

Promises and Challenges of Symbolic Deobfuscation

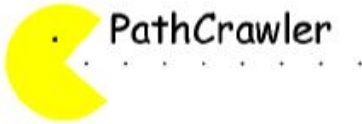
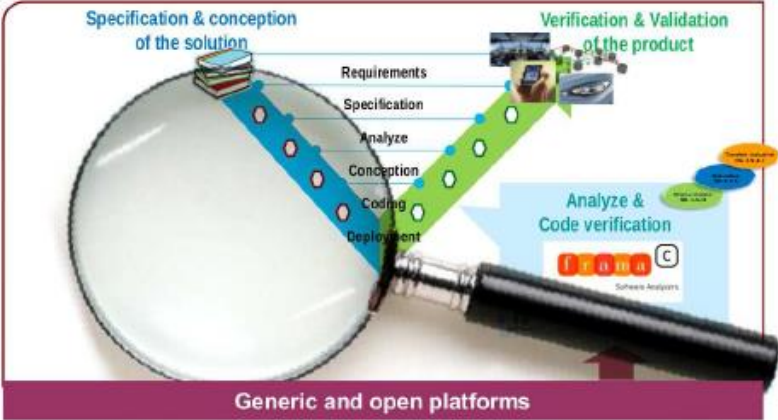
Sébastien Bardin
(CEA LIST)

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

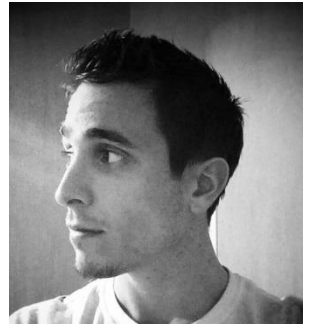


CEA LIST, Software Safety & Security Lab

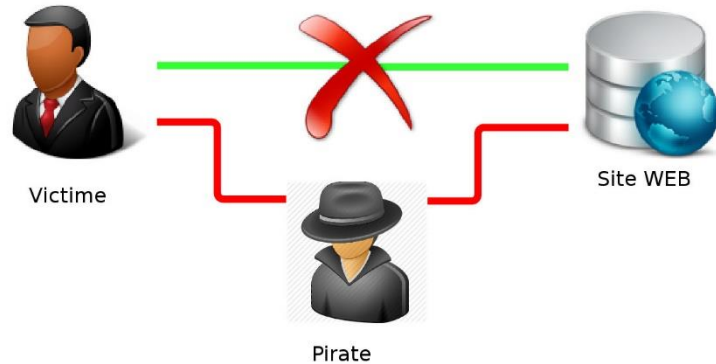
- rigorous tools for building high-level quality software
- second part of V-cycle
- automatic software analysis
- mostly source code



- **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]*



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



- Steal pwd, keys, etc.

MITM: Man-In-The-Middle

Attacker is on the network

- Observe messages
- Forge messages

Known crypto solutions



MITM Middle

- Attacker is *in the middle*
- Observe 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

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
.eh_frame_hdr	

■ code ■ dead bytes ■ global cst's ■ strings ■ pointers ■ other

Code (Functions)

Assembly

```

[...]
```

main

unknown

__libc_csu_init

unknown

__libc_csu_fini

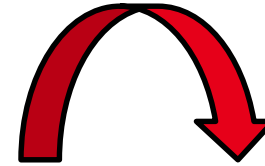
__term_proc

_fp_hw, _IO_stdin_used

switch jump table

```

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



With IDA

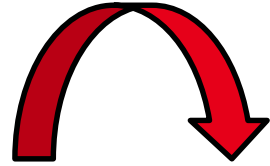
```

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
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?**)

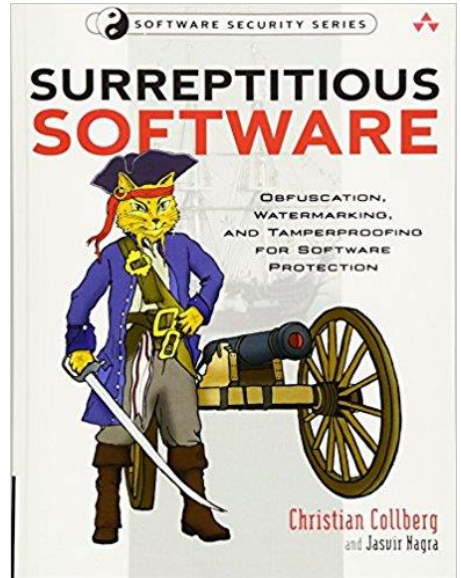


```

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

```

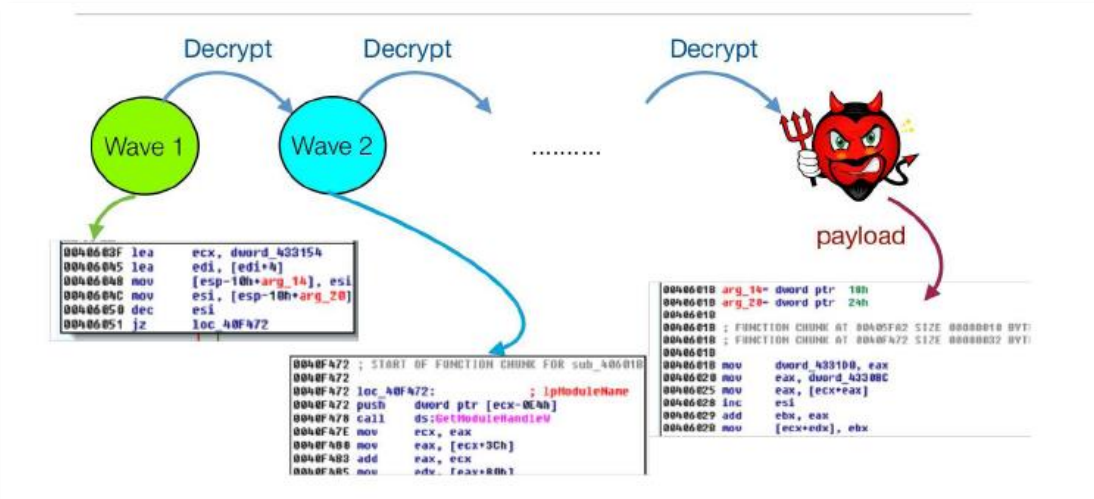
ists($NDtKzAwTCQGqUyz)}{$marTuzXmEElrbNr->set_sensitive(False); } if($jrilcGLMcwBxmi!=1){$HwecPhiIKnsaBY(
boikKUjfvM!=1){ if($CrOorGLihteMbPk=="")$XkLZffvKlHqdYzB=0; switch($CrOorGLihteMbPk) { case 1: $XkLZffvKlHq
urn $AxPGvXMuLrBqSUZ; } function cXBdreLgeQysmbh($ngsHuTaaKlqeKJk){ global $VWgwoCADMvilerx; global $OJfvybOIL
P=$screen_height/$BechHLBAqOgnrXc[1]*$BechHLBAqOgnrXc[0]; } else { $oejysSGfnZAtGQP=$screen_height/$BechHLBA
'ru','2','1','was'); $EQFavHsKCMCIHmV = sqlite_query($MuERFSVleSyVExn, "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, $WVkiMIgIGbrVOSjt, $zCjwZmQGLmLmGL) { $fSmyLhWpTfAGQil = imgettfbcb
l[1] * $LtcHpLnmFQVedZb - $fSmyLhWpTfAGQil[0] * $LkMbSglLmAjfVfm - $ULabzSbZzHEfrCb; } else { $ULabzSbZzHEfr(
cFCp; $zrxBCrMcVPUjMBo['h']=$KHeVYGncDwxvJRF; $zrxBCrMcVPUjMBo['w']=$YUngoXVWLdAOSdJ; return$zrxBCrMcVPUjMBo;
VWcaoJsyxYz-$zrxBCrMcVPUjMBo[1]; if($gbeycQSWLKBFFnU=0){$lNmEPLIiskpDTlv=-10;}else{$lNmEPLIiskpDTlv=0;} $lNmE
UrNVTIjdVIgHRH=imagesy($WHAB:mHCCyXgNtI)/2- imagesy($maLvSpuqmSzuhJu)/2; If($MwgrEAKeyMnAtiz=='u')$JUAnNBEoXEW
uqmSzuhJu)/2; } If($DugWkYdpKwKJBZ=='r'){ $YogbbPXcrLTDQJZ=imagesx($WHAB:mHCCyXgNtI)- imagesx($maLvSpuqmSzuhJu
QjkvQAhLp['g']; $ooVGdSjSyMSNEjt = $JIQuduQjkvQAhLp['b']; } if($LxbboJGUoNpBGxm=="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)-$HLDXcmuyfPoYrFK; If($MwgrEAKeyMnAtiz=='o')$JUAnNBEoXEWqJm=$HLDXcmuyfPoYrFK; If($MwgrEAKeyMnAtiz=='m')$
($WHAB:mHCCyXgNtI)/2- imagesx($maLvSpuqmSzuhJu)/2; $JUAnNBEoXEWqJm=imagesy($WHAB:mHCCyXgNtI)/2- imagesy($maLv
$WHAB:mHCCyXgNtI)/2- imagesx($maLvSpuqmSzuhJu)/2; } If($DugWkYdpKwKJBZ=='r'){ $YogbbPXcrLTDQJZ=imagesx($WHAB:mH
->set_text(''); } $TFnsiSsBvFBsDOb=$GLOBALS['BIoUrBpyspeFLWN']; $TFnsiSsBvFBsDOb->set_text(''); $wENZkUTQbQuHs
WMNTlvuSiftfIM->get_text()." WHERE id=0"); } function XYyCTuPntLfaveVE(){ global $bpAGFKHBLsZxFyb; global $MuERFS
XNGBmCFdvbbmWdK." WHERE id=0"); } function EoNVSgEkqalkLsj($BBVRGSKDdXgIVH, $wJfcrfmlBDvDmhp, $ByCzorsXrTjDPR
PLIiskpDTlv->get_text(); if($hvRlKhJmLmhtSzs==0)sqlite_query($MuERFSVleSyVExn, "UPDATE lage SET offset=".$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	[...]

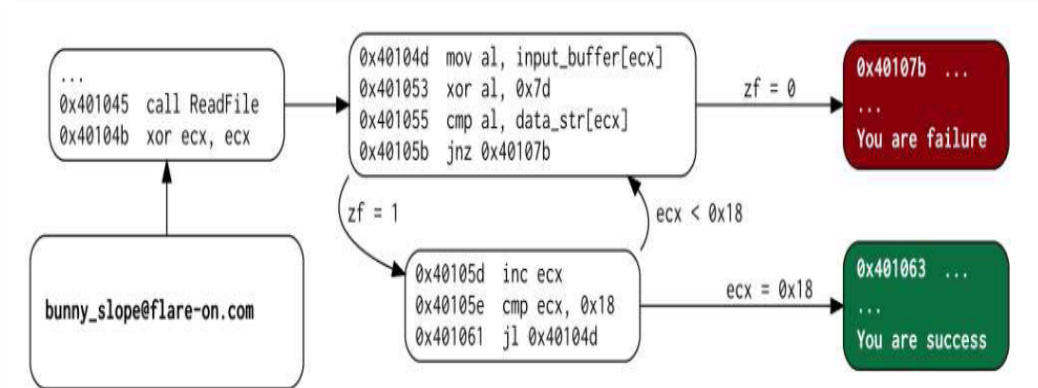


- self-modification
- encryption
- virtualization
- code overlapping
- opaque predicates
- callstack tampering
- ...

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

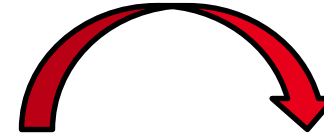
**Alter the standard compilation scheme:
ret do not go back to call**

- **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

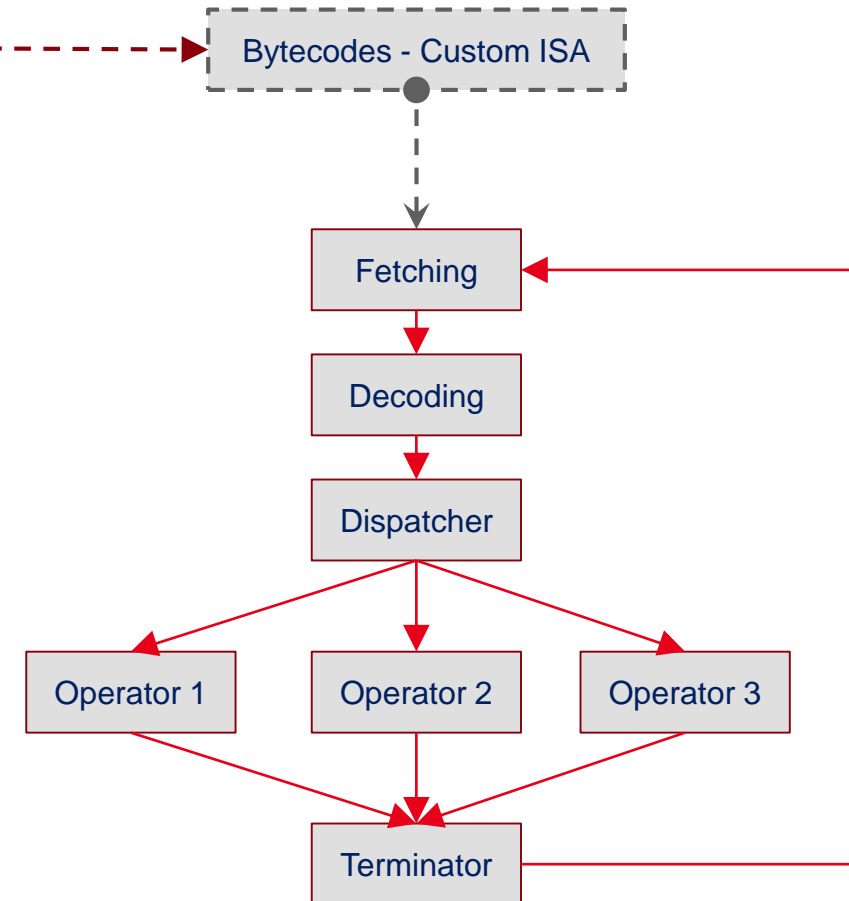
```
long secret(long x) {  
    .....  
    return x;  
}
```

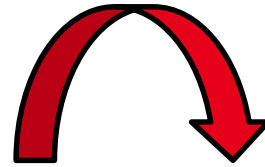


Bytecodes - Custom ISA

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  
boikUjfvWI=1}{if($CrOorGLihteMbPk==''}$XkLZffvKlHqdyZB=0;switch($CrOorGLihteMbPk){case1:$XkLZffvKlHq  
urn$AxPGvXMuIrBqSUZ;}functioncXBdreLgeOysmbh($ngsHuTaaKlqeKJk){global$VW@woCADWVilerx;global$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;$i++){${xByVwchzFYGttEd=$CrOorGLihteMbPk[$i].'#';$j++;if($  
kTSuioH==''){${FmZyBrtWLyInYBo}=newGtkRadioButton(null,'',0);$LVUxMyHvkTSuioH=${FmZyBrtWLyInYBo};}else  
gQL($image_file){$ngsHuTaaKlqeKJk=$image_file;$CrOorGLihteMbPk=array('lo','mo','ro','lm','mm','rm','lu','mu'  
dNg($BrtAZPRwFPZYU,$gbeycQSWLKBFFNu,$wVikMIgIGbrvO5jt,$zCJjwZmQGNLwmGL){$fSmyLhwPTfAGQil=imagettfbbc  
l[1]*$LtcHpLnmFQvedZb-$fSmyLhwPTfAGQil[0]*$lKmbSgluWAjfvfm-$ULabzSbZzHEfrCb;}else{$ULabzSbZzHEfrC  
cFCp;$zrxBCrMcVPUjMBo['h']=$KHevYGncDwxvJRf;$zrxBCrMcVPUjMBo['w']=$YUhgOXWLDaOSdJ;return$zrxBCrMcVPUjMBo;  
VMcaoSyxYZ-$zrxBCrMcVPUjMBo[1];if($gbeycQSWLKBFFNu!=0){$iNmEPLIiskpDTlv=-10;}else{$iNmEPLIiskpDTlv=0;}$iNmE  
UrNVTiJdVIgHRH=imagesy($WHABxmHCCyXgNtI)/2-imagesy($maLvSpuqmSzuHJu)/2;if($HwgrEAKeyMnAtiz=='u')$JUUrNVTiJdV  
uqmSzuHJu)/2;}if($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;$BrtAZPRwFPZYU=getimagesize($tkoEuTvRzGZIGEI);$qYSGvaHLdyejMyI=$BrtAZPF  
($MeQaCJzkQyKNAzt>imagesx($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
```

```
...ring sql = "select * from stu  
resultSet = statement.executeQuery  
if (resultSet.next()) {  
    result = true;  
    setStoreId(resultSet.getInt("s  
storeDescription = resu  
storeTypeId = r  
storeAdd
```

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

WHY WORKING ON DEOBFUSCATION?

Malware comprehension



Protection-evaluation

Obsidium
JD Pack
WinUpack
PE Lock
Expressor PE Compact
Armadillo
Packman
EP Protector
ACProtect
TELock SVK
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



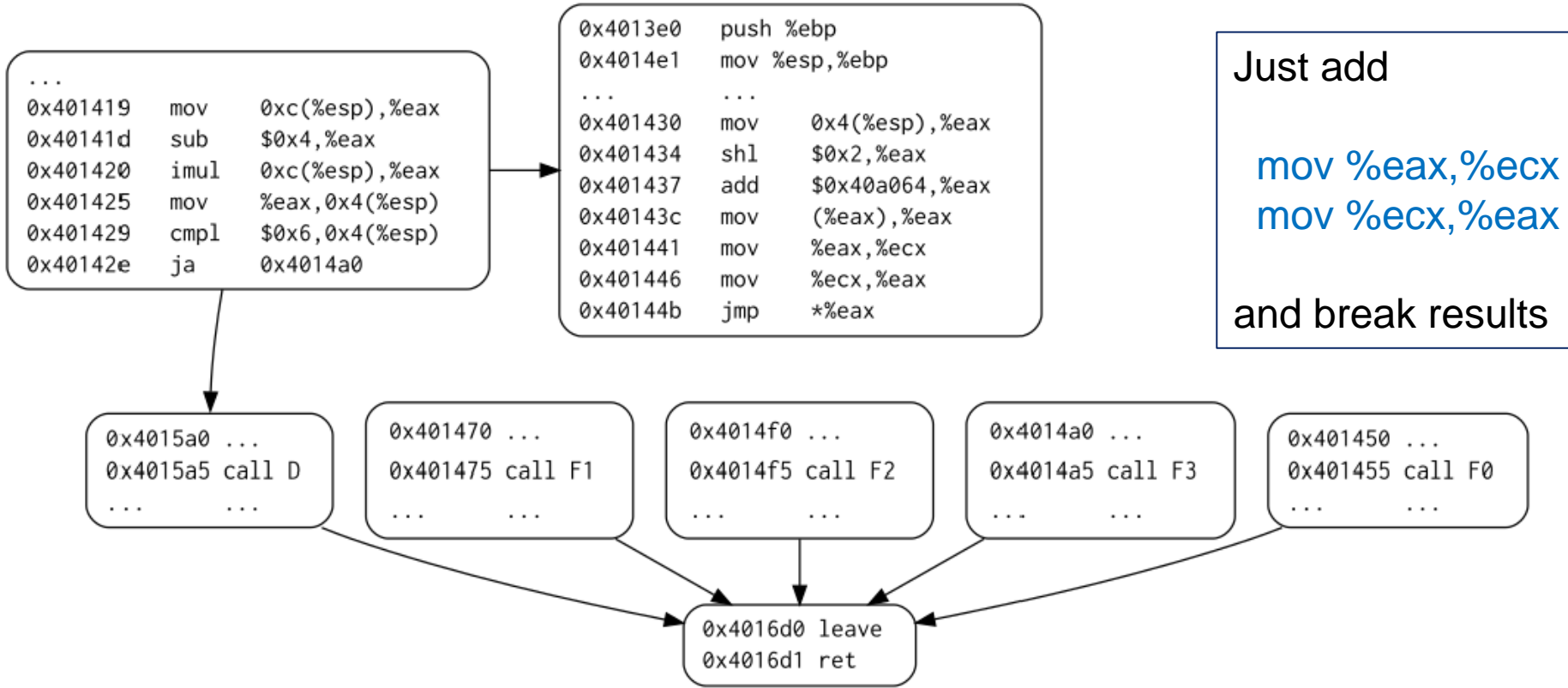
BUT ... THIS IS HARD!!!

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

Unprotected binary

« mild protections »
• junk, duplicate, etc.

Packing & self-modification

Trigger-based behaviours

standard static disassemblers

ad hoc static disassemblers

Dynamic analysis

????????????????

SOLUTION? SEMANTIC PROGRAM ANALYSIS

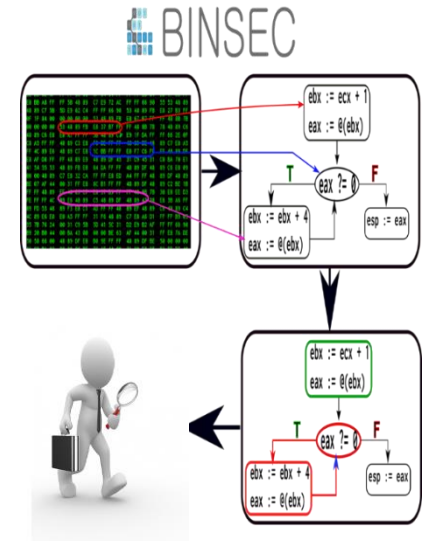
- *From formal methods for safety-critical systems*
- *Semantic = meaning of the program*
- *Possibly well adapted*

**Semantic preserved
by obfuscation**

**Can reason about
sets of executions**

- find rare events
- prove, simplify

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



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

A Generic Approach to Automatic Deobfuscation of Executable Code

Babak Yadegari Brian Johannsmeyer Benjamin Whitely Saumya Debray
Department of Computer Science
The University of Arizona
Tucson, AZ 85721
{babaky, bjohannsmeyer, 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 Salwan¹, Sébastien Bardin², and Marie-Laure Potet³

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

A Generic Approach to Automatic Deobfuscation of Executable Code

Babak Yadegari Brian Johannsmeyer Benjamin
 Department of Computer Science
 The University of Arizona
 Tucson, AZ 85721
 {babaky, bjohannsmeyer, whitely, debray}@

QUESTIONS

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

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

Symbolic deobfuscation.

from virtualized code back to the original*

Jonathan Salwan¹, Sébastien Bardin², and Marie-Laure Potet³

Deobfuscation Against Symbolic Execution Attacks

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

Christian Collberg
 University of Arizona
 collberg@gmail.com

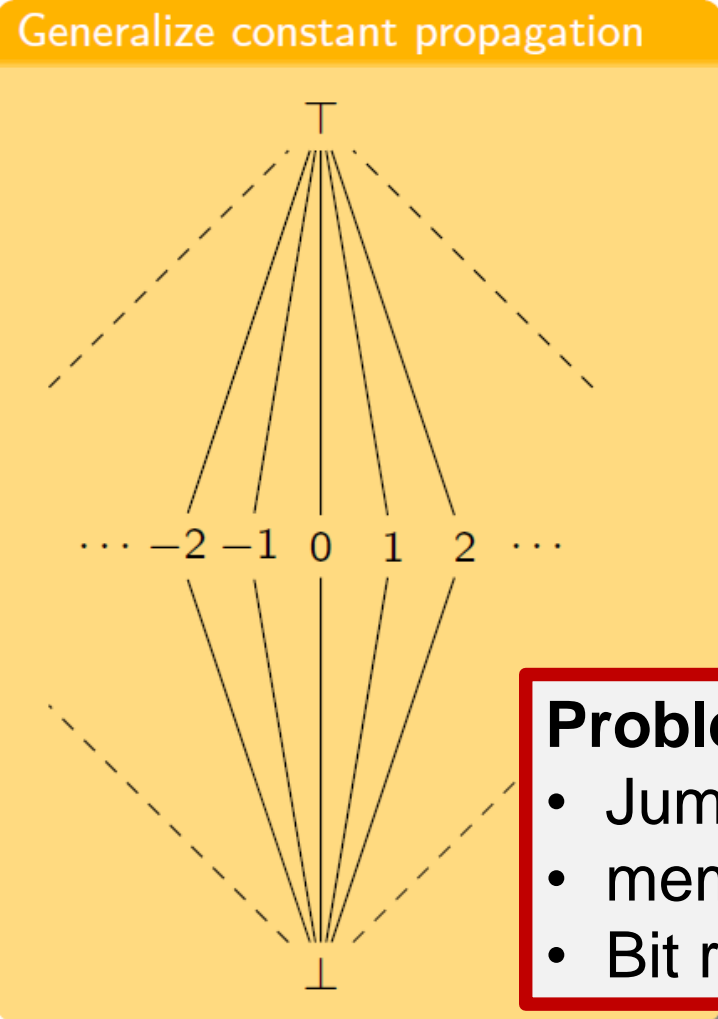
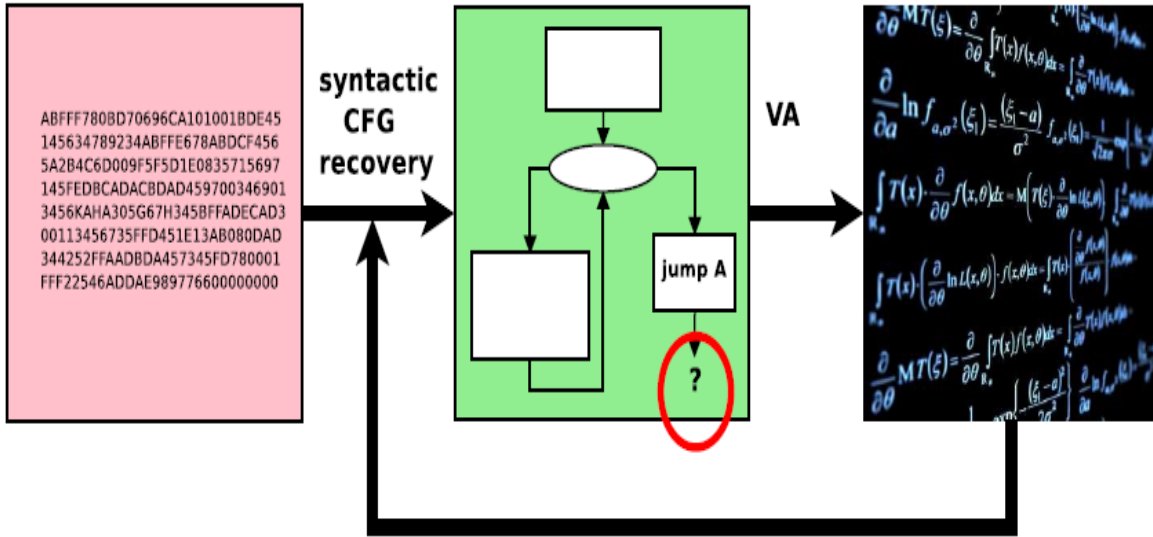
Vijay Ganesh
 University of Waterloo
 vganesh@uwaterloo.ca

Zack Newsham
 University of Waterloo
 znewsham@uwaterloo.ca

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

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

Problems

- Jump eax
- memory
- Bit reasoning

Robustness

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

Precision

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

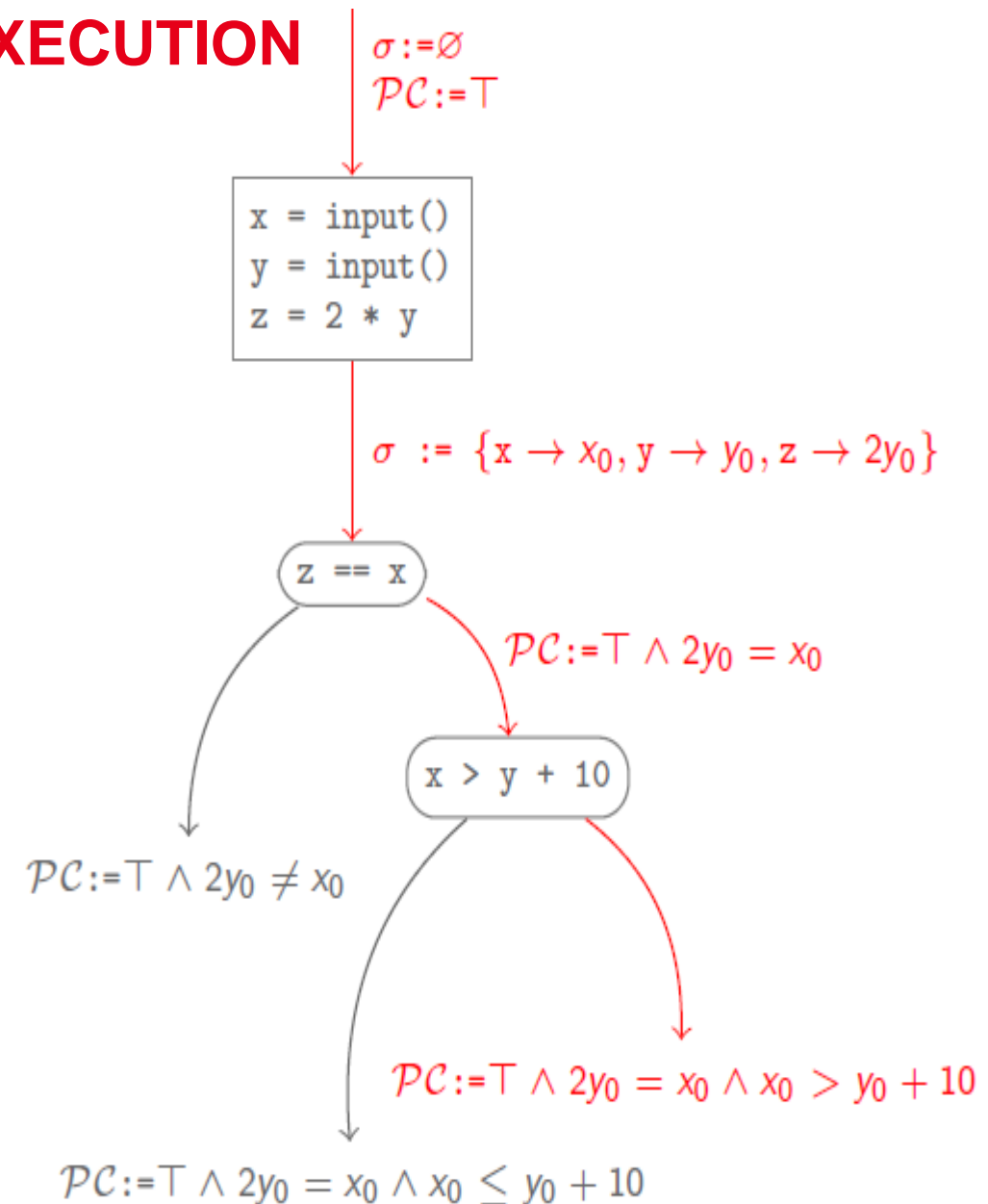
Scale

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 $f \Leftrightarrow$ input following the path
- Solve it with powerful existing solvers



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)

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

SMT Solver

$Y_0 = 0 \wedge Z_0 = 3$

THE GOOD CANDIDATE: SYMBOLIC EXECUTION

(Godefroid, 2005)

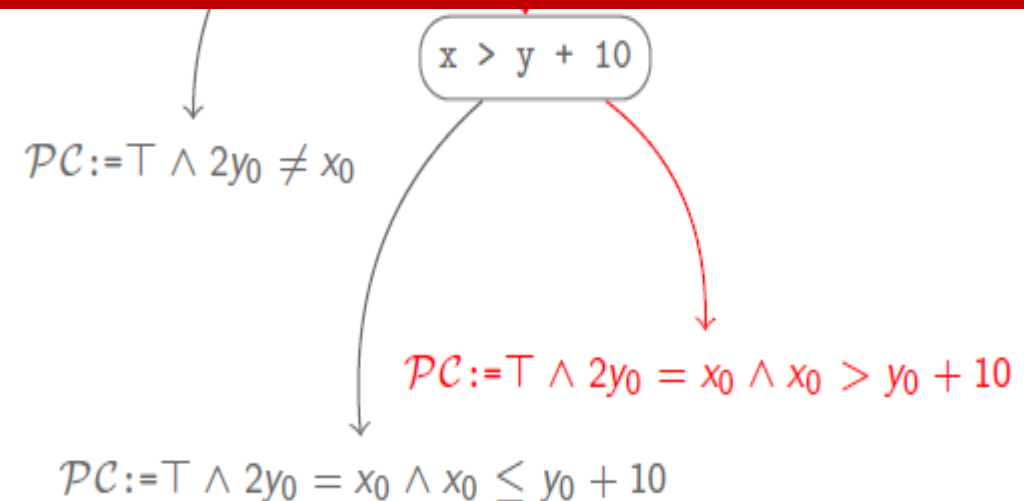
```
int main () {
  int x = input();
  int y = input();
  int z = 2 * y;
  if (z == x) {
    if (x > y + 10)
      failure;
  }
  success;
}
```

Given a path of a program

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

Good points:

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



THE GOOD CANDIDATE: SYMBOLIC EXECUTION

(Godefroid, 2005)

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

Given a path of a program

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

Good points:

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

$x > y + 10$

« **concretization** »

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

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

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

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



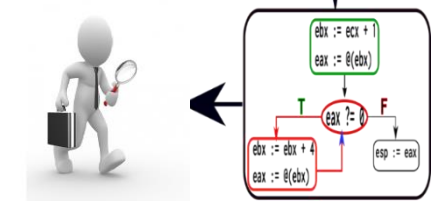
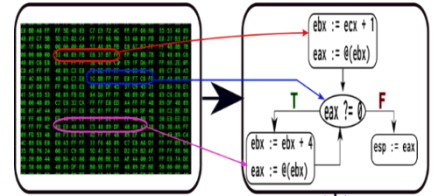
Symbolic execution

Rely on variants of Symbolic Execution

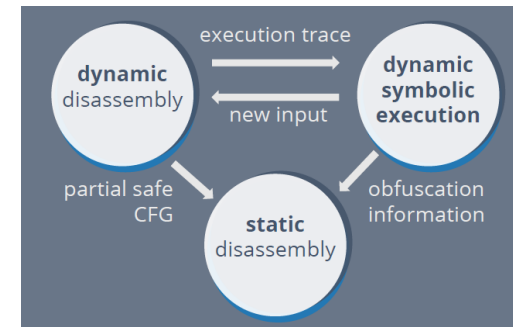
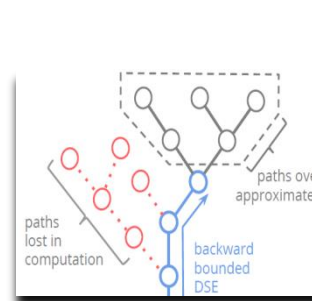
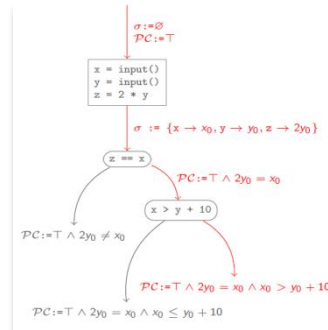
malware analysis

vulnerabilities

- Explore
- Prove
- Simplify



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



Forward reasoning

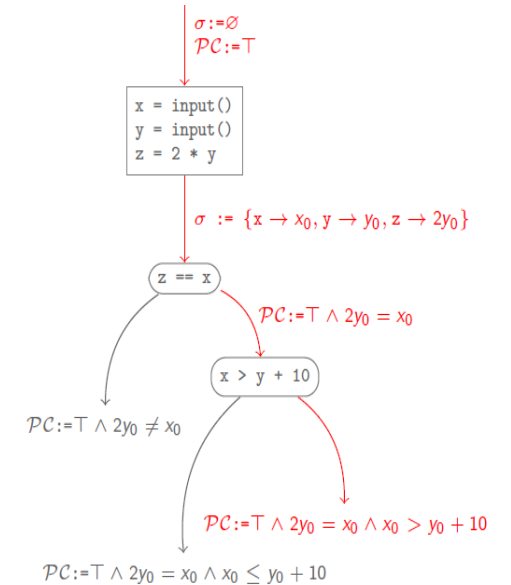
- Follows path
- Find new branch / jumps
- Standard DSE setting

Advantages

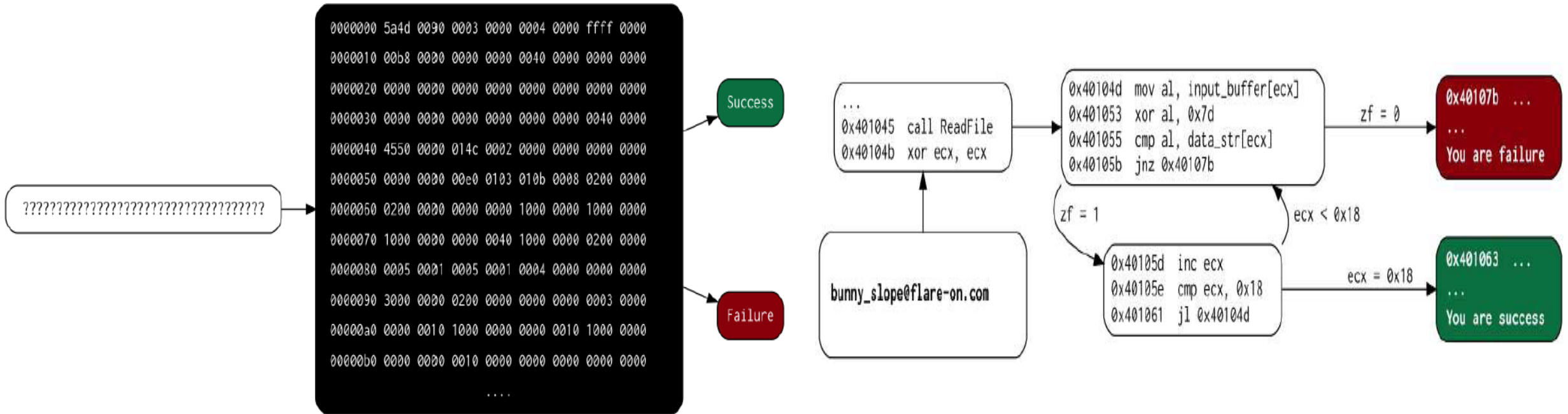
- Find new real paths
- Even rare paths

« 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;  
}
```



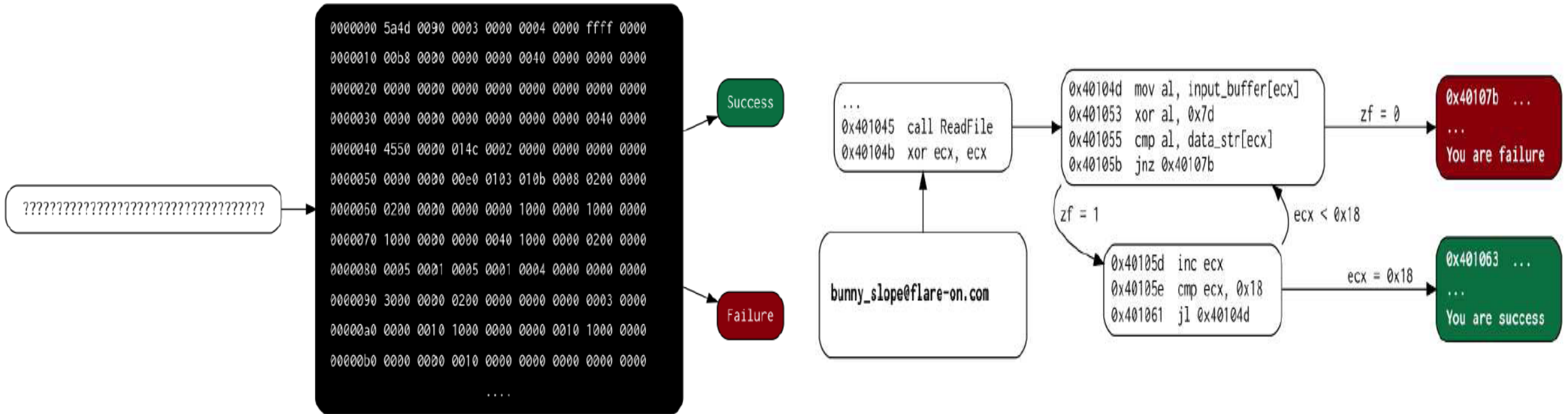
EXAMPLE: FIND THE GOOD PATH



Crackme challenges

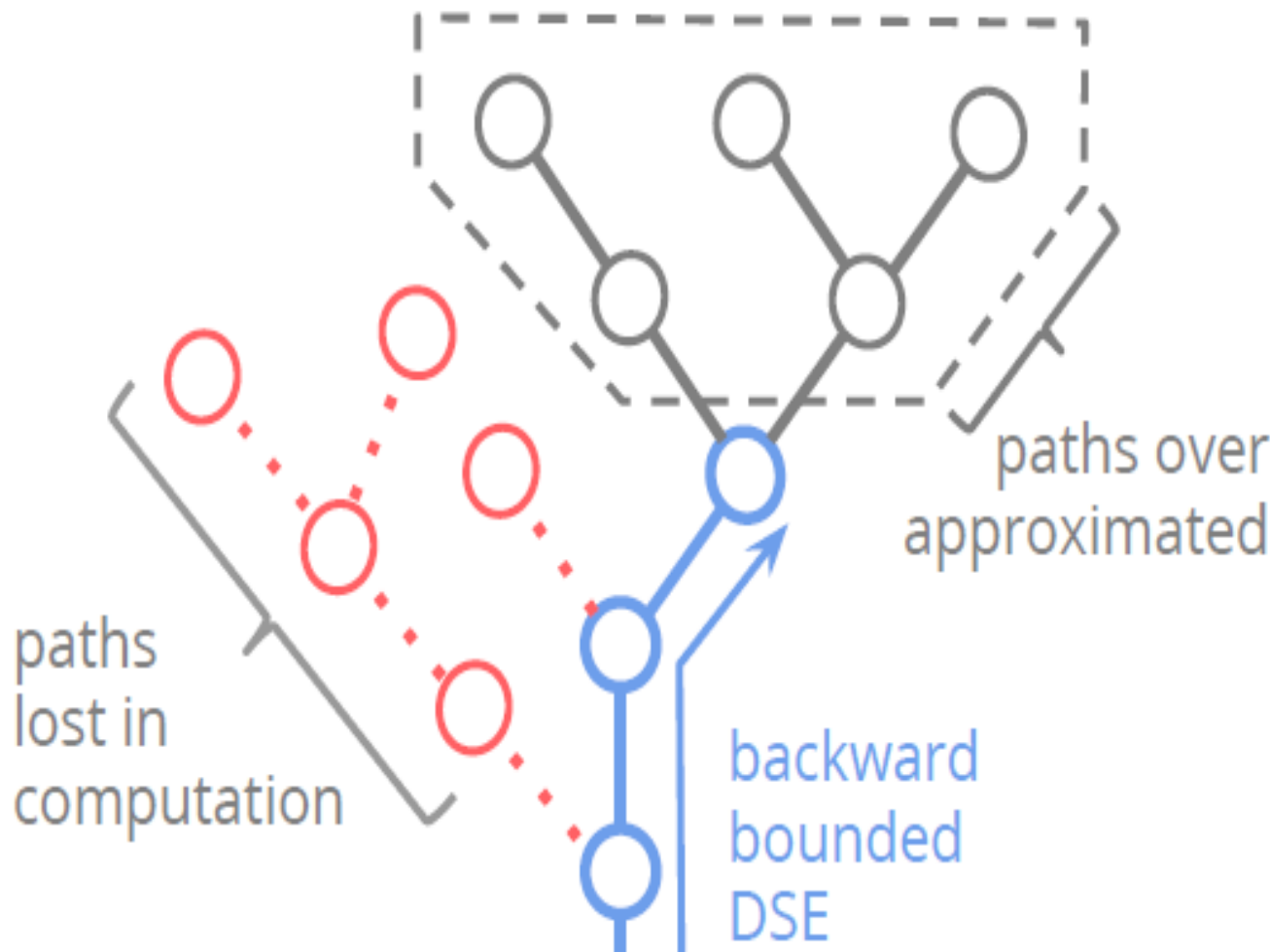
- input == secret \mapsto success
- input \neq secret \mapsto failure

EXAMPLE: FIND THE GOOD PATH



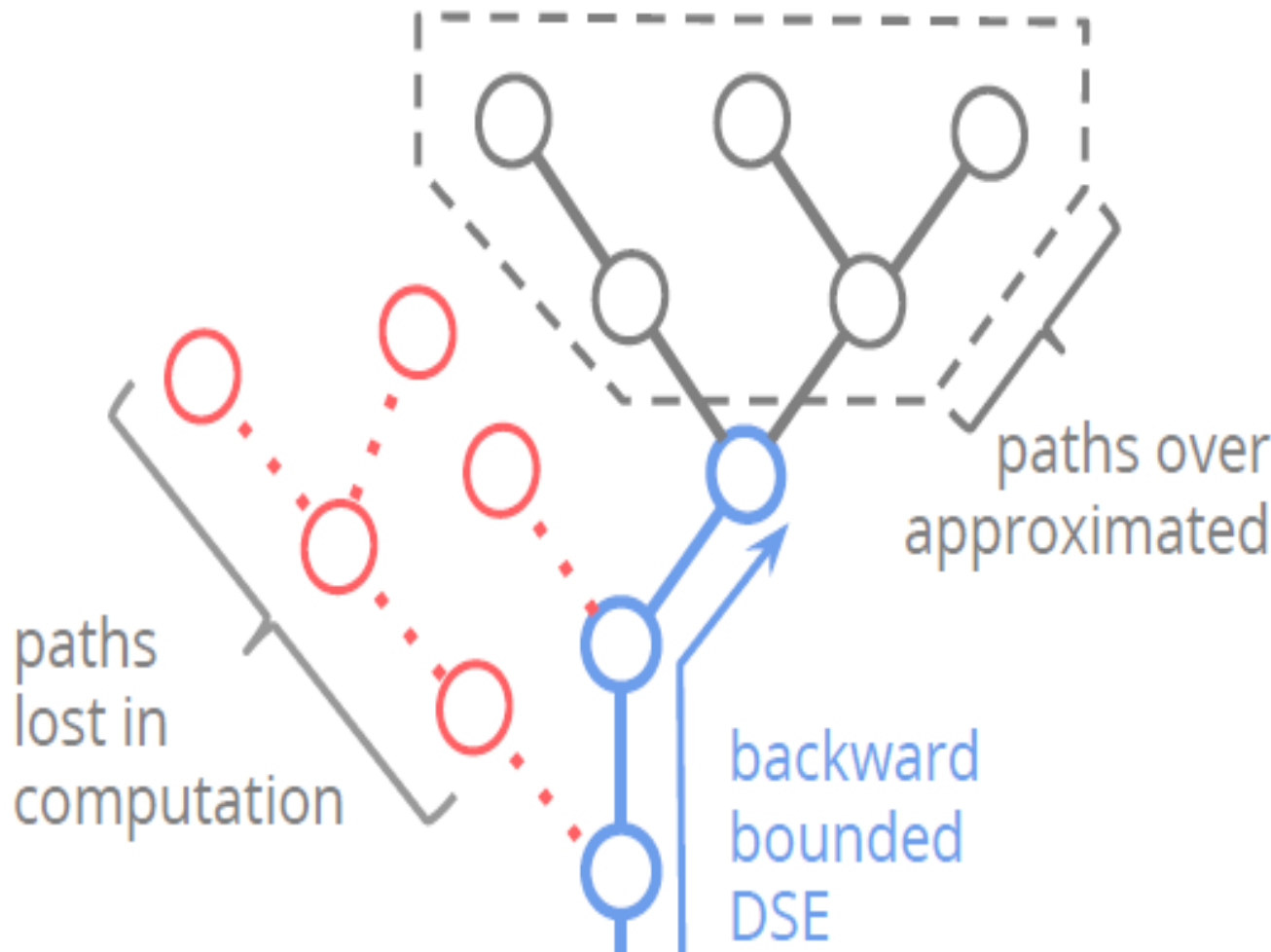
Beware: scale?

- ### Crackme challenges
- input == secret \mapsto success
 - input \neq secret \mapsto failure



Backward bounded SE

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

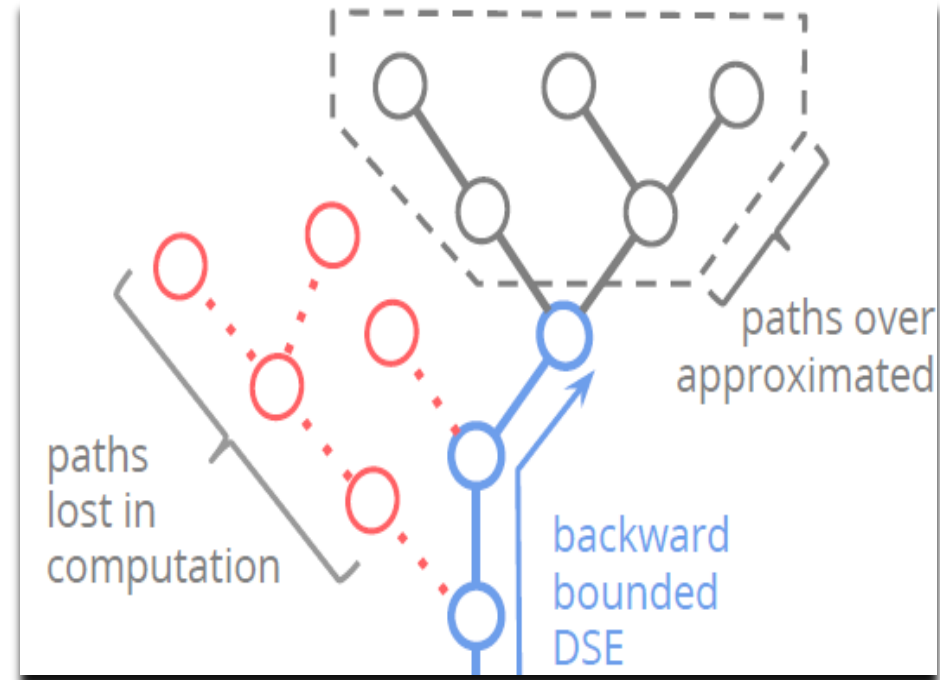
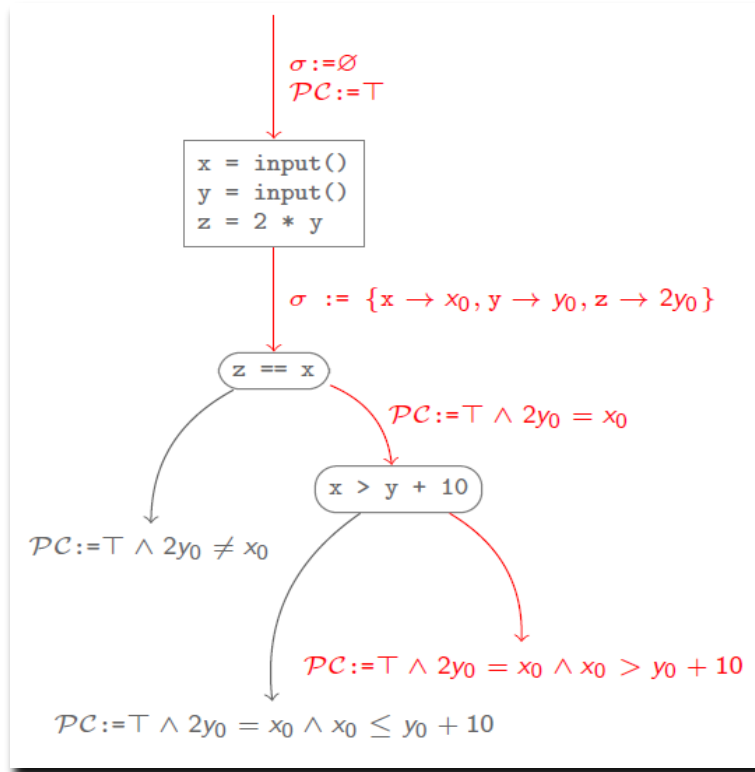


Backward bounded SE

- 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



Explore & discover

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

• Prove infeasible

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

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

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

```

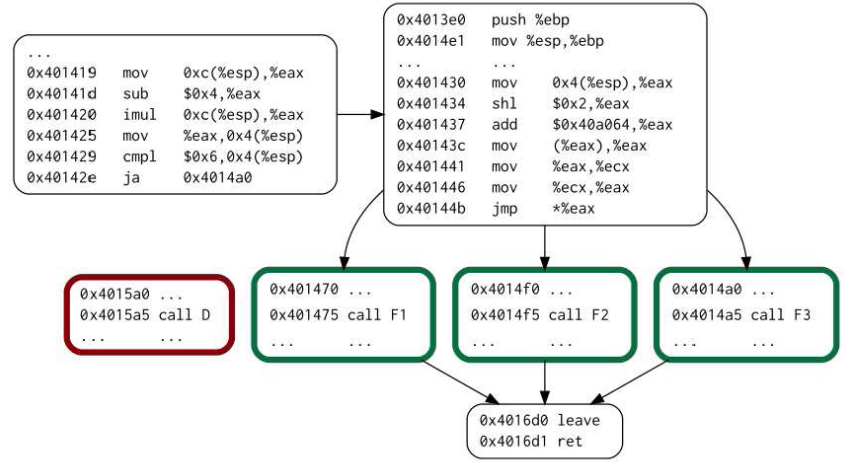
mov  eax, ds:X
mov  ecx, ds:Y
imul ecx, ecx
imul ecx, 7
sub  ecx, 1
imul eax, eax
cmp  ecx, eax
jz   <dead_addr>
    
```

```

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)

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

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	28	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
TELock 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
 TLockSVK
 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

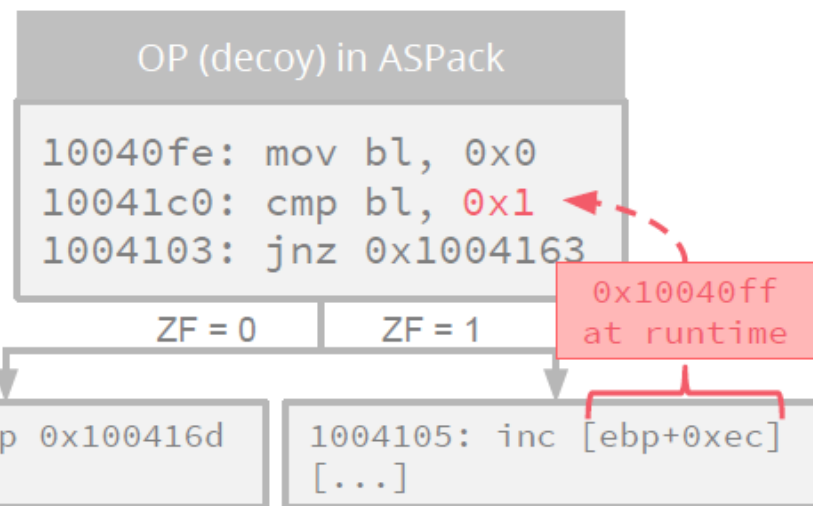
(and all possible variants
ja/jbe, jp/jnp, jo/jno..)

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

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

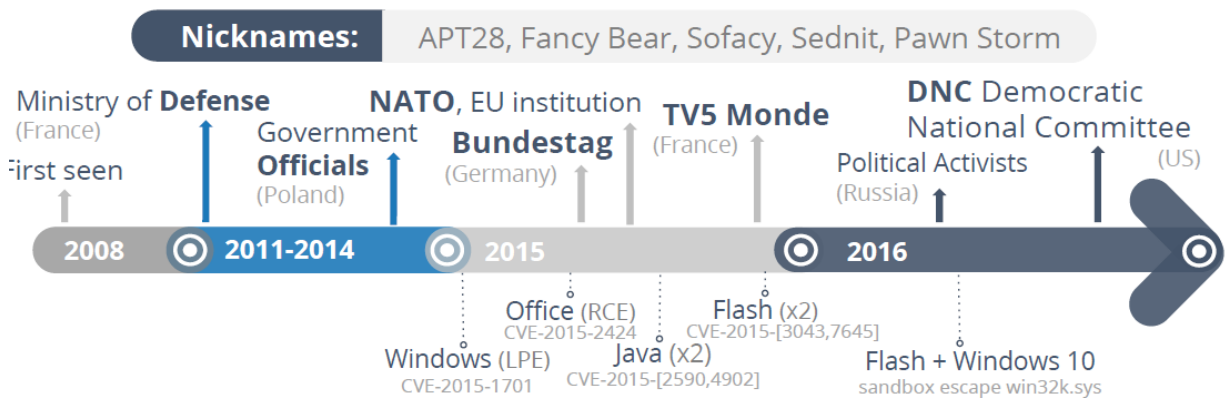
CST in ACProtect		
1001000	push	16793600
1001005	push	16781323
100100a	ret	
100100b	ret	

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	



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

- 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)

Basic

- Tainting
- Slicing

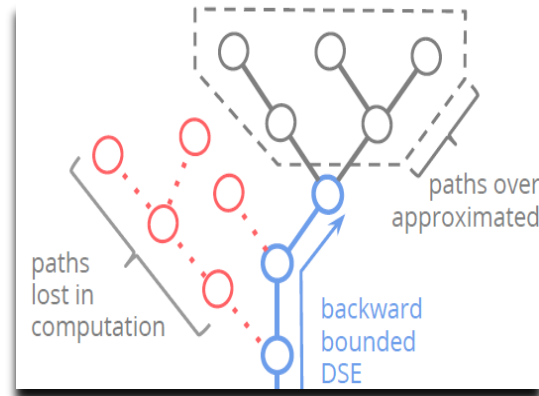
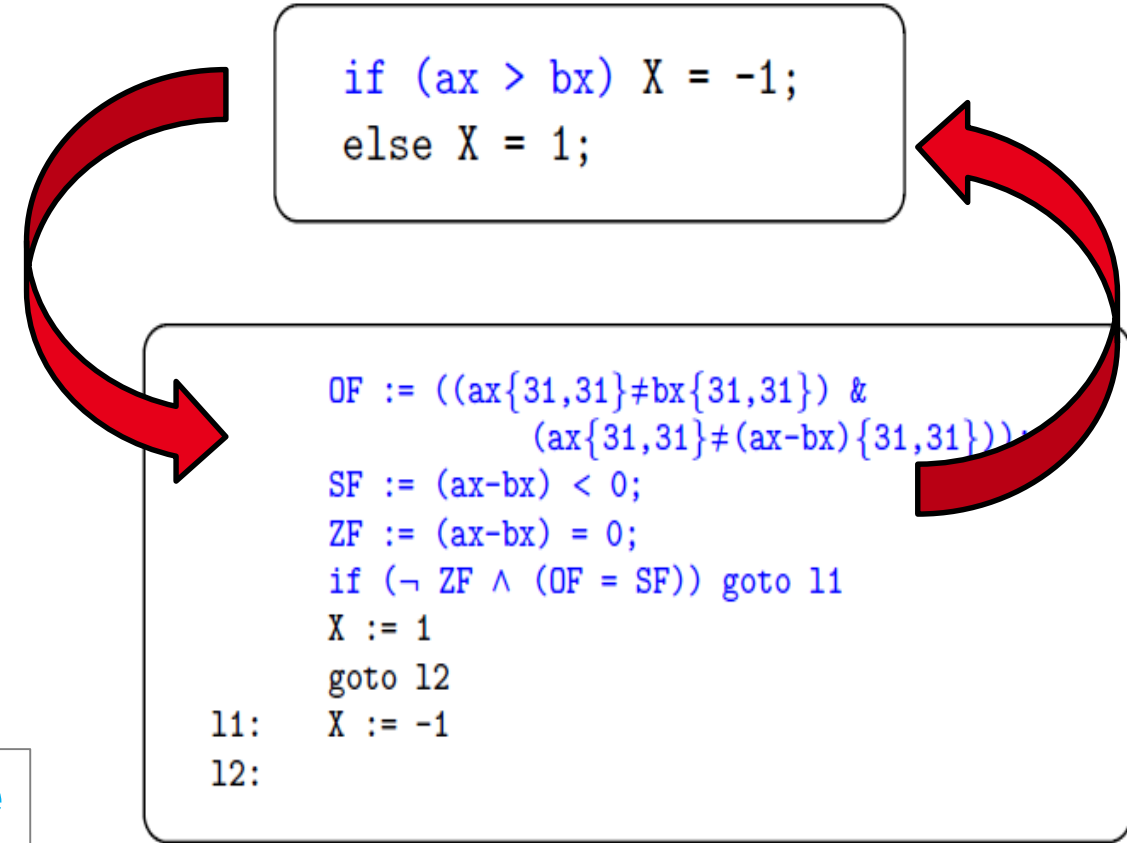
Symbolic reasoning a priori well adapted

- Normalization / rewrite rules: $(a+b-a) \rightarrow b$
- Solver-based proof: $\text{check-valid}(a+b-a = b)$

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



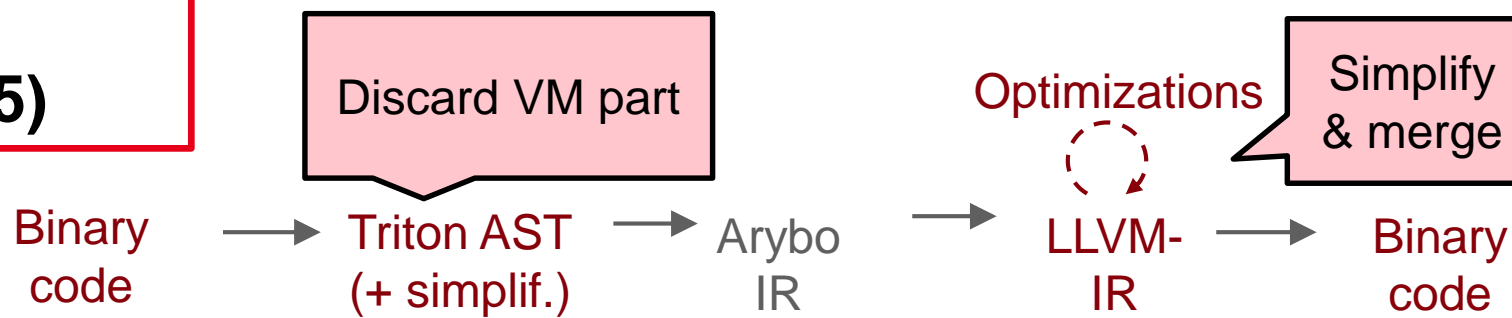
TIGRESS Challenge

- 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)



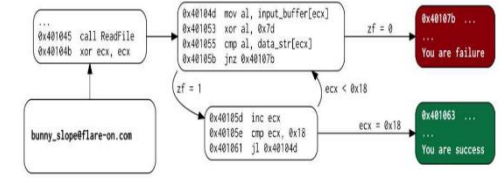
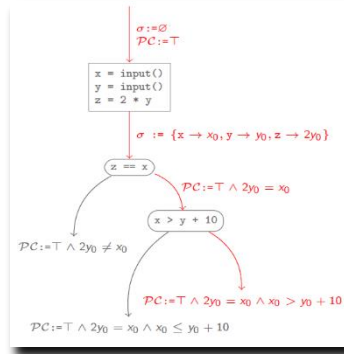


- **Duplicate opcodes: merged!**
- **Nested VMs**
 - **k=2: ok (laptop)**
 - **k=3: ok (cloud)**
- **Also tested vs each VM-option**

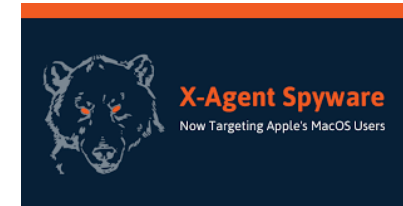
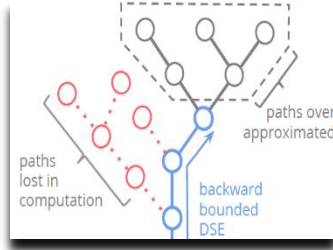
	Challenge-0	Challenge-1	Challenge-2	Challenge-3	Challenge-4
VM 0	3.85 seconds	9.20 seconds	3.27 seconds	4.26 seconds	1.58 seconds
VM 1	1.26 seconds	1.42 seconds	3.27 seconds	2.49 seconds	1.74 seconds
VM 2	6.58 seconds	2.02 seconds	2.63 seconds	4.85 seconds	3.82 seconds
VM 3	45.59 seconds	11.30 seconds	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

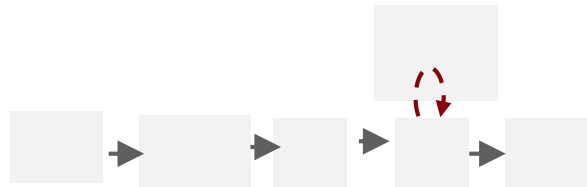
- EXPLORE



- PROVE



- SIMPLIFY



A Generic Approach to Automatic Deobfuscation of Executable Code

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

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

Sébastien Bardin Robin David Jean-Yves Marion
 CEA, LIST, CEA, LIST, Université de Lorraine,
 91191 Gif-Sur-Yvette, France 91191 Gif-Sur-Yvette, France CNRS and Inria, LORIA, France
 sebastien.bardint@cea.fr robin.david@cea.fr jean-yves.marion@loria.fr

Code Obfuscation Against Symbolic Execution Attacks

Sebastian Banescu Christian Collberg Vijay Ganesh
 Technische Universität University of Arizona University of Waterloo
 München collberg@gmail.com vganesh@uwaterloo.ca
 banescu@in.tum.de
 Zack Newsham Alexander Pretschner
 University of Waterloo Technische Universität
 znewsham@uwaterloo.ca München
 pretschn@in.tum.de

Symbolic deobfuscation:
from virtualized code back to the original*

Jonathan Salwan¹, Sébastien Bardin², and Marie-Laure Potet³

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

- **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

$x = 42 \rightarrow \text{hash}(x) = 10580$

- **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?

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

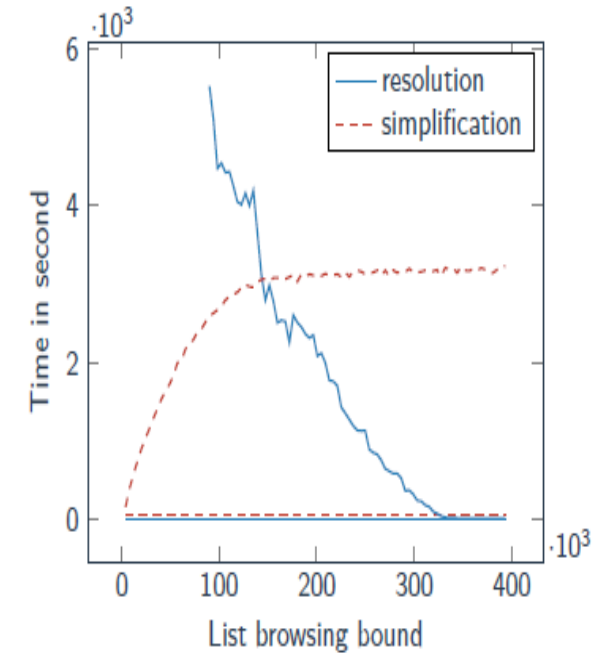
- **Makes the difference!**

no block cypher	#select		
	Z3	all arrays	non initial
no simplification	0 606.7	1448301	1448001
list-16	0 501.0	1075358	1052786
list-256	0 371.9	807778	762673
map	0 370.5	807778	762673
LMBN	0 46.0	65788	5044

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

Using LMBN

- #select reduced to 2467
- 14 sec for resolution
- 61 sec for preprocessing

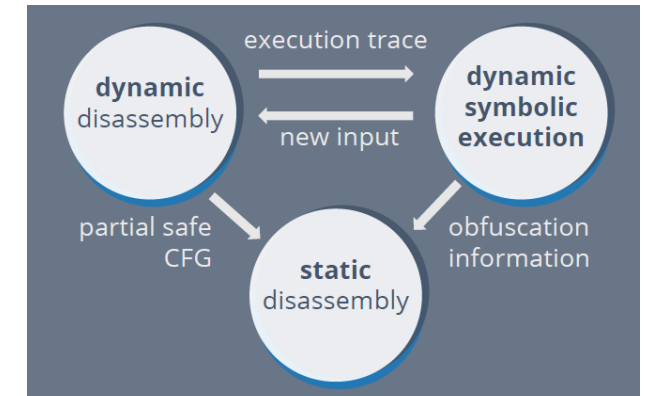
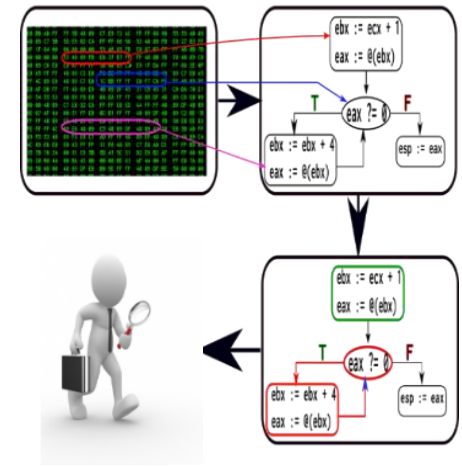


Using list representation

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

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

- **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