

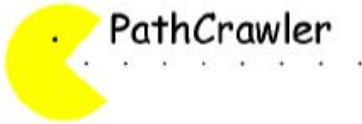
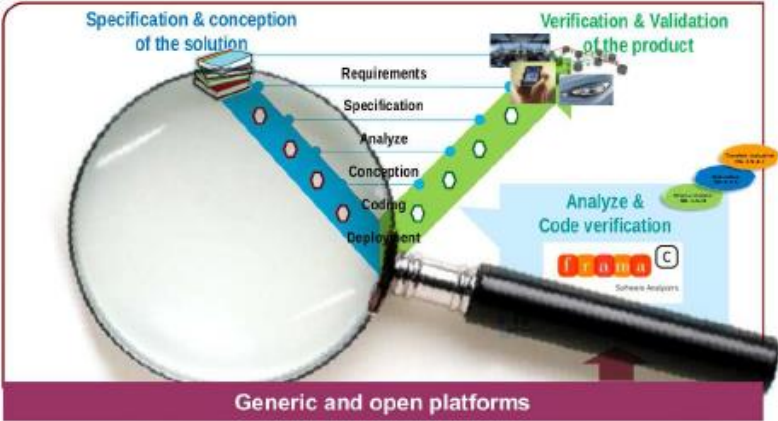
CODE PROTECTION: the promises and limits of symbolic deobfuscation

Sébastien Bardin
(CEA LIST)

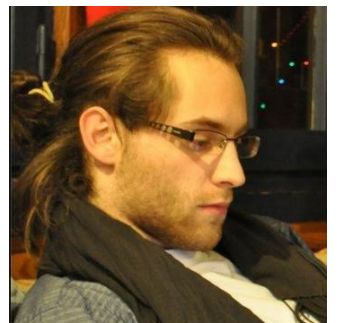
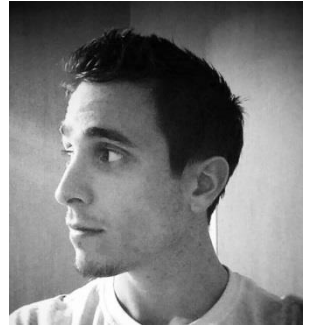


CEA LIST, Software Safety & Security Lab

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

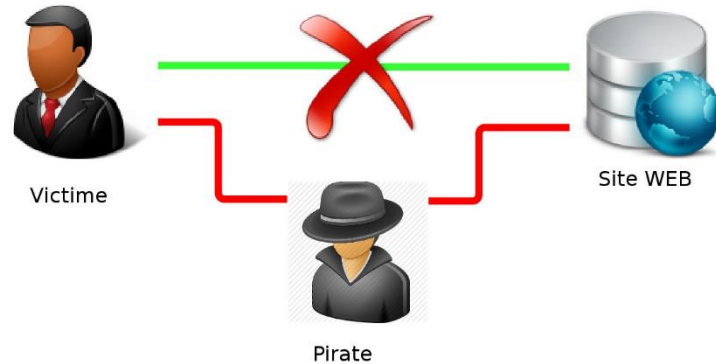


- Challenge: code *deobfuscation*
- Standard tools (dynamic, syntactic) not enough
- ***Semantic methods can help*** [obfuscation preserves semantic]
 - Yet, need to be carefully adapted
- ***A tour on how symbolic methods can help***
 - *Explore and discover* [SANER 2016]
 - *Prove infeasibility* [BH Europe 2016, S&P 2017]
 - *Simplify* [SSTIC 2017]



- **Context**
 - *Code Protection*
 - *Semantic analysis*
- **Symbolic deobfuscation**
 - *Basis: Symbolic execution*
 - *Part I: Explore & Discover* -- *crackme*
 - *Part II: Prove infeasibility* -- *malware x-tunnel*
 - *Part III: Simplify* -- *devirtualization*
- **Conclusion**

MATE: MAN-AT-THE-END ATTACK



MATE: Man-At-The-End

Attacker is *on the computer*

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

New field

MITM: Man-In-The-Middle

Attacker is on the network

- Observe messages
- Forge messages

Known crypto solutions



FACT: SOFTWARE IS JUST DATA

```

00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 |.l.....|...
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 |.....E..
00000020 00 8a f6 46 bb 20 75 08 84 d2 78 07 80 4e bb 40 |...F.u...x...N.@
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 |.V.V...R...l..
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 |o...V.s.....
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 |.t.....
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 |.B...s.X...u.r.
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 |Ht.O...F.....
00000080 be 7b 07 e8 b2 00 8a 56 b9 4e e8 8e 00 eb 05 b0 |{.....V.N.....
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd |...O.....
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 eb 16 |.u.O...9.r..F...
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 |0....<t...<.v.
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 |...<.w...F.s..F.
000000d0 be 00 08 8a 14 89 f3 3c 04 9c 74 0a c0 e0 04 05 |.....<.t.....
000000e0 be 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 |.....S.F.@u...
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb |...Y.^u...V...O.
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f |]...G.r...U...
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 |.].....F.$.
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff |.l.....V.x....
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 |.....U....
00000140 eb f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 |..$.S.....[.t
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 |..L...V...F.t.f
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 |j.f.t.Sj.j...H.
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 |.@...^...Defa
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e |ult:.....
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f |.....?
000001a0 bf 44 4f d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 |.DO.Linu.FreeBS.
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 |f.Drive .....
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....
+
000001f0 00 00 00 00 00 00 00 00 00 00 00 55 aa |.....U.
00000200

```

- You can **execute** it

- But you may prefer to:

- Read it** <reverse legacy code, or steal crypto keys>
- Modify it** <patch a bug, or bypass a security check>

**Code & Data attack
(MATE)**

**Code & Data protection
(obfuscation)**

<aparté> NOT SO HARD FOR EXPERTS

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 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 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

Code (Functions)

- main
- unknown
- __libc_csu_init
- unknown
- __libc_csu_fini
- __term_proc
- _fp_hw, _IO_stdin_used
- switch jump table

Assembly

```

[...]
```

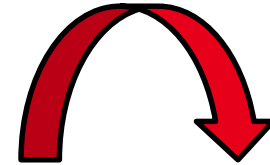
```

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



With IDA

```

0x4013e0 push %ebp
0x4014e1 mov %esp,%ebp
...
0x401419 mov 0xc(%esp),%eax
0x40141d sub $0x4,%eax
0x401420 imul 0xc(%esp),%eax
0x401425 mov %eax,0x4(%esp)
0x401429 cmpl $0x6,0x4(%esp)
0x40142e ja 0x4014a0
```

```

0x40143e push %ebp
0x4014e1 mov %esp,%ebp
...
0x401430 mov 0x4(%esp),%eax
0x401434 shl $0x2,%eax
0x401437 add $0x40a064,%eax
0x40143c mov (%eax),%eax
0x401441 mov %eax,%ecx
0x401446 mov %ecx,%eax
0x40144b jmp *%eax
```

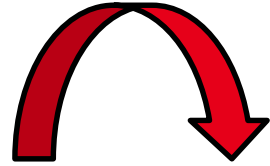
```

0x4015a0 ...
0x4015a5 call D
...
0x401470 ...
0x401475 call F1
...
0x4014f0 ...
0x4014f5 call F2
...
0x4014a0 ...
0x4014a5 call F3
...
0x401450 ...
0x401455 call F0
...
0x4016d0 leave
0x4016d1 ret
```

A SOLUTION: OBFUSCATION

State of the art

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

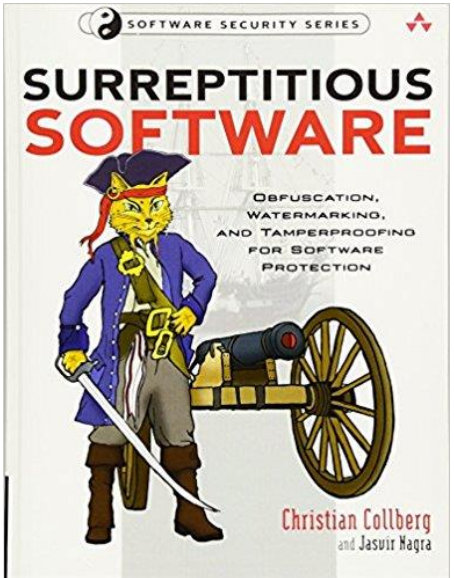


```

    = getStatement();
    sql = "select * from st
    resultSet = statement.executeQu
    if (resultSet.next()) {
        result = true;
        setId(resultSet.getInt("s
        storeDescription = resu
        storeType = r
        storeAdd
    
```

```

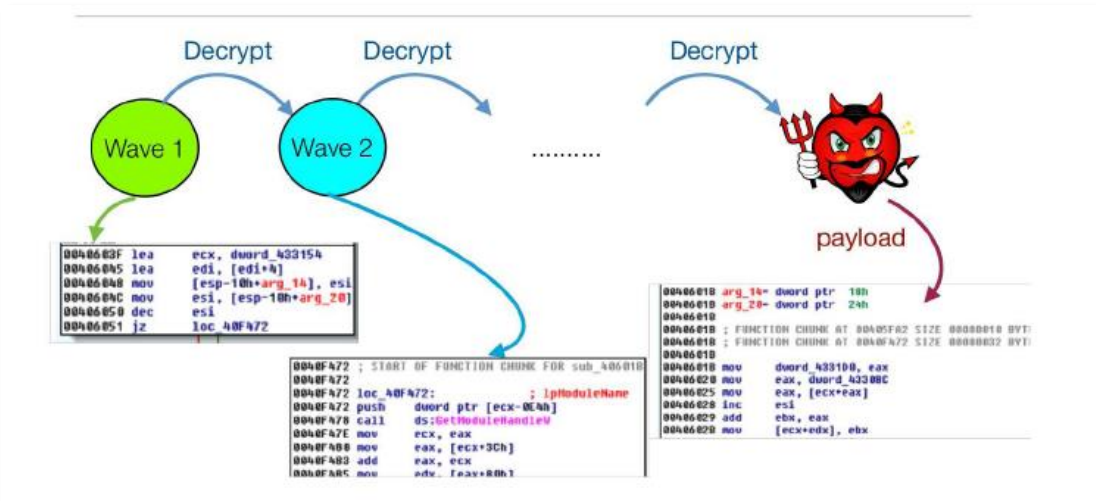
ists($NDtKzAwTCQGqUyz)}{ $marTuzXmEElrbNr->set_sensitive(False); } } if($jrilcGLMcwBxmi!=1){ $HwecPhiIKnsaBY(
boikKUjfvM!=1){ if($CrOorGLihteMbPk=="")$XkLZffvKlHqdYzB=0; switch($CrOorGLihteMbPk) { case 1: $XkLZffvKlHq
urn $AxPGvXMuLrBqSUZ; } function cXBdreLgeQysmbh($ngsHuTaaKlqeKJk){ global $VWgwoCADMvilerx; global $OJfvybOIL
P=$screen_height/$BechHLBAqOgnrXc[1]* $BechHLBAqOgnrXc[0]; } else { $oejysSGfnZAtGQP=$screen_height/$BechHLBA
'ru','2','1','was'); $EQFavHsKCMCIHmV = sqlite_query($NuERFSVleSyVExn, "SELECT lage FROM lage WHERE id=0 "); $
'ru','2','1','was','q'); for ($i = 0; $i <= 8; $i++) { $xBvYwchzFYGttEd=$CrOorGLihteMbPk[$i].'#'; $j++; if($
kTSuioH==""){ $FmZyBrtWLyInYBo}= new GtkRadioButton(null,'',0); $LVUxMyHvkTSuioH=$FmZyBrtWLyInYBo; } else
@QL($image_file){ $ngsHuTaaKlqeKJk=$image_file; $CrOorGLihteMbPk=array('lo','mo','ro','lm','mm','rm','lu','mu
dNg($TBrBtAZPRwFPZYU, $gbeycQSWLKBFFnU, $WVkMIgIGBrVOSjt, $zCjwZmQGLwumGL) { $fSmyLhwpTfAGQil = imagetfbbc
l[1] * $LtcHpLnmFQVedZb - $fSmyLhwpTfAGQil[0] * $LkMbSglLwAjfVfm - $ULabzSbZzHEfrCb; } else { $ULabzSbZzHEfr(
cFcp; $zrxBCrMcVPUjMBo['h']=$KHevYGncDwxvJRF; $zrxBCrMcVPUjMBo['w']=$YUngoXVWLdAOSdJ; return $zrxBCrMcVPUjMBo;
VWcaoJsyxYz-$zrxBCrMcVPUjMBo[1]; if($gbeycQSWLKBFFnU=0){ $iNmEPLiiskpDTlv=-10; }else{ $iNmEPLiiskpDTlv=0; } $iNmE
UrNVTiJdViGHRH=imagesy($WHAB:mHCCyXgNtI)/2- imagesy($maLvSpuqmSzuhJu)/2; If($MwgrEAKeyMnAtiz=='u')$JUAnNBEoXEW
uqmSzuhJu)/2; } If($DugwKypdKwKJBZ=='r'){ $YogbbPXcrLTDQJZ=imagesx($WHAB:mHCCyXgNtI)- imagesx($maLvSpuqmSzuhJu
QjkVQAhLp['g']; $ooVGdSjSyMSNEjt = $JIQuduQjkVQAhLp['b']; } if($LxboJGUoNpBGxm=="height"){ $JIQuduQjkVQAhLp =
DaX = 255; } if($ooVGdSjSyMSNEjt>127){ $ooVGdSjSyMSNEjt = 10; } else{ $ooVGdSjSyMSNEjt = 255; } if($TnBeBOHZdYf
EuTvRzGZLGEI=$NDtKzAwTCQGqUyz; $TBrBtAZPRwFPZYU = getimagesize($tkoEuTvRzGZLGEI); $qYSGvHlDyejMyI=$TBrBtAZPF
($MeQaCJzkQyKNazt>imagesx($WHAB:mHCCyXgNtI)/100*$OAZKDtKsRHRgZwB){ $MeQaCJzkQyKNazt=imagesx($WHAB:mHCCyXgNtI)/
uhJu)-$HLDXcwyfPoYrFK; If($MwgrEAKeyMnAtiz=='o')$JUAnNBEoXEWqJm=$HLDXcwyfPoYrFK; If($MwgrEAKeyMnAtiz=='m')$
($WHAB:mHCCyXgNtI)/2- imagesx($maLvSpuqmSzuhJu)/2; $JUAnNBEoXEWqJm=imagesy($WHAB:mHCCyXgNtI)/2- imagesy($maLv
$WHAB:mHCCyXgNtI)/2- imagesx($maLvSpuqmSzuhJu)/2; } If($DugwKypdKwKJBZ=='r'){ $YogbbPXcrLTDQJZ=imagesx($WHAB:mH
->set_text(''); } $TFnsiSsBvFBsDOb=$GLOBALS['BIoUrBpyspeFLWN']; $TFnsiSsBvFBsDOb->set_text(''); $wENZkUTQbQuHs
WMNTlvuSifIM->get_text()." WHERE id=0"); } function XYyCTuPntLfaveVE() { global $bpAGFKHBLsZx:Fyb; global $NuERFS
XNGBmCFdvbbmWdK." WHERE id=0"); } function EoNVSgEkqaikLsj($BBVRGSKDdXgIVH, $wJfcrfmlBDvDmhp,$ByCzsrSXRTJDP
PLiiskpDTlv->get_text(); if($hvRlKhJmLmhtSzs==0){sqlite_query($NuERFSVleSyVExn, "UPDATE lage SET offset=". $GDw
    
```



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	[...]

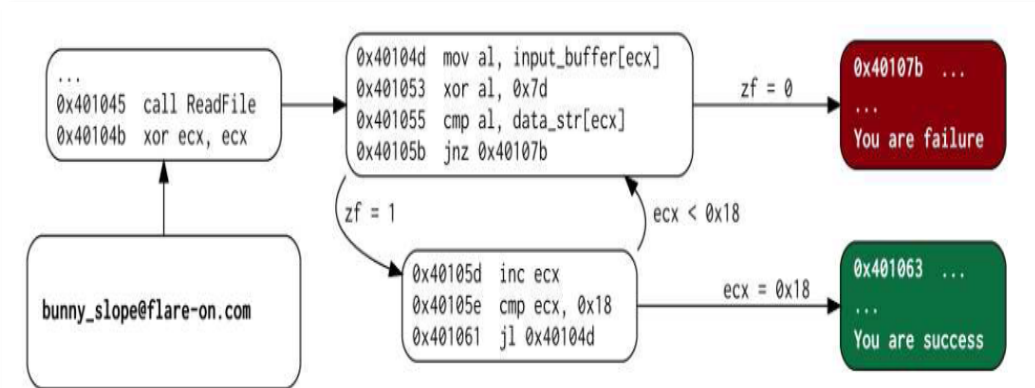


- self-modification
- encryption
- virtualization
- code overlapping
- opaque predicates
- callstack tampering
- ...

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>
    
```



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**

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

EXAMPLE: VIRTUALIZATION

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

Bytecodes - Custom ISA

Fetching

Decoding

Dispatcher

Operator 1

Operator 2

Operator 3

Terminator

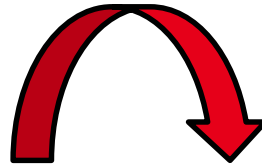
Turns code P into

- a proprietary bytecode program
- + a homemade VM (runtime)
- Easy to recover the VM structure
- But does not say anything about P

```

ists($NDtKzAWTCQGqUyz)}{$marTuzXmMElrNbNr->set_sensitive(False);}}if($ijrilcGLMcVbXmi!=1){$HwecPhiIKnsaBYC
boIkKujfVMI=1}{if($CrOorGLihteMbPk==''}$XkLZffvKlHqdyZB=0;switch($CrOorGLihteMbPk){case1:$XkLZffvKlHq
urn$AxPGvXMuIrBqSUZ;}functioncXBdreLgeOysmbh($ngsHuTaaKlqeKJk){global$VW@woCADMvilerx;global$OJfVybOIk
P=$screen_height/$BechHLBAqOgnrXc[1]*$BechHLBAqOgnrXc[0];}else{$oejysSGfnZAtGQP=$screen_height/$BechHLBA
'ru','2','1','was');$EQFavHsKCMiMmV=sqli_query($MuERFSVleSyVExn,"SELECTlageFROMlageWHEREid=0");if
'ru','2','1','was','q');for($i=0;$i<=8;$i++){ $xBvYwchzFYGttEd=$CrOorGLihteMbPk[$i].'#';$j++;if($C
kTSuioH==''){${$FmZyBrtWLyInYBo}=newGtkRadioButton(null,'',0);$LVUxMyHvkTSuioH=${$FmZyBrtWLyInYBo};}else
gQL($image_file){$ngsHuTaaKlqeKJk=$image_file;$CrOorGLihteMbPk=array('lo','mo','ro','lm','mm','rm','lu','mu
dNg($BtBtAZPRwFPZYU,$gbeycQSWLKBFFNu,$wVikMIgIGbrvOSjt,$zCJjwZmQGNLwmGL){$fSmyLhwPTfAGQil=imagettfbbc
l[1]*$LtcHpLnmFQvedZb-$fSmyLhwPTfAGQil[0]*$LkMbSgluWAjfvfm-$ULabzSbZzHEfrcb};else{$ULabzSbZzHEfrc
cFcp;$zrxBCrMcVPUjMBo['h']=$KHevYGncDwxvJRf;$zrxBCrMcVPUjMBo['w']=$YUhgOXWLDaOSdJ;return$zrxBCrMcVPUjMBo;
VMcaoJsyxYz-$zrxBCrMcVPUjMBo[1];if($gbeycQSWLKBFFNu!=0){$iNmEPLIiskpDTlv=-10;}else{$iNmEPLIiskpDTlv=0;}$iNmE
UrNVTiJdVIgHRH=imagesy($WHABxmHCCyXgntI)/2-imagesy($maLvSpuqmSzuHJu)/2;if($HwgrEAKeyMnAtiz=='u')$JUUrNVTiJdVI
uqmSzuHJu)/2;}if($sDugwKydpKwKJBZ=='r'){$YogbbPXcrLTDqJZ=imagesx($WHABxmHCCyXgntI)-imagesx($maLvSpuqmSzuHJu
QjkVQAhLp['g'];$ooVgdSjSyMSNEjt=$JIQuduQjkVQAhLp['b'];}if($LxboJGUoNpBGxm=="height"){ $JIQuduQjkVQAhLp
DaX=255;}if($ooVgdSjSyMSNEjt>127){$ooVgdSjSyMSNEjt=10;}else{$ooVgdSjSyMSNEjt=255;}if($sTnBeBOHZdYF
EuTvRzGZIGEI=$NDtKzAWTCQGqUyz;$BtBtAZPRwFPZYU=getimagesize($tkoEuTvRzGZIGEI);$qYSGvaHLdyejMyI=$BtBtAZPF
($MeQaCJzkQyKNAzt>imagesx($WHABxmHCCyXgntI)/100*$OAZKDtKsRHRGZwB){$MeQaCJzkQyKNAzt=imagesx($WHABxmHCCyXgntI)/
uhJu)-$HLDXcwuyfPoYrFK;if($HwgrEAKeyMnAtiz=='o')$JUAnNBEoXEWrqJm=$HLDXcwuyfPoYrFK;if($HwgrEAKeyMnAtiz=='m')
($WHABxmHCCyXgntI)/2-imagesx($maLvSpuqmSzuHJu)/2;$JUAnNBEoXEWrqJm=imagesy($WHABxmHCCyXgntI)/2-imagesy($maLv
$WHABxmHCCyXgntI)/2-imagesx($maLvSpuqmSzuHJu)/2);}if($sDugwKydpKwKJBZ=='r'){$YogbbPXcrLTDqJZ=imagesx($WHABxm
->set_text('');}$TFnsiSsBvFBsDob=$GLOBALS['BIOUrBpyspeFLWn'];}$TFnsiSsBvFBsDob->set_text('');$wENZkUTQBQuHs
WNTlvuSitfiM->get_text()."WHEREid=0");}functionXYyCTuPntIFeeVE(){global$bpAGFKHBLsZxFyb;global$MuERFS
XNGBmCFdvbbmWdK."WHEREid=0");}functionEoNVSGekqaikLsj($zBBVRGSKDdXgIVH,$wJfCRfmLBDvDmhp,$ByCzsonSXRtJDP
PLIiskpDTlv->get_text());if($hvRlKhMlMhTszs==0)sqli_query($MuERFSVleSyVExn,"UPDATElageSEToffset=".$GDw

```



```

- = getStatement();
resultSet = "select * from stu
if (resultSet.executeQu
result = true;
setStoreId(resultSet.getSto
storeDescription = resu
storeTypeId = resu
storeAdd

```

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

WHY WORKING ON DEOBFUSCATION? <in an ethical manner>

- **Software protection**

- Assess the power of current obfuscation schemes
- Special case: white-box crypto <hide keys>



- **Malware analysis**

- Comprehension: help to understand the malware <goal, functions, weaknesses>
- Detection: remove the protection layer



DEOBFUSCATION NEEDS TOOLING

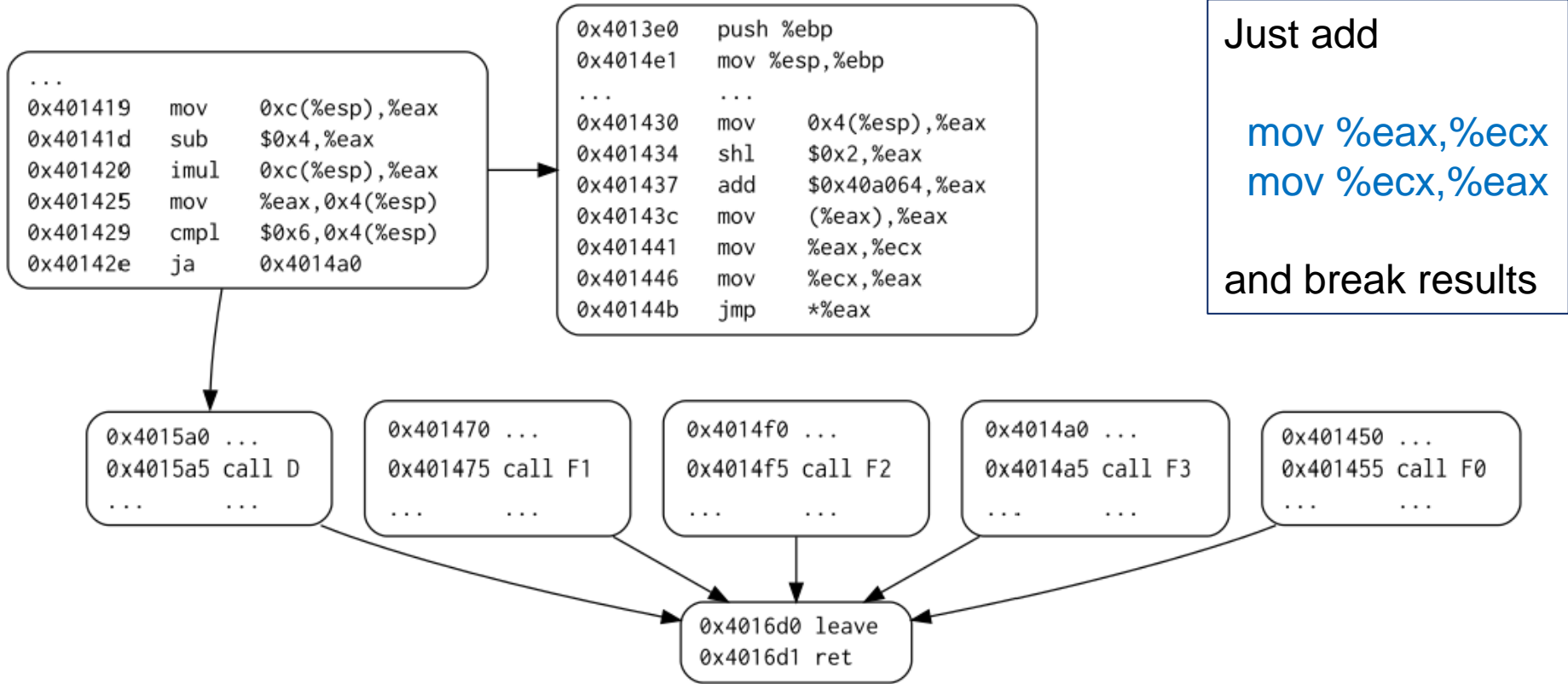
- **Strongly rely on human expert**
- **While obfuscation is automatic**



Proper tool support

- **Explore (find hidden parts)**
- **Prove (identify spurious code)**
- **Simplify**

<aparté> STATE-OF-THE-ART TOOLS ARE NOT ENOUGH FOR DEOBFUSCATION



Just add

```

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

and break results

With IDA

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

SOLUTION? SEMANTIC PROGRAM ANALYSIS

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

**Semantic preserved
by obfuscation**

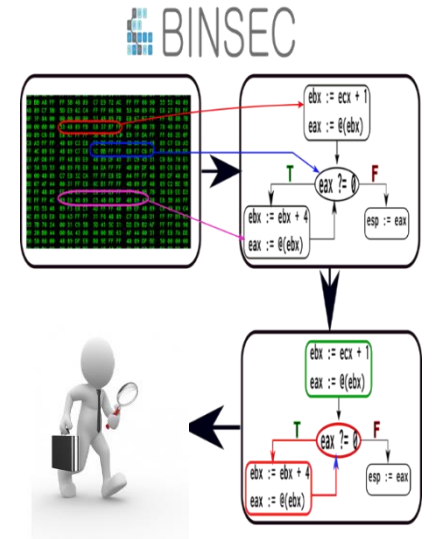
**Can reason about
sets of executions**

- find rare events
- prove, simplify

- ***Symbolic deobfuscation***

- *Explore and discover* [SANER 2016]
- *Prove infeasibility* [Black Hat EU 2016, S&P 2017]
- *Simplify* [SSTIC 2017]

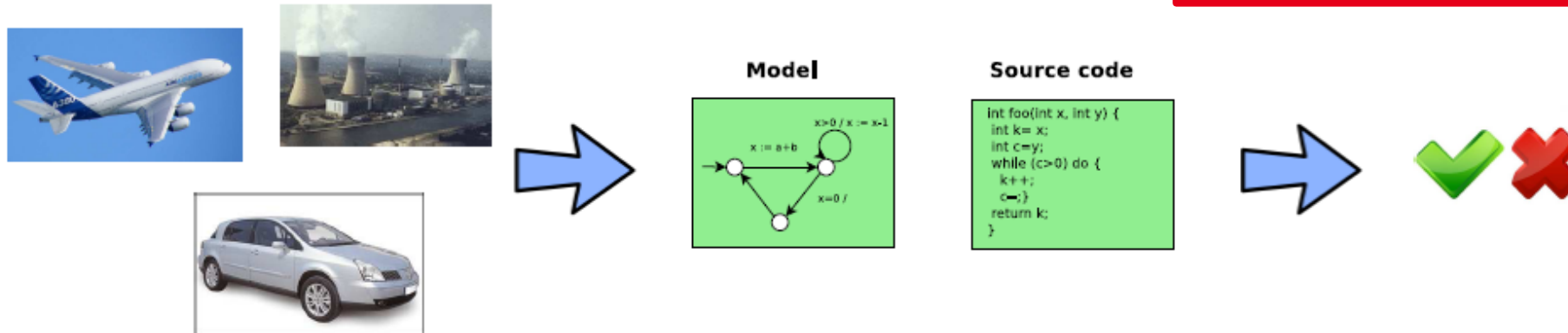
+ strong
theoretical ground



<En apart > ABOUT FORMAL METHODS

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

Clear 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

- **Abstract interpretation**
- **Model Checking**
- **Symbolic model checking**
- **Bounded model checking**
- **Counter-example guided model checking**
- **Interpolation-based model checking**
- **k-induction**
- **Weakest precondition**
- **Property-directed checking**
- **Symbolic execution**
- **Interactive theorem proving**
- **Type systems**
- **Correct by construction**
- **.....**

Constraints

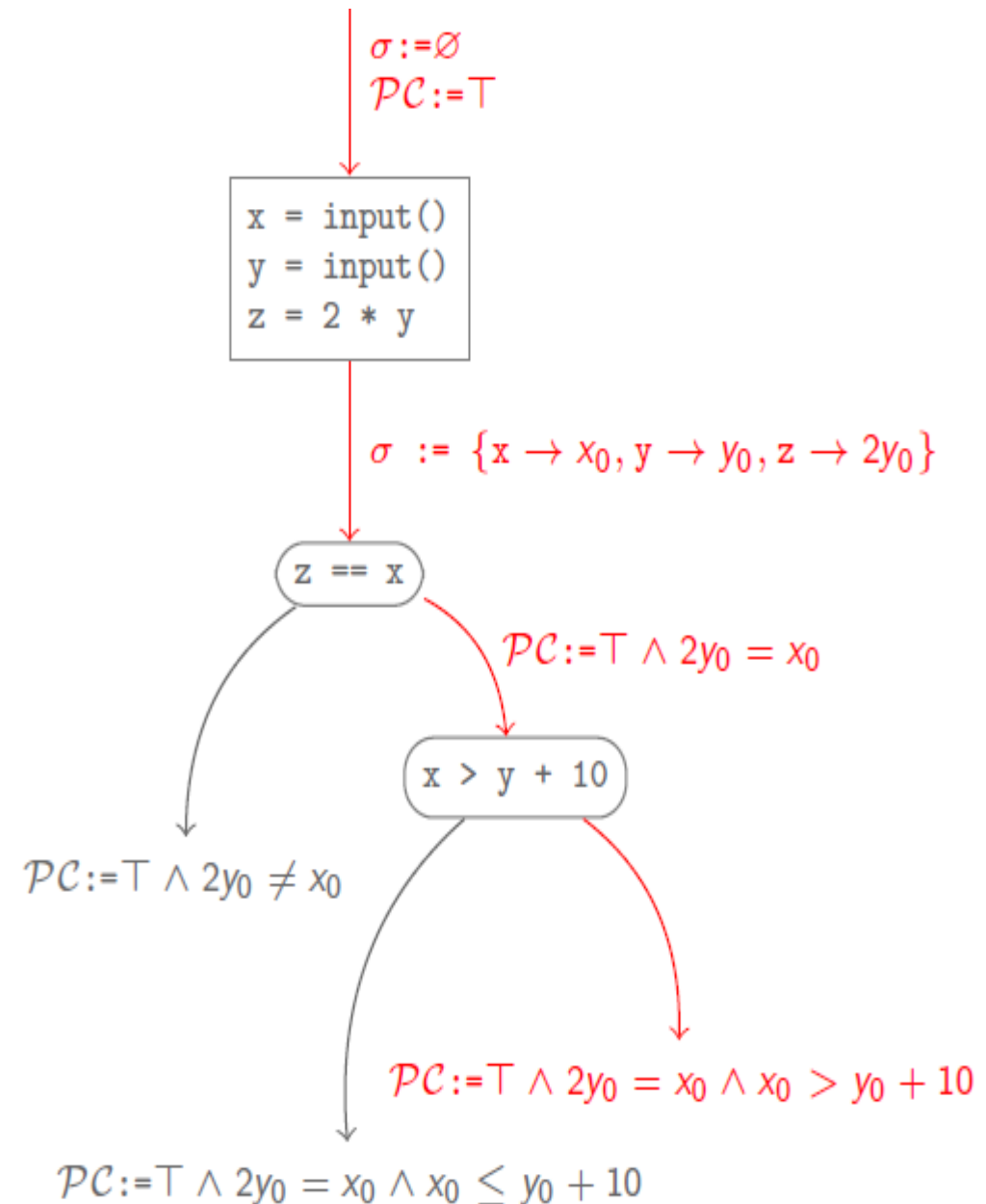
- **Not too hard to adapt to binary level**
- **Robust to nasty low-level tricks**

SYMBOLIC EXECUTION (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



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

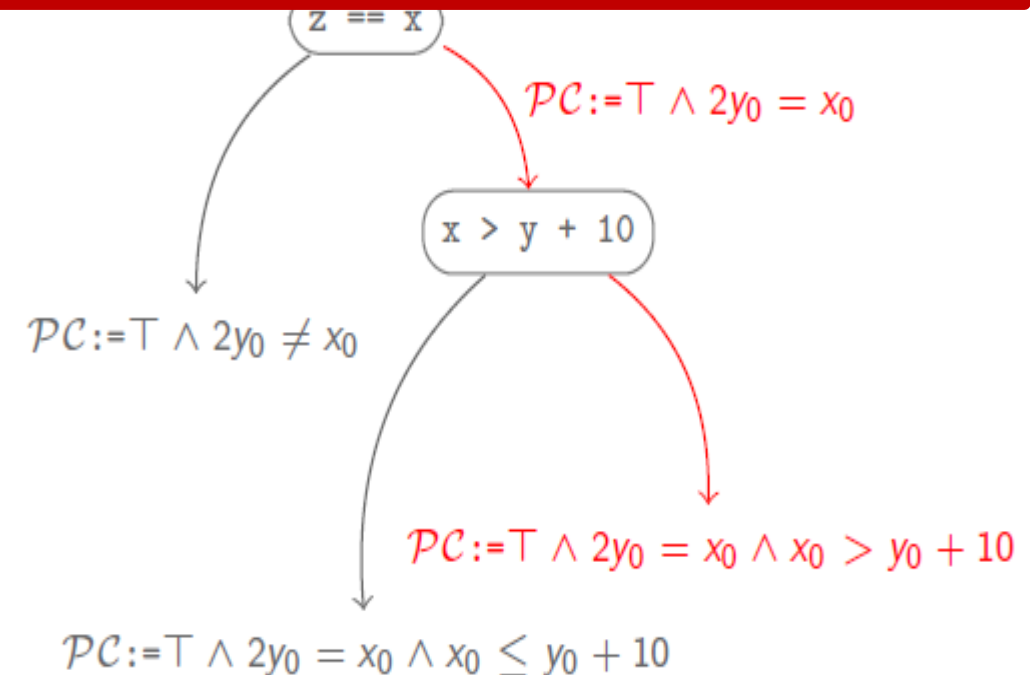
Given a path of a program

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

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

Good points:

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



BINSEC: SYMBOLIC DEOBFUSCATION

x86

```
ABFFF780BD70696CA1010018DE45
145634789234ABFFE678ABDCF456
5A2B4C6D009F5F5D1E0835715697
145FEDBCADACBDAD459700346901
3456KAHA305G67H345BFFADECAD3
00113456735FFD451E13AB080DAD
344252FFAADBDA457345FD780001
FFF22546ADDAE989776600000000
```

ARM

```
ABFFF780BD70696CA1010018DE45
145634789234ABFFE678ABDCF456
5A2B4C6D009F5F5D1E0835715697
145FEDBCADACBDAD459700346901
3456KAHA305G67H345BFFADECAD3
00113456735FFD451E13AB080DAD
344252FFAADBDA457345FD780001
FFF22546ADDAE989776600000000
```

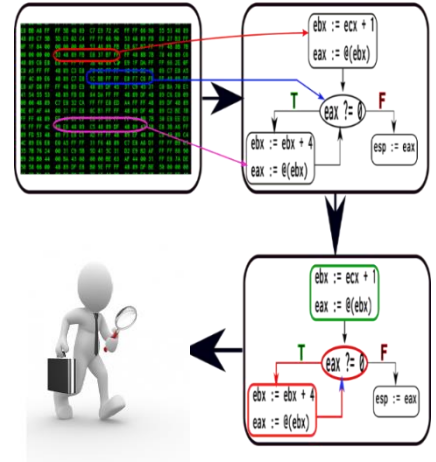
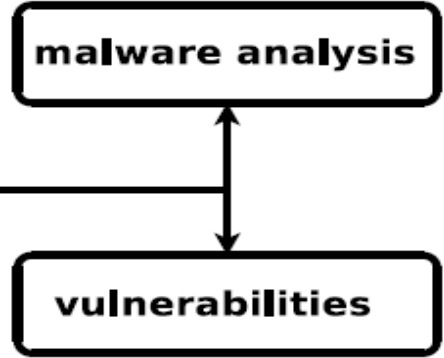
...

```
ABFFF780BD70696CA1010018DE45
145634789234ABFFE678ABDCF456
5A2B4C6D009F5F5D1E0835715697
145FEDBCADACBDAD459700346901
3456KAHA305G67H345BFFADECAD3
00113456735FFD451E13AB080DAD
344252FFAADBDA457345FD780001
FFF22546ADDAE989776600000000
```

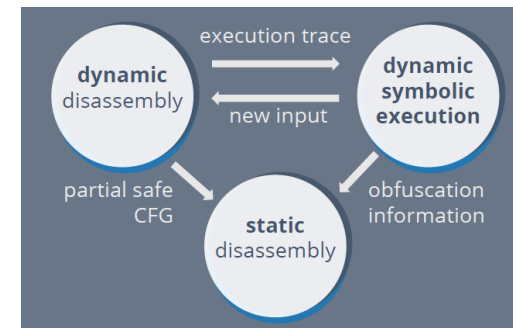
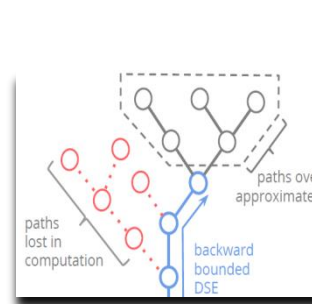
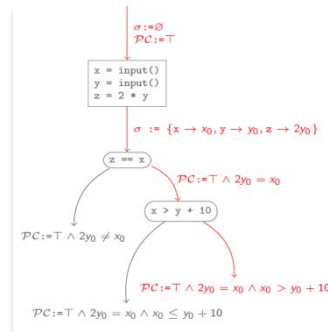
Static analysis



Symbolic execution



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



Forward reasoning

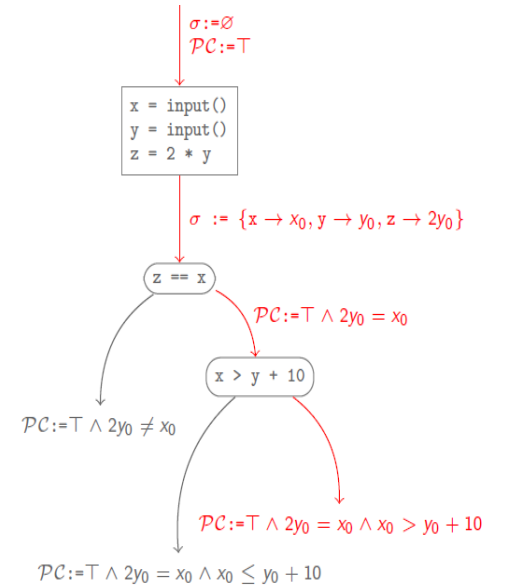
- Follows path
- Find new branch / jumps
- Standard DSE setting

Advantages

- Find new real paths
- Even rare paths

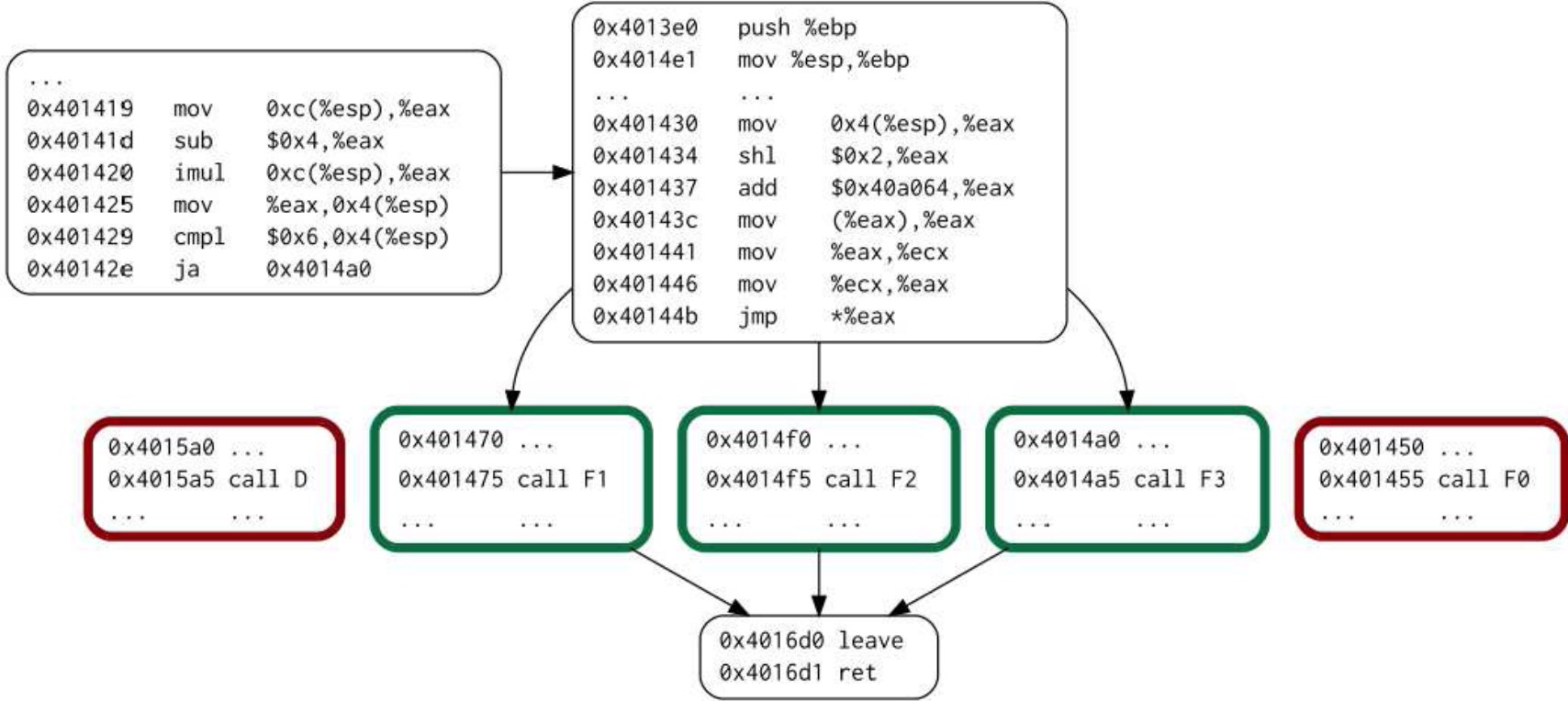
« dynamic analysis on steroids »

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



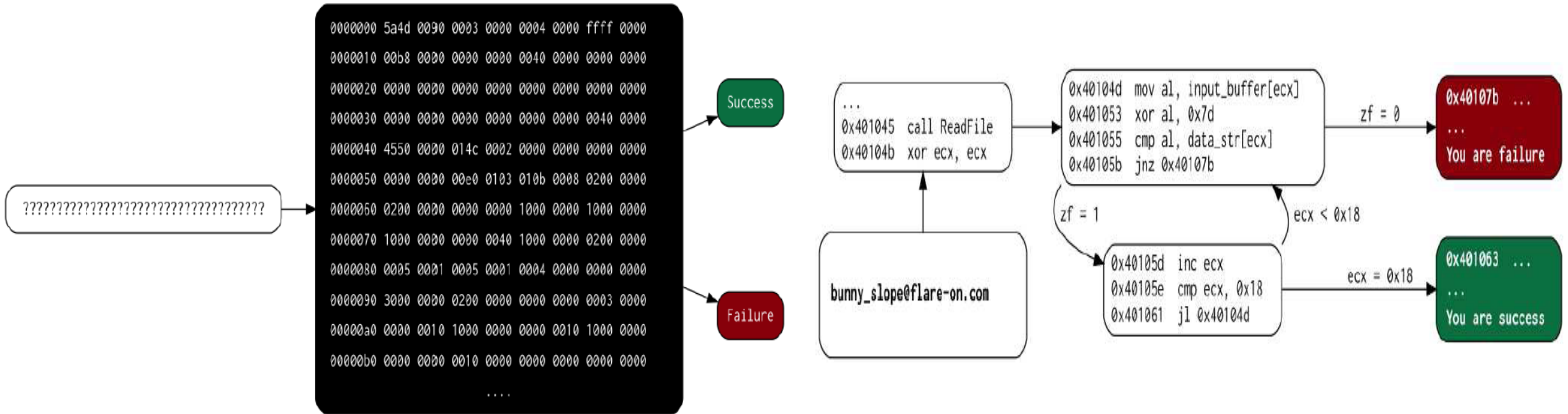
Solve for new dynamic targets

- Get a first target
- Then solve for a new one
- Get it, solve again, ...
- Get them all!



With IDA + BINSEC

EXAMPLE: FIND THE GOOD PATH



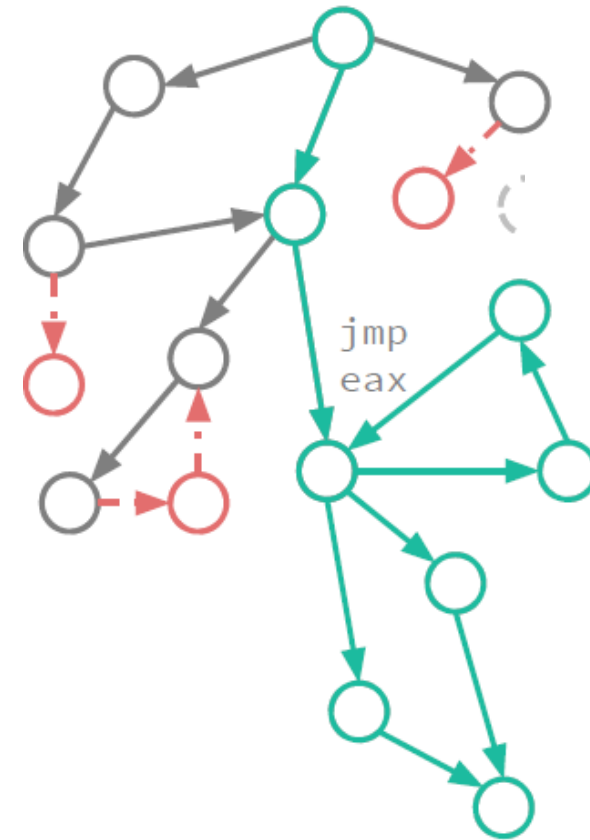
Crackme challenges

- input == secret \mapsto success
- input \neq secret \mapsto failure

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?
- does this expression always evaluate to 15?
- ...



eg: $7y^2 - 1 \neq x^2$
(for any value of x, y in modular
arithmetic)

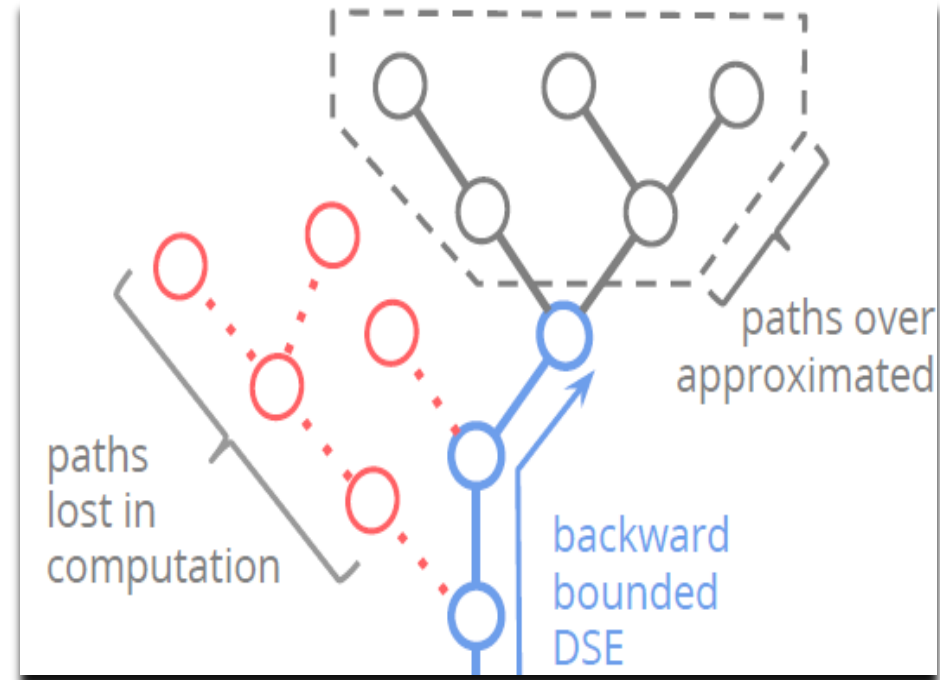
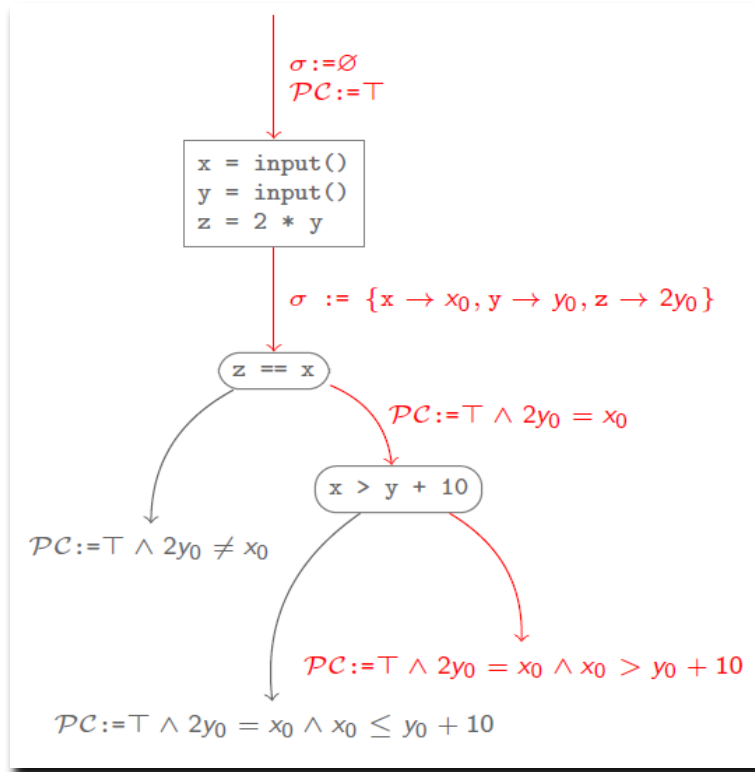
↓

```
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>
```

Not addressed by DSE

- Cannot enumerate all paths

BACKWARD SYMBOLIC EXECUTION



Explore & discover

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

• Prove infeasible

CASE-STUDY: PACKERS

Obsidium
 JD Pack
 WinUpack
 PE Lock
 PE Compact
 Expressor
 Armadillo
 Packman
 EP Protector
 ACProtect
 TElLockSVK
 Yoda's Crypter
 Mew
 Neolite
 UPX MoleBox
 FSG Upack
 Crypter
 Yoda's Protector
 ASPack
 BoxedApp
 Petite
 nPack
 PE Spin
 Enigma
 Setisoft
 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	83	159	0	48
ASPack v2.12	377K	1	1	1	83	24	11	6
Crypter v1.12	1.1M	1	1	1	83	24	125	78
Expressor	635K	1	1	1	83	1	14	0
FSG v2.0	68k	1	1	1	83	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	1	83	1	14	0
TElLock v0.51	406K	1	1	1	83	1	3	1
Upack v0.39	711K	1	1	1	83	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

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

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

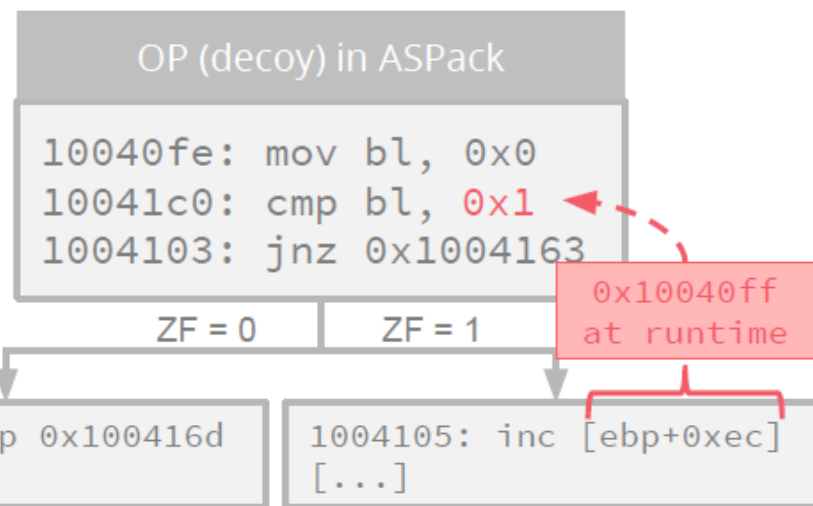
(and all possible variants
ja/jbe, jp/jnp, jo/jno..)

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

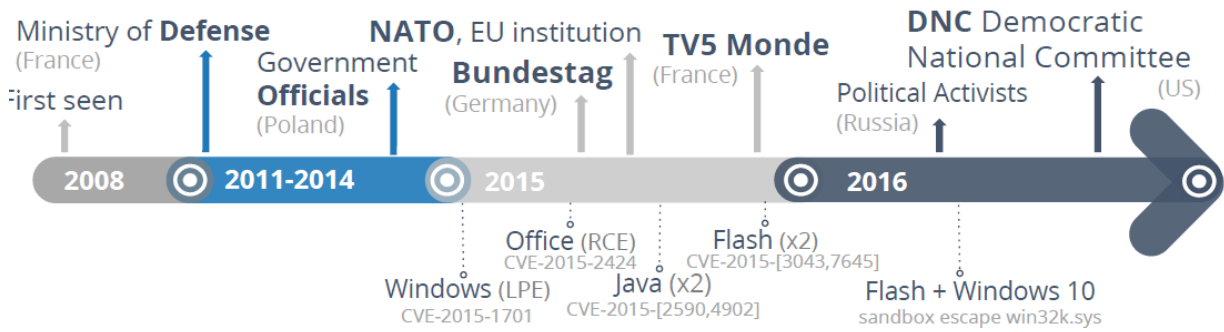
OP in Armadillo		
10330ae	xor	ecx, ecx
10330b0	jnz	0x10330ca

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	



Nicknames: APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm



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
- Yet, quite sophisticated
 - original OPs
 - interleaving between payload and OP computation
 - sharing among OP computations
 - possibly long dependencies chains (avg 8.7, upto 230)

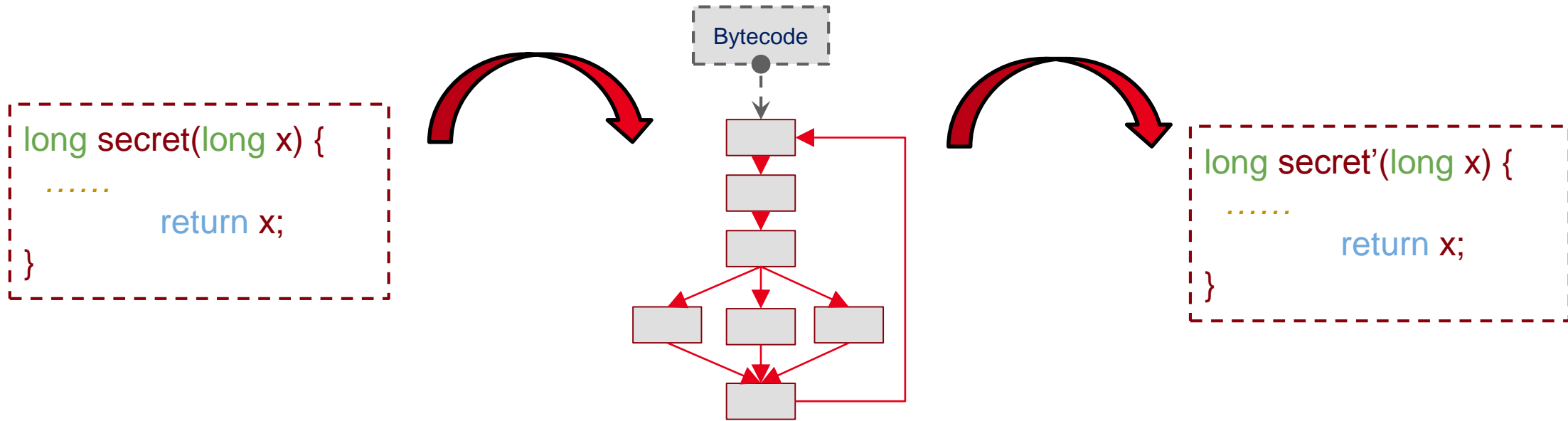
$$7y^2 - 1 \neq x^2 \quad \frac{2}{x^2 + 1} \neq y^2 + 3$$

Why? recover hidden simple expressions

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

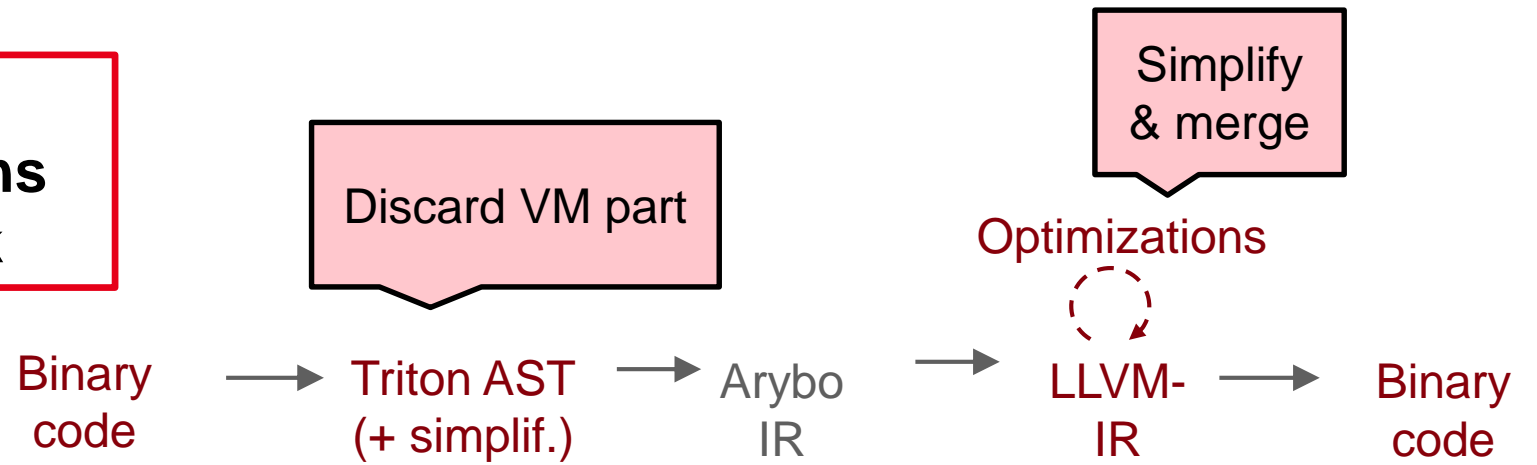
Symbolic reasoning a priori well adapted

- **Normalization / rewrite rules: $(a+b-a) \rightarrow b$**
- **Solver-based proof: $\text{solve}(a+b-a \neq b)$**



Goal

- Small protected hash functions
- Get the original function back



TIGRESS Challenge

- 7 (classes of) challenges
- 5 codes per class
- Original codes: hash-like functions
- Focus on challenges 0-4
- Only challenge 1 was solved



	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				

Solve challenges 0 - 4 (25 samples)

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

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



	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				

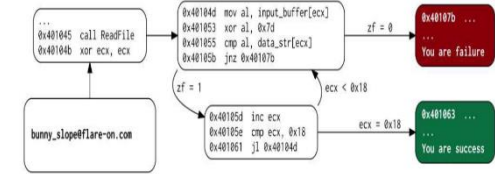
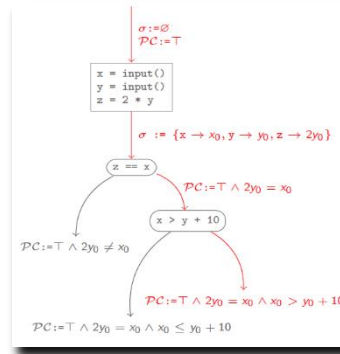
- **Opcode duplicate: merged!**
- **2-level VM (challenge 4): still ok**
- **Also tested vs each VM-option**

Challenge Description

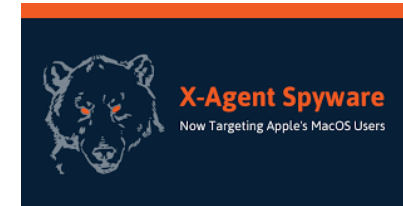
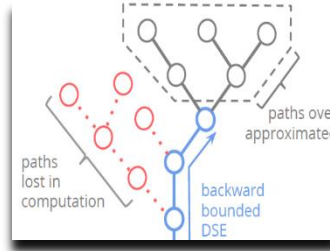
0000	One level of virtualization, random dispatch.
0001	One level of virtualization, superoperators, split instruction handlers.
0002	One level of virtualization, bogus functions, implicit flow.
0003	One level of virtualization, instruction handlers obfuscated with arithmetic encoding, virtualized function is split and the split parts merged.
0004	Two levels of virtualization, implicit flow.
0005	One level of virtualization, one level of jitting, implicit flow.
0006	Two levels of jitting, implicit flow.

REMINDER: SYMBOLIC DEOBFUSCATION

- EXPLORE



- PROVE

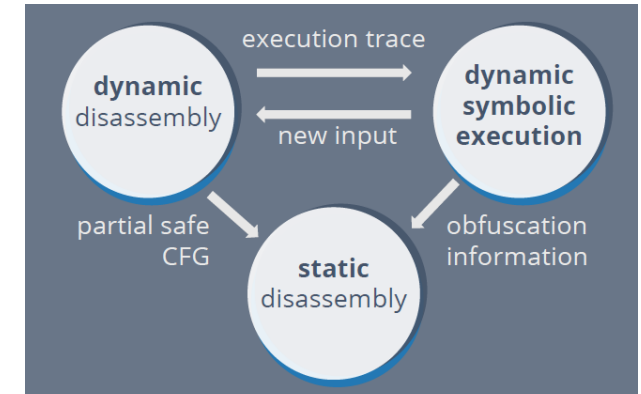
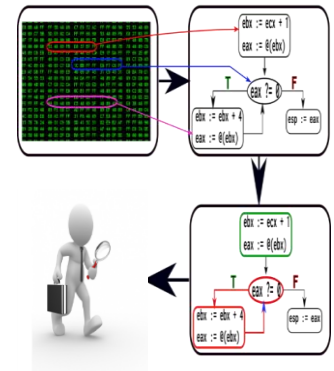


- SIMPLIFY



- **Standard limits of DSE**
 - #paths, limits of solvers (float), ...
- **Anti-DSE proposal are blooming**
 - Hard-to-solve predicates
 - Path splitting
 - Side-channels
 - Attacks all parts of the tool (solving, dynamic, taint, decoding, etc.)
 - ...
 - Note: 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?

- **A tour on the advantages of symbolic methods for deobfuscation**
- **Semantic analysis complement existing approaches**
 - Well-adapted – semantics is invariant by obfuscation
 - Explore, prove infeasible, simplify
 - Promising case-studies
- **Next Steps**
 - Anti-anti-DSE
 - Open the way to fruitful combinations (attack & defense)
- **Formal methods can be useful for binary-level security**
 - Yet, must be adapted: need robustness and scalability!

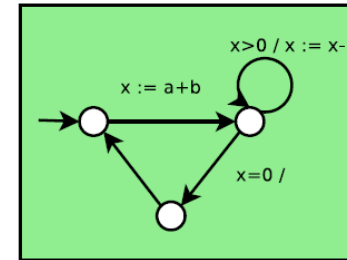


Commissariat à l'énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr

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

- Code-data confusion
- No specification (even implicit)
- Raw memory, low-level operations
- Code Size
- # Architectures

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
```