

# BINSEC:

## Binary-level Semantic Analysis to the Rescue

Sébastien Bardin joint work with Richard Bonichon, Robin David, Adel Djoudi, Benjamin Farinier, Josselin Feist, Laurent Mounier, Marie-Laure Potet, Thanh Dihn Ta, Franck Védrine

CEA LIST (Paris-Saclay, France)





# About the BINSEC project

#### A research project :

- funded by ANR (2013-2017)
- axis 1 (security) and 2 (software engineering)
- formal techniques for binary-level security analysis

Partners : CEA (coordinator), Airbus Group, INRIA Bretagne Atlantique, Université Grenoble Alpes, Université de Lorraine

People : Sébastien Bardin, Frédéric Besson, Sandrine Blazy, Guillaume Bonfante, Richard Bonichon, Robin David, Adel Djoudi, Benjamin Farinier, Josselin Feist, Colas Le Guernic, Jean-Yves Marion, Laurent Mounier, Marie-Laure Potet, Than Dihnh Ta, Franck Védrine, Pierre Wilke, Sara Zennou

Platform : CEA, Université Grenoble Alpes



#### Binary-level security analysis

- many applications, many challenges
- syntactic and dynamic methods are not sufficient

#### Semantic approaches can help!

- semantic exploration, semantic disassembly
- yet, still hard to design

#### The BINSEC Platform [CEA & Uni. Grenoble Alpes]

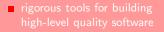
- open source, dual goal :
  - help design new binary-level analyzers (basic building blocks)
  - provide innovative analyzers
- current : multi-architecture support, semantic exploration & semantic disassembly, poc on vulnerability analysis and deobfuscation
- still young : beta-version just released [http ://binsec.gforge.inria.fr/]





# About my lab @CEA

### CEA LIST, Software Safety & Security Lab



- 2nd part of V-cycle
- automatic software analysis
- mostly source code





Software Analyzers













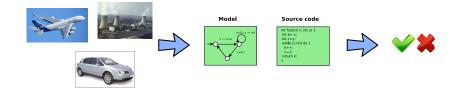
RMLL 2016: The Security Track

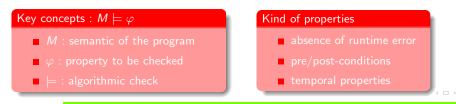




# About formal verification

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





# From (a logician's) dream to reality

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

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

#### Ex : Airbus

#### Verification of

- runtime errors [Astrée]
- functional correctness [Frama-C \*]
- numerical precision [Fluctuat \*]
- source-binary conformance [CompCert]
- ressource usage [Absint]
- \* : by CEA DILS/LSL



# From (a logician's) dream to reality

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

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

#### Ex : Microsoft

Verification of drivers [SDV]

- conformance to MS driver policy
- home developers
- and third-party developers



Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability. - Bill Gates (2002)



Benefits of binary-level analysis
Outline

### Preambule

## Benefits of binary-level analysis

- Challenges of binary-level analysis
- Semantic approaches
- BINSEC platform

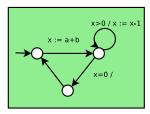
## Achievements

## Conclusion

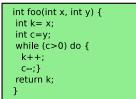
 ${}^{+} \Box \rightarrow$ 

### Benefits of binary-level analysis Binary-level software analysis

### Model



### Source code



### Assembly

_start:		
load	А	100

add B A cmp B 0 jle label

label:

move @100 B

### Executable

ABFFF780BD70696CA101001BDE45 145634789234ABFFE678ABDCF456 A2284C6D009F5F5D1E0835715697 145FEDBCADACBDAD459700346901 3456KAHA305G67H345BFFADECAD3 00113456735FFD451E13AB080DAD 344252FFAADBDA457345FD780001 FFF22546ADDAE989776600000000



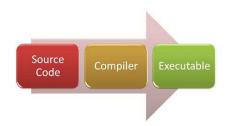
Benefits of binary-level analysis What for ? (1)



## How much do you trust your external components?



Benefits of binary-level analysis What for ?(2)



## How much do you trust your compiler?

 $\leftarrow$   $\square$   $\rightarrow$ 



Benefits of binary-level analysis What for? (2)

#### Security bug introduced by a non-buggy compiler

```
void getPassword(void) {
  char pwd [64];
  if (GetPassword(pwd,sizeof(pwd))) {
  /* checkpassword */
  }
  memset(pwd,0,sizeof(pwd));
}
```

- Optimizing compilers may remove dead code
- pwd never accessed after memset
- Thus can be safely removed
- And allows the password to stay longer in memory

Mentioned in OpenSSH CVE-2016-0777

 $\leftarrow \Box \rightarrow \cdot$ 



Benefits of binary-level analysis What for ? (3)



## Is it Stuxnet?



### Preambule

- Benefits of binary-level analysis
- Challenges of binary-level analysis
- Semantic approaches
- BINSEC platform

### Achievements

## Conclusion

 $< \square >$ 

### Several major security analyses are performed at byte-level

- vulnerability analysis [exploit finding]
- malware dissection and detection [deobfuscation]

### State-of-the-technique

- very skilled experts, many efforts and basic tools
- dynamic analysis : gdb, fuzzing [easy to miss behaviours]
- syntactic analysis : objdump, IDA Pro [easy to get fooled]





### Several major security analyses are performed at byte-level

- vulnerability analysis [exploit finding]
- malware dissection and detection [deobfuscation]

### State-of-the-technique

- very skilled experts, many efforts and basic tools
- dynamic analysis : gdb, fuzzing [easy to miss behaviours]
- syntactic analysis : objdump, IDA Pro [easy to get fooled]

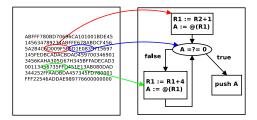




Challenge of binary-level analysis Challenge : correct disassembly

#### Input

- an executable code (array of bytes)
- an initial address
- $\blacksquare$  a basic decoder : file  $\times$  address  $\mapsto$  instruction  $\times$  size

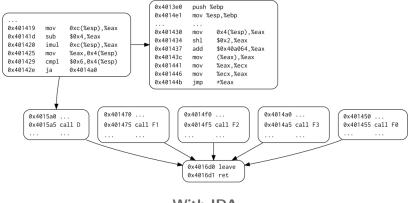


Output : (surapproximation of) the program Control-Flow Graph ■ problem : successors of jmp eax?

BINSEC team

### Challenges of binary-level analysis Limits of syntactic approaches

### Ex : IDA is fooled by simple syntactic tricks

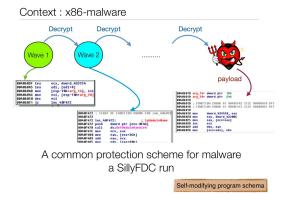


With IDA

### Challenges of binary-level analysis Even worse : obfuscated code

### Understand or recognize malware despite obfuscation

- self-modifying code, virtual machines
- opaque predicates, stack tampering, etc.





#### Use-after-free (UaF) - CWE-416

- dangling pointer on deallocated-then-reallocated memory
- may lead to arbitrary data/code read, write or execution
- standard vulnerability in C/C++ applications (e.g. web browsers) firefox (CVE-2014-1512), chrome (CVE-2014-1713)

```
1 char *login , *passwords;
login=(char *) malloc (...);
3 [...]
free(login); // login is now a dangling pointer
5 [...]
passwords=(char *) malloc (...); // may re-allocate memory of *login
7 [...]
printf("%s\n", login); // security threat : may print the passwords!
```



Challenges of binary-level analysis

## Limits of dynamic analysis

#### Find a needle in the heap!

 sequence of events, importance of aliasing

 strongly depend on implem of malloc and free

4800	0000	5dc3	5589	e5c7 081	2 8880	6658	4800	0000	5dc3	558
6666	6698	4500	0000	-		9820	6666	<b>66P8</b>	4500	000
		0660		Entry	point	540	bf0e	0821	0000	00b
			0822			5 9	e5c7	0540	<b>bf</b> 0e	082
5dc3	5589	e583	ec10	C703 888	8 4900	6600	5dc3	5589	e583	ec1
6666	a148	bf0e	0883	f809 48t	of 0e08	0100	0000	a148	bf0e	088
8604	8548	e10b	0811	eBc6 of	7 8892	8886	8604	8548	e10b	08f
00c6	4519	0000	4519	00c7 45	7 00c6	45f8	00c6	45f9	00c6	45f
8666	6669	9961	0000	c645 054	18 bf0e	0802	6666	66e9	d901	996
C045	1900	C045	1901	8070 f76	01 c645	f860	c645	f900	c645	fa6
4801	0000	0300	0000	887d Fb6	00 750a	c795	48bf	0e08	0300	000
1000	7508	C/05	4801	0e08 fb6 740f 090	00 7410	8974	fc00	750a	c705	485
1000	7415	-000	1000	7401 090	00 0000	807d	fc00	7415	807d	fb8
6701	6646	6900	6646	00e9 c76	05 486	8e98	6666	8666	e988	016
6-00	7406	-705	4014	1900 81	6000	c645	+701	c645	1800	c64
0100	0000	600	4000	6645 64	15 fa02	807d	fc00	740f	c705	48b
645	f0009	100	f 202	807d f70	00 0000	e95e	0100	66c9	5901	000
fe00	758.	C785	48hf	0e08 fde	01 c645	1800	c645	1900	c645	ta6
fc00	75	6785	4001	0e08 656	00 7410	807d	fe00	750a	c705	485
fe00	746	6785	495f	0e08 030	0 0000	807d	tc00	750a	c705	485
0100	-	961	8888	c645 866	00 0000	8019	1600	7401	c705	480
c645	free		6-01	887d f70	0 0000	6906	0100	6669	0901	000
48bf	U									
6666	c645									
									5dc3	558
										000
									se	00b
										082
									e583	ec1
0000	<b>a148</b>	bf0e	0883	f809 48b	f 0e08	0100	0000	a148	bf0e	088
9999	00e9	d901	6666	c645 054	8 bf0e	0802	6669	66e9	d901	000
48bf	0e08	0300	0000	807d Fb0	0 750a	c705	48bf	0e08	0300	000
1000	750a	C/05	48bf	BeB8 fbB	0 7410	807d	fc00	750a	c705	48b
TC00	/415	807d	TD00	740f 090	0 0000	807d	fc00	7415	807d	fb®
0000	0000	6388	0100	00e9 c70	5 48bf	0e08	8699	8666	e988	010



### Preambule

- Benefits of binary-level analysis
- Challenges of binary-level analysis
- Semantic approaches
- BINSEC platform

### Achievements

## Conclusion

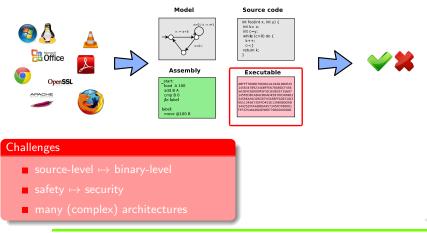
< □ ►

Binary-level semantic approaches

## Our proposal : binary-level semantic analysis

#### Semantic tools help make sense of binary

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

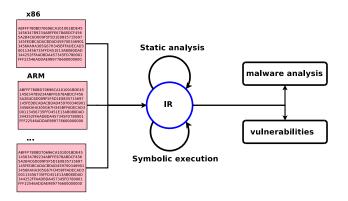


BINSEC team

cea

Binary-level semantic approaches

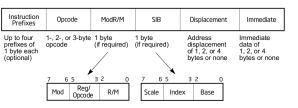
## **BINSEC** approach



- leverage powerful methods from formal software analysis
- pragmatic formal methods (combination, tradeoffs, etc.)
- common basic analysis + dedicated analysis (vuln., malware)

Binary-level semantic approaches

## Focus : modelling



#### Example of x86

- more than 1,000 instructions
  - $_{\sim}pprox$  400 basic
  - . + float, interrupts, mm×
- many side-effects
- error-prone decoding
  - . addressing modes, prefixes, ...

$\begin{array}{c c c c c c c c c c c c c c c c c c c $	r8(/r)			AL	CL	DL	BL	AH	СН	DH	SH .
$\begin{array}{c  c  c  c  c  c  c  c  c  c  c  c  c  $				AX	сх	DX					
$\begin{array}{c c c c c c c c c c c c c c c c c c c $											
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	nn(/r)				MM1	MM2	MH3	MN4	MMS	MMG	MN7
res         Gra         Gra <td>xmm(/r)</td> <td></td> <td></td> <td>хмма</td> <td>XMM1</td> <td>XMM2</td> <td>XMM3</td> <td>XMM4</td> <td>XMM5</td> <td>XMM6</td> <td>CMN7</td>	xmm(/r)			хмма	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	CMN7
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	sreg			ES				FS	GS	res.	res.
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	eee			CRO	invd	CR2	CR3	CR4	invd	invd	invd
$ \begin{array}{  c  c  c  c  c  c  c  c  c  c  c  c  c$	000			DRO	DR1	DR2	DR3	0941	DP5 <sup>1</sup>	DR6	DR7
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	(In decimal) /digit (0	pcor	(et		1	2	3	4	5	6	7
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$				666	881	818	811	188	101	110	111
	Effective Address	Mod	R/M	valu	e of	Mode	/M B	vte (	in He	ex)	-
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	[FAX]	00	000	00	0.0	10	10	20	20	30	30
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	[ECX]		001	01	69	11	19	21	29	31	39
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$											
Image: 1         Image: 1         Image: 2											
$\begin{array}{c c c c c c c c c c c c c c c c c c c $											
$ \begin{array}{  c c c } \hline (c) &  c c  &$											
$ \begin{array}{  c c c } \hline (c) &  c c  &$	(FST)		118	86	<b>BE</b>	16	1E	26	2F	36	3E
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$				07	0F						
$ \begin{array}{ c c c } \hline c c c c c c c c c c c c c c c c c c $		81			48						
ICEN         ICEN <th< td=""><td></td><td></td><td>001</td><td>41</td><td>49</td><td></td><td></td><td>61</td><td></td><td></td><td></td></th<>			001	41	49			61			
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$				42	44		54		66		ZA
12.22         13.23         47         46         54         55         64         56         74         75         75           12.23 </td <td></td> <td></td> <td>811</td> <td>43</td> <td>48</td> <td>53</td> <td>58</td> <td>63</td> <td>68</td> <td></td> <td>78</td>			811	43	48	53	58	63	68		78
Image         Image <th< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></th<>											
Test 7=14:30*         10         40         40         50         70         70         70         71			101	45	40	55	50	65	60		70
[162]         [163]         47         67         67         67         67         77         77           [163]         [164]         [166]         [			118	46	48	56		66	6F	76	7F
EXA         EXA <td></td>											
ICC1         ICL3         ICL3 <th< td=""><td></td><td>10</td><td></td><td>80</td><td>88</td><td></td><td>98</td><td></td><td>48</td><td></td><td>88</td></th<>		10		80	88		98		48		88
IDD:         Constraint         Constraint <td></td>											
Image: 1         Image: 2											
Image: 1         Image: 2											
[E87]         Edits         Bit											
Test											
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$											
ALAZAZUSTOWE/XWE         Des         Des <thdes< th=""></thdes<>											
CL/CXP/CX/T1/MH1/MMH1         001         C1         C0         D1         D1<		11									
DL/DX/EDX/512/MM2/XMM2 058 C2 CA D2 DA E2 EA F2 FA BL/BX/EDX/512/MM2/XMM3 051 C3 C6 D3 D6 E3 E5 F5 F6 AN/SP/ESP/512/MM2/XMM5 100 C4 CC D4 DC E4 EC F4 FC CH/BP/EDP/512/MM2/XMM5 101 C5 C0 D5 D0 E5 ED F5 FD DM/S1/ES1/510/MM2/XMM5 101 L5 C0 D5 D0 E5 ED F5 FD											
BL/BX/EBX/ST3/MM3/XMM3 011 C3 C8 D3 D8 E3 E8 F3 F8 AN/SP/ESP/ST4/MM4/XMM4 100 C4 CC D4 DC E4 EC F4 FC CN/BP/ESP/ST5/MM5/XMM5 100 C5 C0 D5 D0 E5 ED F5 FD DM/SI7ESI76/MM6/XMM5 119 C6 CE 06 DE E8 EE F6 F6 FE											
AH/SP/ESP/ST4/MM4/XMM4 100 C4 CC D4 DC E4 EC F4 FC CH/BP/ESP/ST5/MM5/XMM5 105 C5 CD D5 DD E5 ED F5 FD DH/SJ/SEJ/ST6/MM5/XMM5 1101 C6 C2 D6 UE F6 EE F6 F6 FE											
CH/8P/E8P/STS/MMS/XMM5 101 C5 CD D5 DD E5 ED F5 FD DH/S1/ES1/S16/MM6/XMM6 118 C6 CE D6 DE E6 EE F6 FE											
DH/S1/ES1/S16/MM6/XMM6 118 C6 CE D6 DE E6 EE F6 FE											
		-									
					CE	07	DE	E7	FF		

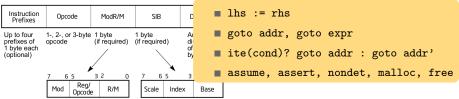
RMLL 2016: The Security Trate RI/EDI/ST7/MM7/XMM7 111 C7 CF D7 DF E7 EF F7 FF

23/44



Binary-level semantic approaches

## Focus : modelling



RMLL 2016: The Security T

#### Intermediate Representation [cav11]

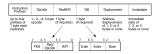
- architecture independent
- (really) reduced set of instructions
  - . 9 instructions, less than 30 operators
- simple, clear semantic, no side-effect

r8(/r)			AL	CL	DL	BL	AH	СН	DH	SH .
r16(/r)			AX	сх	DX	BX	SP	BP	SI	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
mm(/r)			MMO	MM1	MM2	MH3	MN4	MMS	MMG	MN7
xmm(/r)			хмма	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7
sreg			ES	CS	SS	DS	FS	GS	res.	res.
eee			CRO	invd	CR2	CR3	CR4	invd	invd	invd
000			DRO	DR1	DR2	DR3	DR41	DR5 <sup>1</sup>	DR6	DR7
(In decimal) /digit (0	pcor	(et		1	2	3	4 -	5	6	7
(In binary) REG =			888	881	818	811	188	101	110	111
Effective Address	Mod	R/M	valu	e of	ModR	/M B	yte (	in H	ex)	
[EAX]	00	000	00	80	10	18	20	28	30	38
[ECX]		001	01	69	11	19	21	29	31	39
[EDX]		818	82	8A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	18	23	28	33	3B
[sib]		100	04	OC.	14	10	24	2C	34	30
disp32		101	05	0D	15	10	25	20	35	30
[ESI]		110	86	θE	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX]+disp8	01	000	40	48	50	58	68	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		811	43	4B	53	58	63	68	73	78
[sib]+disp8		100	44	4C	54	5C	64	6C	74	70
[EBP]+disp8		101	45	40	55	50	65	6D	75	70
[ESI]+disp8		118	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	98	98	AB	A8	80	88
[ECX]+d1sp32		881	81	89	91	99	A1	A9	B1	89
[EDX]+disp32		818	82	88	92	9A	A2	**	B2	BA
[EBX]+disp32		011	83	88	93	98	A3	AB	63	88
[sib]+disp32		100	84	80	94	90	A4	AC	84	BC
[EBP]+disp32		101	85	80	95	9D	A5	AD	85	BD
[ESI]+disp32		110	86	36	96	98	AG	AE	86	30
[EDI]+disp32		111	87	8F	97	9F	A7	AF	87	BF
AL/AX/EAX/ST0/MM0/XMM0	11	000	CO	CB	00	DB	63	EB	FO	F8
CL/CX/ECX/ST1/MM1/XMM1		001	C1	C9	D1	09	81	29	F1	F9
DL/DX/EDX/ST2/MM2/XMM2		018	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/ST3/MM3/XMM3		011	C3	CB	03	DB	E3	EB	F3	FB
AH/SP/ESP/ST4/MM4/XMM4		100	C4	cc	D4	DC	E4	EC	F4	FC
		101	C5	CD	05	DD	65	ED	F5	FD
CH/BP/EBP/ST5/MM5/XMM5										
CH/BP/EBP/ST5/MM5/XMM5 DH/SI/ESI/ST6/MM6/XMM6 DH/DI/EDI/ST7/MM7/XMM7		118		CE	06	DE	EÐ	EE	F6	FR.

23/44

Binary-level semantic approaches

## x86 front-end



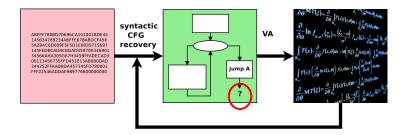
```
(0x29e,0) tmp := EBX + 7511;
(0x29e,1) OF := (EBX{31,31}=7511{31,31}) && (EBX{31,31}<>tmp{31,31});
(0x29e,2) SF := tmp{31,31};
(0x29e,3) ZF := (tmp = 0);
(0x28e,4) AF := ((extu (EBX{0,7}) 9) + (extu 7511{0,7} 9)){8,8};
(0x29e,6) CF := ((extu (EBX 33) + (extu 7511 33)){32,32};
(0x29e,7) EBX := tmp; goto (0x2a4,0)
```

< 🗆 🕨



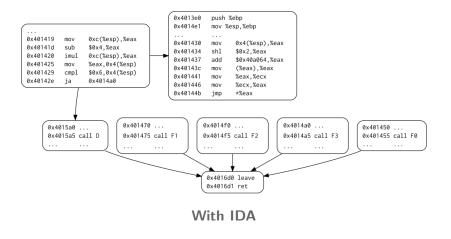
Binary-level semantic approaches Semantic disassembly

- simple obfuscation confuses soa disassemblers such as IDA
- ... because they rely on syntax
- semantic techniques complement and strengthen these approaches



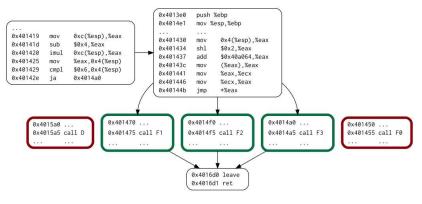
Binary-level semantic approaches

## Semantic disassembly (2)



Binary-level semantic approaches

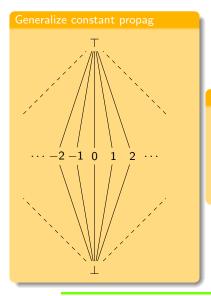
## Semantic disassembly (2)



### With IDA + BINSEC

Binary-level semantic approaches

## Semantic disassembly : keys



#### Framework : abstract interpretation

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

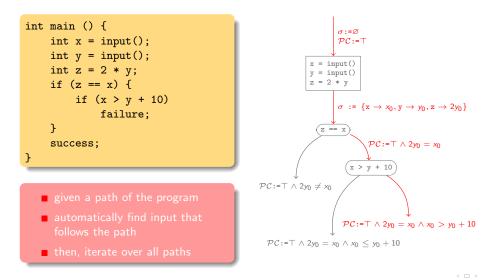
 $< \Box \rightarrow$ 

27/44



Binary-level semantic approaches

## Semantic exploration



BINSEC team



Binary-level semantic approaches

## Path predicate computation

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)

 $\prec \square \rightarrow$ 

Binary-level semantic approaches

## Path predicate computation

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

BINSEC team

Binary-level semantic approaches

## Path predicate computation

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

< □ ►

Binary-level semantic approaches

## Path predicate computation

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  
 $X_2 < 2 \times Z_0$ 

< □ ►

BINSEC team

Binary-level semantic approaches

# Path predicate computation

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  
 $X_2 < 2 \times Z_0 \land X_2 \ge Z_0$ 

< □ ►

BINSEC team

DE LA RECHERCHE À L'INDUSTR

Binary-level semantic approaches

# Semantic exploration (2)



 0000000
 5a4d
 0900
 0003
 0000
 0004
 0004
 fff
 0000

 0000010
 0005
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 0000
 <



Failure

### Crackme challenges

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

Binary-level semantic approaches

# Semantic exploration (2)

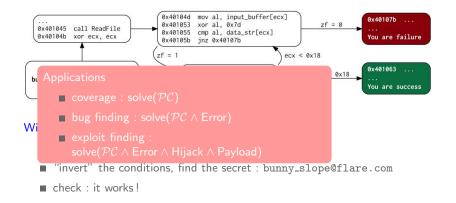


# With BINSEC [https://youtu.be/0xUc2jbpjQo]

- find the path leading to success
- "invert" the conditions, find the secret : bunny\_slope@flare.com
- check : it works !

Binary-level semantic approaches

# Semantic exploration (2)



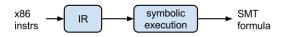


Binary-level semantic approaches

# Semantic exploration : keys

# Symbolic Execution

- path predicate computation
- formula preprocessing + SMT solver
- sound execution of the program [path necessarily feasible]



# Dynamic Symbolic Execution [DSE]

- combine dynamic and symbolic reasoning
- much more robust [missing code, self-modification, etc.]

 ${}^{+} \square {}^{+} \vdash$ 



# Preambule

- Benefits of binary-level analysis
- Challenges of binary-level analysis
- Semantic approaches
- BINSEC platform
- Achievements
- Conclusion

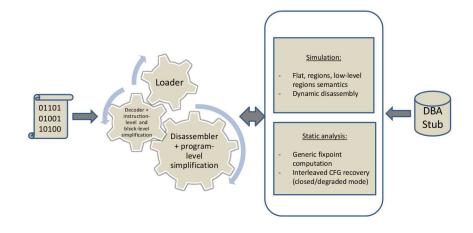
# BINSEC platform

### The BINSEC Platform [CEA & Uni. Grenoble Alpes]

- open source, lgpl v2.1
- mostly OCaml, 30 kloc (and pintool in C++)
- dual goal
  - help design new binary-level analyzers (basic building blocks)
  - provide innovative analyzers
- allows for combination of techniques
- current : multi-architecture support, semantic exploration & semantic disassembly, poc on vulnerabilities and deobfuscation
- still young : beta-version just released [http ://binsec.gforge.inria.fr/]

Thx to a bunch of enthusiastic students : Robin David, Adel Djoudi, Josselin Feist, Than Dihn Ta, Benjamin Farinier

# BINSEC platform (2)



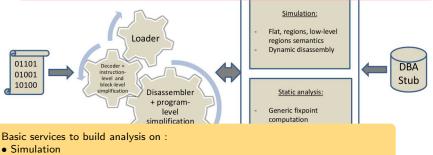
BINSEC team



Platform

# BINSEC platform (2)

- loader ELF/PE
- decoder (x86) + IR simplification
- 460/500 instructions : 380/380 "basic", 80/120 SIMD, no float/system
- prefixes : op size, addr size, repetition
- standard syntactic disassembly techniques : recursive, linear, combination

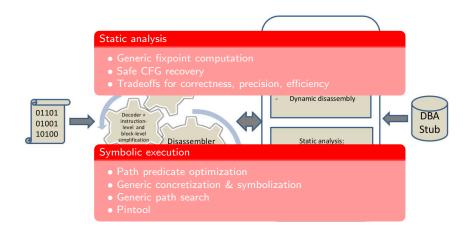


- Static analysis [semantic disassembly] [Adel Djoudi tacas15, sub. fm16]
- Symbolic execution [semantic exploration] [Robin David saner16,issta16]
- Combinations

BINSEC team

Platform

# BINSEC platform (2)



# DBA simplifications

#### Instruction level simplifications

Idiom simplifications [local rewriting rules]

## Block level simplifications

- Constants propagation
- Remove redundant assigns

### Program level simplifications

- Flag slicing (remove must-killed variables)
- granularity : function level+automatic summary of callees



### Platform DBA simplifications

#### Instruction level simplifications

Idiom simplifications [local rewriting rules]

### Block level simplifications

- Constants propagation
- Remove redundant assigns

#### Program level simplifications

Flag slicing (remove must \_\_\_\_\_

granularity : function level+automatic summary of callees

- Inspired from standard compiler optim
- Targets : flags & temp
- Sound : w.r.t. incomplete CFG
- Inter-procedural (summaries)



# DBA simplifications : Experiments

program	native	DBA	opt (DBA)				
	loc	loc	time	loc	red		
bash	166K	559K	673.61s	389K	30.45%		
cat	8K	23K	18.54s	18K	23.02%		
echo	4K	10K	6.96s	8K	24.26%		
less	23K	80K	69.99s	55K	30.96%		
ls	19K	63K	65.69s 44K		30.58%		
mkdir	8K	24K	19.74s	17K	29.50%		
netstat	17K	50K	52.59s	40K	20.05%		
ps	12K	36K	36.99s	27K	23.98%		
pwd	4K	11K	7.69s	9K	23.56%		
rm	10K	30K	24.93s	22K	25.24%		
sed	10K	32K	28.85s	23K	26.20%		
tar	64K	213K	242.96s	154K	27.48%		
touch	8K	26K	24.28s	18K	27.88%		
uname	3K	10K	6.99s	8K	23.62%		

	reduction							
	time	dba instr	tmp assigns	flag assigns				
BINSEC	1279.81s	28.64%	90.00%	67.04%				

< □ ►

Platform

# BINSEC platform (3)

#### What can be reused?

### whole analyses

- semantic exploration
- semantic disassembly

### basic blocks [need cleaner APIs]

- decoding
- disassembly (cfg, call graph)
- abstract fixpoint computation
- path predicate, formula simplification & solving
- generic path exploration
- ▶ pintool



# Preambule

- Benefits of binary-level analysis
- Challenges of binary-level analysis
- Semantic approaches
- BINSEC platform

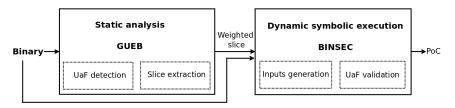
# Achievements

Conclusion

# Finding use-after-free vulnerabilities

A pragmatic two-step approach implemented within the BINSEC plateform :

not complete, but scalable and correct in some cases



- GUEB : scalable lightweight static analysis (not sound, not complete) → produces a set of CFGs slices containing potential UaF
- BINSEC/SE : guided symbolic execution → confirm the UaF by finding concrete program inputs



## Achievements Help to find the needle in the heap

4860	6966	5dc3	5589	e5c7	0812	0000	<b>00</b> b8	4860	6666	5dc3	558
			6666						00b8		
			66P8		ntry p	oint	540	bf0e	0821	6666	60b
e5c7	0540	bf0e	0822				5589	e5c7	0540	bf0e	082
5dc3	5589	e583	ec10	c705	0058	4960	0000	5dc3	5589	e583	ec1
6666	a148	bf0e	0883	f809	495F	0000	0100	0000	=149	bf0e	099
00c6	45f9	00c6	45 Fa	00c7	45£7	00c6	45.68	8866	45.69	8866	454
0000	00e9	d901	6666	c645	0548	hffle	6862	6666	6669	4981	866
c645	f980	c645	fa01	8070	£701	c645	£860	c645	£988	c645	Faß
0660	6666	e988	0100	60c9	c705	4851	0e08	0600	6969	e988	010
tc00	7401	c705	4866	0008	c645	fa02	807d	fc00	740f	c705	48b
0100	00e9	5991	6666	c645	0400	0000	e95e	0100	00c9	5901	666
c645	1900	C645	ta03	807d	f701	c645	f800	c645	f980	c645	fa0
fe00	7504	c705	4861	0e08	fd00	7410	807d	fe00	750a	c705	48b
fc00	750	c705	4861	0e08	0500	0000	807d	fc00	750a	c705	48b
fe00	7407	c705	4861	0e08	0300	0000	807d	fe00	740f	c705	48b
0100 c645	free	901	0000	C645	0600	0000	e90e	0100	00e9	0901	666
6645 48bf		045	1001	8019	f701	c645	f800	c645	f901	c645	fa0
4861	_	400	6666	eyen	1000	750f	c705	48bf	0e08	6466	666
0000	C645	1/01	C645	1800	0000	00e9	dfee	6666	c645	f701	c64
1404	80/0	TCUU	7410	80/0	c645	1988	c645	fa04	807d	fc00	741
4867	0608	0700	0000	80/0	ffee	790a	c705	48bf	0c08	0700	898
TTUU	7401	C/05	4801	0608	fc00	7415	807d	ff00	740f	c705	48b
0000	6000	9966	6-05	C645	0600	0000	e99e	0000	00c9	9980	666
6-00	7500	-705	4865	0-08	+701	c645	f800 807d	c645	1980	c645	ta0
6-00	7500	-705	4001	0000	1000	7410	807d	te00	750a	c705	486
6000	7506	8074	6600	740-	0800	0000	807d 807d	tc00	750a	c705	486
0600	0000	abdb	ah40	-645	0900	0000	8079	te00	7506	8070	110
6645	£001	6645	6.02	9074	C705	4861	e08	0000	0000	eb4b	eb4
Cd43	1901		0540	560-	1/01	C645	0000	C645	1981	C645	rae
5003	2289	esc/	0540	oroe	<b>00</b> b8	5400	0000	5dc3	5589	e5c7	054
4800	0000	4500	2202	esc/	0812	0000	0858 0820	4800	0000		
6000	0008	4500	0000	5003	0540	b†0e	0820 0540	0000	99t		666
0100	0540	bf0e	0008	0000	5589	e5c7	0540 5589	bt0e	682 U	se 🛛	
Ede 7	6540	-503	0022	-705	0000	5dc3	5589 G	Sc7	054	-	082
0000	-149	bf0e	0003	FROG	0008	4900	0100	Saca	558	e583	eci
8504	8548	e10b	0855	e0c6	4861	0e08	0100	0000	a148	DTOE	088
0004	4559	000	45fa	8867	0187	0002	0000 45f8	8004	8548	e100	081
8868	8869	4981	8888	c645	4517	0000	4518 0802	0000	4519	0006	451
c645	fgee	6645	fa81	8874	0548	bree.	0802 f800	0000	0009	0901	000
48hf	8688	8388	8888	8874	1/01	co45	c705	C045	1900	C045	T 80
fcee	750a	6785	48hf	8688	1000	1000	6705 807d	4801	0008	0300	000
fcee	7415	8874	fbee	748f	1000	/410	807d 807d	1000	1508	C/05	460
8688	8888	e988	8188	8869	-705	4056	807d 0c08	1000	/415	00/0	100
					C/05	40DT	0698	0000	0000	6398	010



## Achievements Help to find the needle in the heap

4860	6966	5dc3	5589	e5c7	0812	0000	00b8	4860	6969	5dc3	558
	00b8			-			0820	0000	00b8	4580	888
	0821			1 Er	ntry p	oint			0821		
e5c7	0540	bf0e	0822				\$589	e5c7	0540	bf0e	082
5dc3	5589	e583	ec10	C705	0058	4960	00.0	5dc3	5589	e583	ec1
6666	a148	bf0e	6883	f809	495F	0000	0100	0000	=149	bf0e	099
										00c6	45f
										d901	808
										C645	fa0
										\$300	666
fc80	750a	c785	48bf	-6e98	fbee	7410	8074	fcee	758a	c785	48b
fc00	7415	<b>#</b> 07d	fb80	740f	0900	0000	207d	fc00	741	807d	fb0
0600	0000	e988	0100	60e9	c705	485	0c08	0600	0000	c988	
f701	c646	f800	c645	f988	8201	0000	c645	f701	c645	f800	
fc00	740f	c705	4866	0008	c645	fa02	807d	fc00	40f	c705	48b
0100	90e9	5991	6000	c645	0400	0000	e95e	0100	00c9	5901	666
c645	1966	C645	ta03	807d	f701	c645	f800	c646	f980	c645	fa0
fe80	7504	c705	4861	0e08	fd00	7410	807d	fe00	750a	c705	48b
fc00	750	c705	4861	0e08	0500	0000	807d	£600	750a	c705	48b
Feee	7407	c705	4861	0e08	0300	0000	807d	/Fe00	740f	c705	48b
645	free	901	0000	C645	0600	0000	e98e	0100	00e9	0901	666
48bf	nee	1045	0000	8070	f701	-6645	1860	c645	f901	c645	fa0
4801	c645	6700	000	6000	1000	7501	705	485f	0e08	6460	666
0000	807d	6-00	200	1800	0009 c645	00e9	0166		c645		
401-6	8070	0700	7410	8070	c645	1966	C64	ta04	807d 0c08	†c00	741
4601 ££00	740f	6705	49bf	0070	1100	790a	c705	4861	0e08 740f	6766	666
0000	00e9	0000	0000	-	TCOD	1410	80/0	1400	00e9	C/05	480
645	f980	645	£=05	8074	0600	0000	6996	0000	6069		
fe00	750.	c705	486F	0.000	X/01	C645	1800	C645	\$50a	C645	Tau
fc00	750a	c705	48bf	0.000	1000	7410	8070	Teuu	7500	C/05	480
feßß	7586	807d	ff00	740c	0000	0000	8074	6.00	7506	2074	660
							1000	0600	0000	obth	ahd
c645	f901	c645	fa02	807d	€701	Sac	2000	6645	f901	645	6.04
5dc3	5589	e5c7	8548	hffle	0010	~~~~	1000	Cd-3	5589	X	05.4
4800	0000	Sdc3	5589	e5c7	0008	5400	0000	1000	0000	224	558
8666	00b8	4588	8888	Sdc3	0612	6000	3340	4800	0000	200.2	556
bfBe	0821	0000	Rebs	5886	5540	bi be	22.0	6600	082 U	se	eeb:
								567	054		882
										0503	ac 11
8666	a148	bf0e	0883	f809	495F	0.000	0100	0000	a149	bf0e	000
8666	00e9	d901	0000	c645	0548	hffe	6862	0000	0000	1.000	
c645	f900	c645	fa01	807d	£701	c645	£888	c645	£988	c645	fat
4851	0e08	0300	8888	807d	fbee	750a	C785	48hf	8688	0300	666
8688	0000	e988	0100	60e9	c705	48bf	0c08	0600	0000	e988	010
-		-		-							

< □ ▶



Achievements Results

### Combination of techniques is fruitful!

#### Several new vulnerabilities found

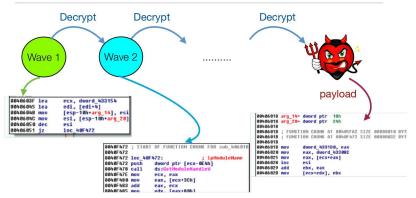
- GUEB + manual analysis [j. comp. virology 14]
  - tiff2pdf : CVE-2013-4232
  - openjpeg : CVE-2015-8871
  - gifcolor : CVE-2016-3177
  - accel-ppp
- GUEB + BINSE/SE [sefm16]
  - Jasper JPEG-2000 : CVE-2015-5221

# Gueb [Josselin Feist]

- MIT licence
- Ocaml, 5kloc
- https://github.com/montyly/gueb

# Malware deobfuscation

# Context : x86-malware



# A common protection scheme for malware a SillyFDC run

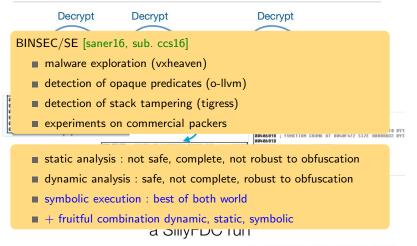
Self-modifying program schema

BINSEC team

RMLL 2016: The Security Track

# Malware deobfuscation

# Context : x86-malware



Self-modifying program schema

BINSEC team



# Preambule

- Benefits of binary-level analysis
- Challenges of binary-level analysis
- Semantic approaches
- BINSEC platform

# Achievements

# At last

 $\leftarrow$   $\square$   $\rightarrow$ 

#### Binary-level security analysis

- many applications, many challenges
- syntactic and dynamic are not enough

### Semantic approaches can help!

- semantic exploration, semantic disassembly
- yet, still hard to design

# The BINSEC Platform [CEA & Uni. Grenoble Alpes]

- open source, dual goal
  - help design new binary-level analyzers (basic building blocks)
  - provide innovative analyzers [already a few ones]
- current : multi-architecture support, semantic exploration & semantic disassembly, poc on vulnerability detection and deobfuscation
- still young : beta-version just released [http ://binsec.gforge.inria.fr/]

#### Binary-level security analysis

- many applications, many challenges
- syntactic and dynamic are not enough
- In progress
  - tutorials, doc
  - code cleaning
  - ARM and PowerPC

ntic disassembly

# The BINSEC Platform [CEA & Uni. Grenoble Alpes]

- open source, dual goal
  - help design new binary-level analyzers (basic building blocks)
  - provide innovative analyzers [already a few ones]
- current : multi-architecture support, semantic exploration & semantic disassembly, poc on vulnerability detection and deobfuscation
- still young : beta-version just released [http ://binsec.gforge.inria.fr/]

Conclusion

#### Binary-level security analysis

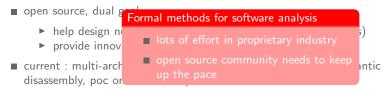
- many applications, many challenges
- syntactic and dynamic are not enough

#### In progress

- tutorials, doc
- code cleaning
- ARM and PowerPC

#### ntic disassembly

## The BINSEC Platform [CEA & Uni. Grenoble Alpes]



still young : beta-version just released [http ://binsec.gforge.inria.fr/]