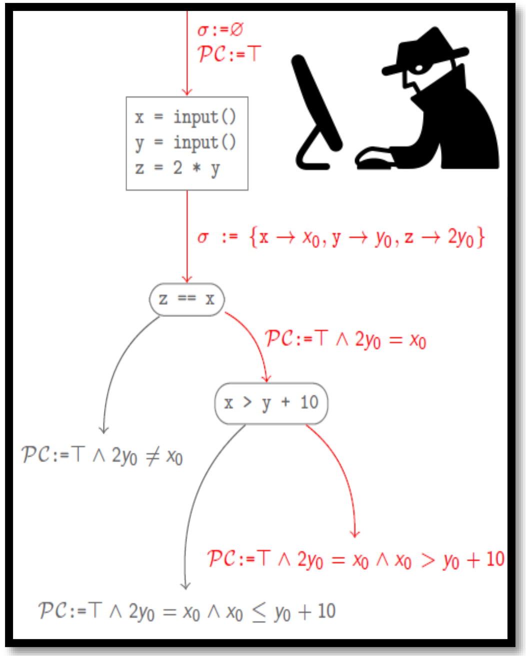
- Relational symbolic execution
- Follows paires of execution
- Check for divergence

- 397 crypto code samples, x86 and ARM
- New proofs, 3 new bugs (of verified codes)
- 600x faster than standard approach

FAULT INJECTION: PROTECTION EVALUATION
(Maxime Puys, Louis Dureuil, Marie-Laure Potet, Laurent Mounier)

Standard SE + fault model

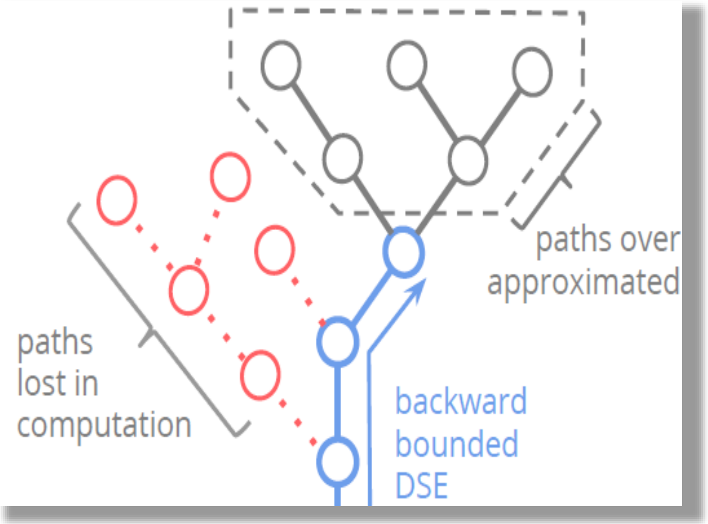
Security-critical code + CFI protections



Is the code secure?

REVERSE OF ADVERSARIAL MALWARE
-- [BlackHat EU 2016, S&P 2017] (Robin David)

- Backward-bounded SE
- + dynamic analysis



Two heavily obfuscated samples
- **Many opaque predicates**

Goal: detect & remove protections
- Identify 40% of code as spurious
- Fully automatic, < 3h

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

OUTLINE

- Looking back: the success of formal methods for safety
- Safety Is Not Security
- From source-level safety to binary-level security: some examples
- Conclusion

CONCLUSION

- **Advanced program analysis techniques can help cybersecurity**
- **Yet, often based on safety-oriented methods**
- **Need a real « security-oriented » code analysis framework**
  - Binary level, attacker model, true security properties
- **Some results in that direction, still many exciting challenges**