

BINARY-LEVEL SECURITY: SEMANTIC ANALYSIS TO THE RESCUE

Sébastien Bardin (CEA LIST)

Joint work with Richard Bonichon, Robin David, Adel Djoudi & many other people







ABOUT MY LAB @CEA

CEA LIST, Software Safety & Security Lab























- Binary-level security analysis: many applications, many challenges
- Standard techniques (dynamic, syntactic) not enough
- Formal methods can help ... but must be strongly adapted
 - [Complement existing methods]
 - Need robustness, precision and scalability!
 - Acceptable to lose both correctness & completeness in a controlled way
 - New challenges and variations, many things to do!
- A tour on how formal methods can help
 - Explore and discover
 - Prove infeasibility or validity
 - Simplify (not covered here)
- -- with Josselin Feist
- -- with Robin David
 - -- with Jonathan Salwan











• Why binary-level analysis?

- Focus mostly on Symbolic Execution
- Give hints for abstract Interpretation
- Some background on source-level formal methods
- The hard journey from source to binary
- A few case-studies
- Conclusion

Cover both

- vulnerability detection
- deobfuscation





- Why binary-level analysis?
- Some background on source-level formal methods
- The hard journey from source to binary
- A few case-studies
- Conclusion













What for: vulnerabilities, reverse (malware, legacy),

protection evaluation, etc.





EXAMPLE: COMPILER BUG



- 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

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));
}

OpenSSH CVE-2016-0777

Our goal here:

Check the code after compilation





EXAMPLE: MALWARE COMPREHENSION

APT: highly sophisticated attacks

- Targeted malware
- Written by experts
- Attack: 0-days
- Defense: stealth, obfuscation
- Sponsored by states or mafia

The day after: malware comprehension

- understand what has been going on
- mitigate, fix and clean
- improve defense



USA elections: DNC Hack







Highly challenging [obfuscation]



CHALLENGE: CORRECT DISASSEMBLY



Basic reverse problem

- aka model recovery
- aka CFG recovery

list Ceatech

universite



CAN BE TRICKY!

dynamic jumps (jmp eax)





list Ceatech

STATE-OF-THE-ART TOOLS ARE NOT ENOUGH





list ^{Clatech}

[See later] CAN BECOME A NIGHTMARE WHEN OBFUSCATED



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









EXAMPLE: VULNERABILITY DETECTION

Find vulnerabilities before the bad guys

- On the whole program
- At binary-level
- Know only the entry point and program input format

4800	6669	5dc3	5589	e5c7	0812	6669	00b8	4800	6669	5dc3	558
0000	6698	4500	0000	-			0820	0000	6698	4566	000
bf0e	0821	0000	0068	4 Er	ntrv n	oint	540	bf0e	0821	0000	00b
e5c7	0540	bf0e	0822	<u>-</u> ا	, P	01110	519	e5c7	6540	bf0e	082
5dc3	5589	e583	ec10	C705	0058	4900	0000	5dc3	5589	e583	ec1
0000	a148	bf@e	0883	f809	48bf	6e08	0166	0000	a148	bf8e	088
8604	8548	e10b	08FF	e0c6	0597	6002	0000	8504	8548	e10b	08f
00c6	45f9	00c6	45 a	60c7	45f7	00c6	45f8	00c6	45f9	00c6	45f
0000	60c9	d901	0000	C645	0548	bf0e	0802	0000	60e9	d901	000
c645	f900	c645	fa01	807d	f701	c645	f860	c645	f900	c645	fa⊖
48bf	0e08	0360	0000	807d	Fb00	750a	c795	48bf	0e08	0300	000
fc00	750a	c705	48bf	6e08	fb00	7410	807d	fc00	750a	⊂705	48b
fc00	7415	807d	fb00	740f	0900	6669	207d	fc00	7415	807d	fb⊚
0600	6669	e988	0100	60e9	c705	485	0e68	0600	6669	e988	010
f701	c645	F800	c645	f900	8301	6000	c645	f701	c645	F800	c64
fc00	740f	c765	4856	0e08	c645	fa02	807d	fc00	740f	<765	48b
0100	60e9	5901	0000	c645	0400	6669	e95e	0100	60e9	5901	000
c645	f900	645	fa03	807d	f701	c645	F860	c645	f900	c645	fa0
fe00	7507	⊂705	48bf	6e08	fd00	7410	807d	fe00	750a	⊂705	48b
fc00	756	C705	48bf	6e08	0500	6669	807d	fc00	750a	⊂765	48b
fe00	746 F	C785	48bf	6e08	0300	6669	807d	fe00	740f	C785	48b
0100	free a	901	0000	c645	0600	6666	e90e	0100	60e9	0901	000
c645	free	045	1001	8070	f701	c645	f860	c645	f901	c645	fa0
48bf	_	466	0000	c9c4	F400	750f	c765	48bf	6e08	0466	000
0000	c645	F701	c645	f800	0005	68e9	dfee	0000	c645	f701	c64
Ta04	8070	FC00	7410	8070	c645	K 900	c645	fa04	807d	fc00	741
48bf	6668	0766	0000	8070	ff00	75Qa	c705	48bf	6e08	0766	000
1100	7401	c765	4851	6e08	fc00	7416	807d	ff00	740f	C705	48b
0000	00e9	9900	0000	C045	0600	6666	e99e	0000	60c9	9966	000
C645	1900	C645	Ta05	80/08	f701	c645	F800	c645	f900	c645	fa0
TC00	7500	C785	48DT	0008	fd00	7410	807d	fe00	750a	C765	48b
6-00	7508	0074	4801	240	0800	6669	807d	fc00	750a	C785	48b
Te00	7500	8070	ob 40	7400	0900	0000	807d	fe00	7506	807d	ff0
0000	6000	0040	6049	0074	c705	486f	9 e08	0600	0000	eb4b	eb4
C045	1901	0645	1402	8070	f701	C645	f860	c645	f901	C645	fa0
SOCS	5589	esc/	0540	bree	0008	5400	0000	5dc3	5589	e5c7	054
1800	6669	50C3	5589	escr	0812	6669	0008	4800	6669	5dc3	558
3000	GODS	4566	0000	5dc3	0540	bf0e	0820	0000	60F	. P	000
proe	0821	0000	0058	2800	5589	eSc7	0540	bf0e	082 U	ise	00bi
2507	0540	bree	0822	6000	0000	50C3	5589	85C7	054	<u> </u>	082
>0C3	5589	6583	ec10	6705	0008	4900	0000	5004	558	e583	eci
3000	8148	DTUE	0883	T809	4851	6e08	0166	0000	a148	bfBe	088
3004	8548 45f0	0000	4565	euco	0187	0002	0000	8504	8548	e105	081
2000	4519	doce	4519	0007	4517	0006	4518	0000	4519	0006	4511
-645	5000	0901	5-01	0074	0548	DTUE	0802	0000	6663	0901	000
1045	0-02	0366	0000	8074	T701	C645	7866	C645	1900	C645	T 80:
FCOP	750-	0300	49b€	8010	1000	7508	C/05	4801	0008	0300	0000
Ecolo	7308	0074	fbae	7405	1000	/410	80/0	1000	750a	C/05	48D
1000	0000	-000	0100	00.00	9900	9999	8670	TCOO	/415	8670	TDO
1000	0000	2366	0100	0009	c705	4801	0668	0600	6666	6988	010



EXAMPLE: VULNERABILITY DETECTION

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!
```





CHALLENGE: In-depth exploration (example: use after free)

Find a needle in the heap!

Dynamic: not enough

• Too incomplete

- sequence of events, importance of aliasing
- strongly depend on implem of malloc and free

montdenine.com

4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558 0000 00b8 4500 0000 820 0000 0068 4500 000 bf0e 0821 0000 00b8 540 bf0e 0821 0000 00b Entry point 519 e5c7 0540 bf0e 082 e5c7 0540 bf0e 0822 5dc3 5589 e583 ec10 c705 00b8 4900 00 0 5dc3 5589 e583 ec1 0000 a148 bf0c 0883 f809 48bf 0c08 0100 a148 bf0c 088 8b04 8548 e10b 08ff e0c6 org7 0002 0000 8b04 8548 e10b 08f 00c6 45f9 00c6 45fa 60c7 45f7 60c6 45f8 00c6 45f9 00c6 45f 0000 60c9 d961 0000 c645 0548 bf0e 0862 0000 60e9 d961 000 c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0 485f 0e08 0300 0000 807d fb00 750a c745 485f 0e08 0300 000 fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 48b fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0 0600 6000 c988 0100 60c9 c705 48br 0c68 0500 6000 c988 010 f701 c645 f800 c645 f900 g301 0000 c645 f701 c645 f800 c64 fc00 740f c705 48bf 0008 c645 fa02 807d fc00 740f c705 48b 60e9 5901 0000 c645 0400 6000 e95e 0100 60e9 5961 000 c645 f900 645 fa03 807d f701 c645 f800 c645 f900 c645 fa0 750 c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b fc00 75 e c705 48bf 0008 0500 0000 807d fc00 750a c705 48b 746 c785 48bf 8e88 0300 8000 807d fe00 740f c785 48b fe00 961 0000 c645 0600 6000 e96e 0100 60e9 0961 000 0100 free 645 fa01 807d f701 c645 f800 c645 f901 c645 fa0 c645 460 0000 c9c4 5400 750f c765 48bf 6e08 0460 000 c645 f761 c645 f800 0000 60c9 df60 0000 c645 f761 c64 48bf fa04 807d fc00 7410 807d c645 900 c645 fa04 807d fc00 741 48bf 6e08 0760 0000 807d ff00 730a c765 48bf 6e08 0760 000 740f c705 48bf 8e08 fc00 7416 807d ff00 740f c705 48b 0000 00e9 9900 0000 c645 0600 6000 e99e 0000 60e9 9960 000 c645 f900 c645 fa05 807d f701 c645 f800 c645 f900 c645 fa0 fe00 750a c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b fc00 750a c705 48bf 0e08 0800 0000 807d fc00 750a c705 48b fe00 7506 807d ff00 740c 0900 0000 007d fe00 7506 807d ff0 0600 0000 eb4b eb49 c645 c705 48bf de08 0600 0000 eb4b eb4 c645 f901 c645 fa02 807d f701 c645 f800 c645 f901 c645 fa0 idc3 5589 e5c7 0540 bf0e 00b8 5400 0000 5dc3 5589 e5c7 0540 1800 0000 5dc3 5589 e5c7 0812 0000 0008 4800 0000 5dc3 558 3000 00b8 4500 0000 5dc3 0540 bf0e 0820 0000 00b 000 >F0e 0821 0000 00b8 5800 5589 e5c7 0540 bf0e 082 USE 00bi ≥5c7 6540 bf6e 0822 6000 0000 5dc3 5589 €5c7 654 082 5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 558 3000 a148 bfGe 0883 f809 48bf Ge08 0160 0000 a148 bfGe 3b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 30c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f; 3000 00c9 d901 0000 c645 0548 bf0e 0802 0000 00e9 d901 :645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0 18bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a c705 fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d fb0 3600 6000 c988 0100 60c9 c705 48bf 0c68 0600 6000 c988 010



list ^{Ceatech}

BONUS: (MULTI-)ARCHITECTURE SUPPORT



Example of x86

- more than 1,000 instructions
 - $. \approx 400$ basic
 - . + float, interrupts, mmx
- many side-effects
- error-prone decoding
 - . addressing modes, prefixes,

rð(/r)			AL	CL	DL	DL	AH	CH	DH	DH
r16(/r)			AX	сx	DX	БX	SP	5P	51	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	CBP	E51	EDI
nn(/r)			MMO	MM1	MM2	MM3	MM4	MM5	MMG	MM7
xnn(/r)			XMM6	XMM1	XMM2	хииз	XMM4	ХММ5	XMM6	ХММ7
sreg			ES	CS	SS	DS	FS	GS	res.	res.
eee			CR0	invd	CR2	CR3	CR4	1nvd	invd	1nvd
000			DRB	DR1	DR2	DR3	DR4 ¹	DR5 ¹	DR6	DR7
(In decimal) /digit (O	pcod	ie)	0	1	2	3	4	5	6	7
(In binary) REG =			000	001	010	011	100	101	110	111
Effective Address	Mod	R/M	Valu	e of	ModR	/M By	yte (1n H	ex)	
[EAX]	00	000	66	08	10	18	20	28	30	38
[ECX]		001	61	09	11	19	21	29	31	39
[EDX]		010	62	θA	12	1٨	22	2A	32	3A
[EDX]		011	03	05	13	10	23	20	33	30
[<u>sib</u>]		100	04	00	14	10	24	20	34	30
disp32		101	05	00	15	1D	25	2D	35	30
[ESI]		110	00	0E	16	15	26	ZE	36	35
[EDI]		111	07	ØF	17	1F	27	ZF	37	3F
[EAX]+disp8	81	999	46	48	58	58	68	68	76	78
[ECX]+disp8		881	41	49	51	59	61	69	71	79
[EDX]+disp8		919	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	58	63	6B	73	7B
[sib]+disp8		100	44	4C	54	5C	64	6C	74	70
[EBP]+disp8		101	45	4D	66	50	65	6D	75	70
[ESI]+disp0		110	46	4E	56	5C	66	6C	76	7E
[EDI]+disp0		111	47	4F	57	5F	67	0F	77	7 F
[EAX]+d1sp32	10	000	88	88	90	98	AB	AB	80	88
[ECX]+d1sp32		001	81	89	91	99	A1	A9	81	89
[EDX]+d1sp32		010	82	8A	92	9A	A2	AA	82	BA
[EBX]+disp32		811	83	88	93	9B	A3	AB	83	BB
[<u>sib</u>]+disp32		100	84	80	94	90	A4	AC	B4	BC
[EBP]+disp32		101	85	80	95	90	AB	AD	85	BD
[ESI]+disp32		110	86	8E	96	9E	AG	AE	B6	BE
[EDI]+disp32		111	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/ST0/MM0/XMM0	11	000	CO	Cõ	DO	DB	EO	55	FØ	Fð
CL/CX/ECX/ST1/MM1/XMM1		001	C1	C9	01	09	E1	E9	F1	F9
DL/DX/EDX/ST2/MM2/XMM2		010	C2	GA	DZ	DA	EZ	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	04	DC	E4	EC	F4	FC
CH/BP/EBP/ST5/MM5/XMM5		101	C5	CD	05	00	E5	ED	E5	FD





THE SITUATION

- Binary-level security analysis is necessary
- Binary-level security analysis is highly challenging (*)
- Standard tools are not enough experts need better help!

(*) i.e., more challenging than source code analysis

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





SOLUTION? 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

Semantic preserved by compilation or obfuscation





- Why binary-level analysis?
- Some background on source-level formal methods
- The hard journey from source to binary
- A few case-studies
- Conclusion





BACK IN TIME: THE SOFTWARE CRISIS (1969)

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly : as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. - Edsger Dijkstra, The Humble Programmer (EWD340)



http://en.wikipedia.org/wiki/List_of_software_bugs

Testing can only reveal the presence of errors but never their absence. - E. W. Dijkstra (Notes on Structured Programming, 1972)







ABOUT FORMAL METHODS

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



Key concepts : $M \models \varphi$				
M : semantic of the program				
• φ : property to be checked				
= : algorithmic check				

Kind of properties

- absence of runtime error
- pre/post-conditions
- temporal properties



A DREAM COME TRUE ... IN CERTAIN DOMAINS

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





A DREAM COME TRUE ... IN CERTAIN DOMAINS (2)

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)





OVERVIEW OF FORMAL METHODS

Semantics

- Precise meaning for the domain of evaluation and the effect of instructions
- Operational semantics = « interpreter »

Properties

- From Invariants / reachability to safety/liveness/hyper-properties/...
- On software: mostly invariants and reachability

Algorithms:

- Historically: Weakest precondition, Abstract interpretation, model checking
- Correctness: the analysis explores only behaviors of interest
- Completeness: the analysis explores at least all behaviors of interest





OVERVIEW OF FORMAL METHODS

Trends:

- Frontier between techniques disappear
- master abstraction (correct xor complete)
- reduction to logic
- sweet spots

Next:

- AI: complete (can prove invariants) -- 1977
- DSE: correct (can find bugs)

- Representative
- Industrial successes at source-level
- Adaptation to binary: very different situations



-- 2005



ABSTRACT INTERPRETATION

$$(\mathcal{P}(states), \cup, \cap, \rightarrow) \stackrel{\gamma}{\underset{\alpha}{\hookrightarrow}} (states^{\#}, \sqcup, \Pi, \rightarrow^{\#})$$





list Ceatech

ABSTRACT INTERPRETATION IN PRACTICE



nœud	С	X	у	Ζ
0	Т	\vdash	Τ	Т
1	Т	2	Т	Т
2	Т	2	Τ	3
ip 3	Т	2	Τ	3
4	0	2	Τ	3
5	Т	- 2	Т	3
6	0	2	Т	3
7	Т	- T 2	Т	3
8	\top	T	\top	3
9	Т	Т	Τ	9





ABSTRACT INTERPRETATION IN PRACTICE

Key points:

- Infinite data: abstract domain
- Path explosion: merge
- Loops: widening

In practice:

- Tradeoff between cost and precision
- Tradeoff between generic & dedicated domains
- It is sometimes simple and useful
- taint, pointer nullness, typing

Big successes: Astrée, Frama-C, Clousot







DYNAMIC SYMBOLIC EXECUTION

(DSE, 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 the program
- automatically find input that follows the path
- then, iterate over all paths

 $\sigma := \emptyset$ $\mathcal{PC} := \top$

Perfect for intensive testing

- Correct, relatively complete
- No false alarm
- Robust
- Scale in some ways

// incomplete

$$\mathcal{PC} := \top \land 2y_0 \neq x_0$$

$$\mathcal{PC} := \top \land 2y_0 = x_0 \land x_0 > y_0 + 10$$

$$\mathcal{PC} := \top \land 2y_0 = x_0 \land x_0 \leq y_0 + 10$$



DSE: PATH PREDICATE COMPUTATION (DSE, Godefroid 2005)

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





DSE: GLOBAL PROCEDURE

(DSE, Godefroid 2005)

input : a program P

output : a test suite *TS* covering all feasible paths of $Paths^{\leq k}(P)$

- **pick a path** $\sigma \in Paths^{\leq k}(\mathbb{P})$
- compute a *path predicate* φ_{σ} of σ
- **solve** φ_{σ} for satisfiability
- **SAT**(s)? get a new pair < s, σ >
- loop until no more path to cover







ABOUT ROBUSTNESS (imo, the major advantage)

```
Goal = find input leading to ERROR
(assume we have only a solver for linear integer arith.)
```

```
g(int x) {return x*x; }
f(int x, int y) {z=g(x); if (y == z) ERROR; else OK }
```

Symbolic Execution

• create a subformula z = x * x, out of theory [FAIL]

Dynamic Symbolic Execution

- first concrete execution with x=3, y=5 [goto OK]
- during path predicate computation, x * x not supported
 - . x is concretized to 3 and z is forced to 9
- resulting path predicate : $x = 3 \land z = 9 \land y = z$
- a solution is found : x=3, y=9 [goto ERROR] [SUCCESS]

« concretization »

- Keep going when symbolic reasoning fails
- Tune the tradeoff genericity

- cost



Three key ingredients

- Path predicate & solving
- Path enumeration
- C/S policy

Limits

- #paths -> better heuristics (?), state merging, distributed search, path pruning, adaptation to coverage objectives, etc.
- solving cost -> preprocessing, caching, incremental solving, aggressive concretization (good?)
 [wait for better solvers ©]
- Preconditions/postconditions/advanced stubs



DSE: PATH PREDICATE MAY BE COMPLICATED

x := a + b

 $X_{n+1} = A_n + B_n$

store(M, addr(X), load(M, addr(A)) + load(M, addr(B)))

```
let tmpA = load(M, addr(A)) @ load(M, addr(A)+1) @ load(M, addr(A)+2)
and tmpB = load(M, addr(B)) @ load(M, addr(B)+1) @ load(M, addr(B)+2)
in
let nX = tmpA+tmpB
in
store(
store(
store(M, addr(X), nX[0]),
addr(X) + 1, nX[1]),
addr(X) + 2, nX[2])
```





DSE: SEARCH





Search heurstics matters

- But no good choice (hint: DFS is often the worst)
- The engine must provide flexibility





DSE: SEARCH (2)



Generic engine

- Score each active prefix
- Pick the best & expand
- Easy encoding of many heuristics


C/S POLICIES

Robustness : what if the instruction cannot be reasoned about?

- missing code, self-modification
- hash functions, dynamic memory accesses, NLA operators



Solutions

- Concretization : replace by runtime value [lose completeness]
- Symbolization : replace by fresh variable [lose correctness]



C/S POLICIES (2)

Consider the following situation

- instruction x := @(a * b)
- your tool documentation says : *"memory accesses are concretized"*
- suppose that at runtime : a = 7, b = 3

What is the intended meaning? [perfect reasoning : $x == select(M, a \times b)$]

CS1 : x == select(M, 21)[incorrect]CS2 : $x == select(M, 21) \land a \times b == 21$ [minimal]CS3 : $x == select(M, 21) \land a == 7 \land b == 3$ [atomic]

No best choice, depends on the context

- acceptable loss of correctness / completeness?
- is it mandatory to get rid off \times ?

C/S policy matters

- But no good choice
- The engine must provide flexibility











- Why binary-level analysis?
- Some background on source-level formal methods
- The hard journey from source to binary
- A few case-studies
- Conclusion





NOW: BINARY-LEVEL SECURITY





Source code



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 FFF22546ADDAE98977660000000





THE HARD JOURNEY FROM SOURCE TO BINARY

Low-level semantics of data

- machine arithmetic, bit-level operations, untyped memory
- difficult for any state-of-the-art formal technique

Low-level semantics of control

- no distinction data / instructions, dynamic jumps (jmp eax)
- no (easy) syntactic recovery of Control-Flow Graph (CFG)
- violate an implicit prerequisite for most formal techniques

Diversity of architectures and instruction sets

- support for many instructions, modelling issues
- tedious, time consuming and error prone

Wanted

- robustness
- precision
- scale





DSE is quite easy to adapt

- thx to SMT solvers (arrays+bitvectors)
- thx to concretization
- yet, performance degrades

Al is much more complicated

- Even for « normal » code
- btw, cannot expect better than source-level precision

Problems

- Low-level control: jump eax
- Low-level data: memory
- Low-level data: flags

Problem solved: multi-architecture

rely on some IR





FULL DISCLOSURE: the BINSEC tool

Semantic analysis for binary-level security

- Help make sense of binary
- more robust than syntactic
- more exhaustive than dynamic

Some features

- Help to recover a simple model
- Identify feasible events (+ input)
- Identify infeasible events (eg, protections)
- Multi-architecture

Challenges

- Binary analysis
- Scalability
- Robustness w.r.t obfuscation

Still very young!







CARNOT TN@UPSaclay



INTERMEDIATE REPRESENTATION

81 c3 57 1d 00 00

Instruction Prefixes	Opcode	ModR/M	SIB	Displacement	Immediate
Up to four prefixes of 1 byte each (optional)	1-, 2-, or 3-byte 1 byte opcode (if required)		1 byte (if required)	Address displacement of 1, 2, or 4 bytes or none	Immediate data of 1, 2, or 4 bytes or none
	7 6 5 Mod Reg/ Opcod	3 2 0 R/M	7 65 Scale Inde	3 2 0 ex Base	

x86reference

ADD EBX 1d57

- goto addr, goto expr
- ite(cond)? goto addr

- Concise
- Well-defined
- Clear, side-effect free

```
(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)
```



INTERMEDIATE REPRESENTATION + simplifications

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%	88.00%	75.04%					

Approach

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

- IR level
- machine-instruction level
- program level



list

BINARY-LEVEL DSE (Godefroid)





DSE COMPLEMENTS DYNAMIC ANALYSIS







With IDA + BINSEC

Can recover useful semantic information

- More precise disassembly
- Exact semantic of instructions
- Input of interest

. . .







Ceatech



list Ceatech

ABSTRACT INTERPRETATION IS VERY VERY HARD ON BINARY CODE





Sébastien Bardin -- ISSISP 2017 | 52





ISSUE: LACK of HIGH-LEVEL STRUCTURE

Problems

- Jump eax
- memory ullet
- Bit resoning

universite

High-level conditions translated into low-level flag predicates

11

Condition on flags, not on register (nor stack)



LOW-LEVEL CONDITIONS

	flag predicate	cmp x y	sub x y	test x y
		predicate	predicate ²	predicate
ja, jnbe	$\neg CF \land \neg ZF$	$x >_{u} y$	x' ≠ 0	<i>x</i> & <i>y</i> ≠ 0
jae, jnb, jnc	$\neg CF$	$x \ge_{u} y$	true	true
jb, jnae, jc	CF	х < _и у	x' ≠ 0	false
jbe, jna	$CF \lor ZF$	$x \leq_{u} y$	true	x&y = 0
je, jz	ZF	x = y	x' = 0	x&y = 0
jne, jnz	$\neg ZF$	$x \neq y$	x' ≠ 0	$x\&y \neq 0$
jg, jnle	$\neg ZF \land (OF = SF)$	x > y	x' > 0	$(x \& y \neq 0) \land$ $(x \ge 0 \lor y \ge 0)$
jge, jnl	(OF = SF)	$x \ge y$	true	$(x \ge 0 \lor y \ge 0)$
jl, jnge	$(OF \neq SF)$	x < y	x' < 0	$(x < 0 \land y < 0)$
jle, jng	$ZF \lor (OF \neq SF)$	$x \leq y$	true	$(x\&y = 0) \lor \\ (x < 0 \land y < 0)$





LOW-LEVEL CONDITIONS

example	retrieved condition	patterns
or eax, O	if $(eax = 0)$ then goto	×
je		
cmp eax, O	if (eax \ge 0) then goto	×
jns		
sar ebp, 1	if $(ebp = 0)$ then goto	×
je		
dec ecx	if $(ecx \ge 0)$ then goto	×
jg		





Precision refinement [Brauer, 2011]



Degraded mode [Kinder, 2012]







SOLUTIONS? (2)

Insights

-) Complex predicates often hide simple predicates
- Only a few templates : $>_{u,s}, <_{u,s}, \ge_{u,s}, \le_{u,s}, =, \neq$

Try to find the appropriate one through equivalence checking
 Optimization :

- Once per address using cache
 - Cheap pruning through filtering

Approach	archi.	Sound	Complete
	independent		enough
Patterns	×	√/×	×
Logic-based	\checkmark	\checkmark	×
Template-based	\checkmark	\checkmark	\checkmark



HIGH-LEVEL CONDITION RECOVERY

method	#loc [†]	#cond [‡]	#success*	time	time _{all}
templates	242884	1978	1760 (<mark>89%</mark>)	22.93	2674.81
logic-based	247894	2260	694 (<mark>31%</mark>)	0.003	2561.64
patterns	229255	1987	1357 (<mark>68%</mark>)	0.014	2373.33
templates+patterns	242884	1978	1838 (92%)	9.17	2659.95





STATIC ANALYSIS in BINSEC an overview

	difficulty	solution
Domains	low-level arithmetic	dual-intervals
	ubiquitous data moves	equality domain
	low-level conditions	flag domain + condition recovery
Widening	no loop structure	loop detection
		widening point positioning
CFG	unavailable	incremental CFG recovery
		backward precision recovery
	recovery scale	degraded mode [Kinder2012]





OVERVIEW

	Correct	Complete	Efficient	Robust
Static syntactic	Х	X /	OK	Х
Dynamic	OK	XX	OK	OK
DSE	OK		Х	OK
Static semantic	Х	OK / X	Х	Х



- Why binary-level analysis?
- Some background on source-level formal methods
- The hard journey from source to binary
- A few case-studies
- Conclusion





APPLICATION: VULNERABILITY DETECTION

Find vulnerabilities before the bad guys

- On the whole program
- At binary-level
- Know only the entry point and program input format



4800	6669	5dc3	5589	e5c7	0812	6669	00b8	4800	6669	5dc3	558
0000	6698	4566	0000	1			0820	0000	6698	4566	000
bf0e	0821	0000	0058	4 Er	ntrv p	oint	540	bf0e	0821	0000	00b
e5c7	6540	bf0e	0822	<u>ا</u>	, .		5 9	e5c7	0540	bf0e	082
5dc3	5589	e583	ec10	C705	00b8	4900	00 0	5dc3	5589	e583	ec1
0000	a148	bf8e	0883	f809	48bf	8e08	0166	0000	a148	bf8e	088
8504	8548	e10b	08FF	e0c6	0597	6002	0000	8504	8548	e10b	08f
00c6	45f9	00c6	45 a	60c7	45f7	00c6	45f8	00c6	45f9	00c6	45f
0000	60c9	d901	0000	c645	0548	bf0e	0862	0000	60e9	d901	000
c645	f900	c645	fa01	807d	f701	c645	f86 0	c645	f900	c645	fa⊖
48bf	0e08	0300	0000	807d	F600	750a	c795	48bf	0e08	0300	000
fc00	750a	C765	48bf	0e08	fb00	7410	867d	fc00	750a	⊂705	48b
fc00	7415	807d	fb00	740f	0900	6669	807d	fc00	7415	807d	fbø
0600	6669	e988	0100	60c9	c705	485	0e68	0600	6669	e988	010
f701	c645	F860	c645	f900	8391	0000	c645	f701	c645	F800	c64
fc00	740f	⊂765	4855	0008	c645	fa02	807d	fc00	740f	c705	48b
0100	60e9	5991	0000	c645	0400	6669	e95e	0100	60e9	5901	000
c645	f900	645	fa03	807d	f701	c645	F860	c645	f900	c645	faΘ
fe00	7507	⊂765	48bf	0e08	fd00	7410	807d	fe00	750a	⊂705	48b
fc00	756	C705	48bf	0e08	0500	6669	807d	fc00	750a	<765	48b
fe00	746 F	C785	48bf	6e08	0300	6669	807d	fe00	740f	C705	48b
0100		901	0000	c645	0600	6666	e96e	0100	60e9	0901	000
c645	free	045	1001	8070	f701	c645	f800	c645	f901	c645	fa0
48bf	_	466	0000	c9c4	Fd00	750f	c765	48bf	0e08	0466	000
0000	c645	f701	c645	f800	0006	60e9	dfee	0000	c645	f701	c64
fa04	807d	fc00	7410	807d	c645	1900	c645	fa04	807d	fc00	741
48bf	6e08	0766	0000	8070	ff00	75Qa	c765	48bf	6e08	0766	000
ff00	740	c785	48bf	8e08	fc00	7416	807d	ff00	740f	c705	48b
0000	00e9	9900	0000	c645	0600	6666	e99e	0000	66c9	9966	000
C645	1900	C645	1905	8070	f701	c645	F800	c645	f900	c645	fa0
1e00	750a	C705	4861	0c08	fd00	7410	807d	fe00	750a	c765	48b
1000	750a	C705	4851	0e08	0800	6669	807d	fc00	750a	C705	48b
†e00	7506	8074	++00	740c	0900	6666	807d	fe00	7506	807d	ff0
0600	6669	eb4b	eb49	C645	c705	48bf	0e08	0600	0000	eb4b	eb4
c645	† 901	C645	†a02	8070	f701	C645	f 860	c645	f901	C645	fa0
5dc3	5589	e5c7	0540	bf0e	00b8	5400	0000	5dc3	5589	e5c7	054(
1800	6669	5dc3	5589	e5c7	0812	6669	8000	4800	6665	5dc3	558
9996	6698	4500	0000	Sdc3	0540	bf0e	<u>082</u> 0	0000	661		000
of0e	0821	0000	0058	5800	5589	e5c7	0540	bf0e	682 U	ise 🕨	00bi
≥5c7	6540	bf0e	0822	6669	0000	5dc3	5589	\$5c7	654		082
5dc3	5589	e583	ec10	C705	00b8	4900	0000	5002	558	e583	ec10
9000	a148	bf0e	0883	f809	48bf	6e08	0166	0000	a148	bf8e	088
3504	8548	e10b	08Ff	e0c6	0f87	6002	0000	8604	8548	e10b	08F
90C6	4519	00c6	45fa	60c7	45f7	00c6	45f8	00c6	45f9	00c6	45fi
0000	60c9	d901	0000	C645	0548	bf0e	0802	0000	60e9	d901	000
:645	1900	c645	†a01	807d	f701	c645	f860	c645	f900	c645	fa0:
186f	0e08	0300	0000	807d	F600	750a	c705	48bf	0e08	0300	000
FC00	750a	<705	48bf	0e08	fb00	7410	807d	fc00	750a	⊂705	48b
FC00	7415	807d	T 000	740f	0900	6669	807d	fc00	7415	807d	fboi
1600	6669	e988	0100	60e9	c705	48bf	0e68	0600	6669	e988	010





APPLICATION: VULNERABILITY DETECTION

Many successful applications of pure DSE

- SAGE @ Microsoft
- Mayhem/VeriT @ ForallSecure

cf. Cyber Grand Challenge



4800 0000 5dc3 5589 e5c7 0812 0000 00b8 4800 0000 5dc3 558 66P8 4266 6996 820 0000 0068 4500 000 540 bf0e 0821 0000 Entry point e5c7 0540 bf0e 0822 519 e5c7 0540 5dc3 5589 e583 ec10 c705 00b8 4900 00 B 5dc3 5589 e583 a148 bf6e 0883 f809 48bf 6e08 0166 0000 a148 bf6e 088 8548 e10b 08ff e0c6 oroz 0002 0000 8b04 8548 e10b 08f 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 45f 60c9 d961 0000 c645 0548 bf0e 0862 0000 60e9 d961 000 c645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 fa0 48bf 0e08 0300 0000 807d fb00 750a c7#5 48bf 0e08 0300 fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 0600 6000 c988 0100 60c9 c705 48br 0c68 0500 6000 c988 010 f701 c645 f800 c645 f900 g301 0000 c645 f701 c645 740f c705 48bf 0008 c645 fa02 807d fc00 740f c705 48b 68e9 5901 0000 c645 0400 6000 e95e 0100 60e9 5961 c645 f900 2645 fa03 807d f701 c645 f800 c645 f900 c645 fa0 750g c705 48bf 0e08 fd00 7410 807d fe00 750a c705 48b 75 c705 48bf 0e08 0500 0000 807d fc00 750a fe00 740 c705 48bf 0e08 0300 0000 807d fe00 901 0000 c645 0600 0000 e90e 0100 00e9 0100 free 645 fee1 807d f701 c645 f800 c645 f901 c645 fa0 c645 460 0000 c9c4 5400 750f c765 48bf 6e08 0460 000 c645 f701 c645 f800 0000 60c9 df80 0000 c645 f781 c64 48bf 807d fc00 7410 807d c645 900 c645 fa04 807d 48bf 6e08 0760 0000 807d ff00 730a c765 48bf 6e08 740f c705 48bf 8e08 fc00 7416 807d ff00 740f c705 48b 0000 00e9 9900 0000 c645 0600 0000 e99e 0000 00e9 9900 000 c645 f900 c645 fa05 807d f701 c645 f800 c645 f900 fe00 750a c705 48bf 0c08 fd00 7410 807d fe00 750a c705 48b fc00 750a c705 48bf 0e08 0800 0000 807d fc00 fe00 7506 807d ff00 740c 0900 6000 807d fe00 7506 0600 6000 eb4b eb49 c645 c705 48bf e88 0600 6000 c645 f901 c645 fa02 807d f701 c645 f800 c645 f901 5dc3 5589 e5c7 0540 bf0e 00b8 5400 0000 5dc3 5589 e5c7 054 5dc3 5589 e5c7 0812 0000 0008 4800 0000 3000 00b8 4500 0000 5dc3 0540 bf0c 0820 0000 00b 000 5f0e 0821 0000 0058 5800 5589 e5c7 0540 bf0e 082 USE 00bi 25c7 6540 bf6e 0822 6000 0000 5dc3 5589 5c7 654 082 5dc3 5589 e583 ec10 c705 00b8 4900 0000 5dc3 558 0000 a148 bf0e 0883 f809 48bf 6e08 0160 0000 a148 3b04 8548 e10b 08ff e0c6 0f87 0002 0000 8b04 8548 30c6 45f9 00c6 45fa 00c7 45f7 00c6 45f8 00c6 45f9 00c6 3000 00c9 d901 0000 c645 0548 bf0e 0802 0000 00e9 d901 :645 f900 c645 fa01 807d f701 c645 f800 c645 f900 c645 18bf 0e08 0300 0000 807d fb00 750a c705 48bf 0e08 0300 fc00 750a c705 48bf 0e08 fb00 7410 807d fc00 750a fc00 7415 807d fb00 740f 0900 0000 807d fc00 7415 807d 3600 6000 c988 0100 60c9 c705 48bf 0c68 0600 6000 c988





APPLICATION: VULNERABILITY DETECTION [SSPREW 2016, with VERIMAG]

Here:

- Focus on use-after-free
- Combine static and DSE



4800	6666	50C3	2287	e5C/	0812	6669	0058	4800	6669	5dc3	558	
0000	6698	4566	0000	1			0820	0000	6668	4566	000	
bf0e	0821	0000	0068	1 En	ntrv n	oint	540	bf0e	0821	0000	00b	
e5c7	6540	bf0e	0822		, P	•	519	e5c7	6540	bf0e	082	
5dc3	5589	e583	ec10	C705	0008	4900	00 0	5dc3	5589	e583	ec1	
0000	a148	bf8e	0883	f809	48bf	6e08	0166	0000	a148	bf8e	088	
8504	8548	e10b	08FF	e0c6	0597	6002	0000	8504	8548	e10b	08f	
00c6	45f9	00c6	45fa	60c7	45f7	00c6	45f8	00c6	45f9	00c6	45f	
0000	60c9	d901	0000	c645	0548	bf0e	0862	0000	60e9	d901	000	
c645	f900	c645	fa01	807d	f701	c645	f86 0	c645	f900	c645	fa⊖	
486f	0e08	0300	0000	807d	fb00	750a	c795	48bf	0e08	0360	000	
TC00	750a	C705	48 D T	0e08	fb00	7410	897d	fc00	750a	⊂765	48b	
1000	7415	807d	1000	7401	0900	6669	807d	fc00	7415	807d	fbΘ	
0600	6669	e988	0100	66e9	c705	4857	0e68	0600	6669	e988	010	
f701	c645	F800	c645	f900	8301	6000	c645	f701	c645	F800	c64	
1C00	7401	C765	4864	6668	c645	fa02	807d	fc00	740f	<705	48b	
0100	66e9	5901	0000	c645	0400	6669	e95e	0100	60e9	5901	000	
c645	f900	645	fa03	807d	f701	c645	F866	c645	f900	c645	faΘ	
1600	7507	C705	48bf	0e08	fd00	7410	807d	fe00	750a	⊂705	48b	
1C00	756	C705	48bf	0e08	0500	6669	807d	fc00	750a	c765	48b	
fe00	740 F	C785	48bf	8e08	0300	6669	807d	fe00	740f	C785	48b	
0100	fron	901	0000	c645	0600	6669	e96e	0100	60e9	0901	000	
C645	rree	045	1001	8070	f701	c645	f800	c645	f901	c645	fa0	
48bf	_	466	0000	c9c4	F400	750f	c765	48bf	6e08	0466	000	
0000	c645	F701	c645	f800	0005	60c9	dfee	0000	c645	f701	c64	
1a04	8070	fc00	7410	807d	c645	X 900	c645	fa04	807d	fc00	741	
48bf	6e08	0766	0000	8670	ff00	75Qa	c765	48bf	6e08	0760	000	
ff00	740f	c785	48bf	6e08	fc00	7416	807d	ff00	740f	C705	48b	
0000	00e9	9900	0000	c645	0600	6666	e99e	0000	60e9	9966	000	
c645	1900	C645	1a05	8070	f701	c645	F800	c645	f900	c645	fa0	
fe00	750a	C785	48bf	6c08	fd00	7410	807d	fe00	750a	c765	48b	
fc00	750a	c705	48bf	0e08	0800	6669	807d	fc00	750a	C785	48b	
†e00	7506	807d	++00	740c	0900	6669	807d	fe00	7506	807d	ff0	
0600	6669	eb4b	eb49	C645	c705	48bf	0e08	0600	6000	eb4b	eb4	
c645	t901	C645	†a02	807d	f701	c645	f 860	c645	f901	c645	faΘ	
5dc3	5589	e5c7	0540	bf0e	00b8	5400	0000	5dc3	5589	e5c7	054	
1800	6669	5dc3	5589	e5c7	0812	6669	8000	4800	6665	5dc3	558	
9996	6698	4566	0000	Sdc3	0540	bf0e	<u>082</u> 0	0000	661		000	
of0e	0821	0000	0058	5800	5589	e5c7	0540	bf0e	682 U	ise 🕨	<u>өөы</u>	
25c7	0540	bf0e	0822	6669	0000	5dc3	5589	\$5c7	054		082;	
5dc3	5589	e583	ec10	C705	0008	4900	0000	5003	558	e583	ec10	
9000	a148	bf6e	0883	f809	48bf	8e08	0166	0000	a148	bf8e	088	
3604	8548	e10b	08FF	e0c6	0f87	6002	0000	8504	8548	e10b	08F	
90c6	45f9	00c6	45fa	60c7	45f7	60c6	45f8	00c6	45f9	00c6	45fi	
9000	60c9	d901	0000	C645	0548	bf0e	0802	0000	60e9	d901	000	
:645	f900	c645	fa01	807d	f701	c645	f860	c645	f900	c645	fa0:	
186F	0e08	0360	0000	807d	F600	750a	c705	48bf	0e08	0360	000	
Fc00	750a	⊂7 65	48bf	6e08	fb00	7410	807d	fc00	750a	⊂705	48b	
Fc00	7415	807d	fb00	740f	0900	6669	807d	fc00	7415	807d	fboi	
9696	6669	e988	0100	60c9	c705	48bf	0e68	0600	6669	e988	010	





KEY IDEAS (Josselin Feist)



A Pragmatic 2-step approach

- Static: scale, not complete, not correct
- Symbolic: correct, directed by static
- Combination: scalable and correct

4800	0000	5dc3	2288	esc/	0812	0000	0058	4800	0000	5dc3	558
0000	0068	4500	0000	-			820	0000	0068	4500	000
bfΘe	0821	0000	0068	4 Er	ntrv n	oint	540	bf 0e	0821	0000	eeb
e5c7	0540	bf0e	0822	<u>ا ا</u>	illy p	onn	519	e5c7	0540	bf 0e	082
5dc3	5589	e583	ec10	C705	66b8	4900	00.0	5dc3	5589	e583	ec1
0000	a148	bf0e	0883	f809	48bf	0=08	0100	0000	a148	bf0e	088
8604	8548	e10b	08 <u>ff</u>	e0c6	0407	0000	0000	8604	8548	e10b	08f
00c6	45f9	00c6	45 Fa	00c7	45 F7	00-6	45.59	00-6	0.540	00-6	454
0000	00c9	d901	0000	c645	0549	bfaa	4516	0000	00.09	doni	000
c645	f900	C645	fa01	8074	0.140	CE 45	fede	CE45	foee	CAL	f 20
48bf	0e68	0300	0000	807d	fbee	7585	6745	49hF	0000	200	000
fc00	750a	C725	48bf	0e08	fbee	7410		fc 88	7500	6705	495
fc00	7415	807d	fb00	740f	0000	0000	274	6-00	744	0074	460
0600	0000	e988	0100	00e9	e705	40	0.00	0000	0000	-000	010
f701	c646	f 800	c645	f900	0.705	9000	0000	£701		£900	010
fc00	746f	c705	48bf	DCD8	2645	£203	0074	f=00	2.45	705	405
0100	00e9	5991	0000	c645	0400	0002	0070	0100	0000	C705	400
c645	900	645	fa03	807d	£701	0000	6950 feee	0100	foee	5901	520
feee	750	c705	48bf	6e08	fdee	7410	207d	E CO	7500	C045	405
fce	75	c705	48bf	0e08	0500	00000	8070 807d	Zaa	7500	C705	460
fello	746 F	c705	48bf	0e08	0300	0000	8070 807d	100	7405	-705	480
0100		901	0000	c645	0.500	0000	-074	0100	00-0	0001	480
645	free	045	Fe01	807d	5701	0000	1000	-645	5001	645	5-0
48bf		400	0000	-0-1	170		700	4055	0-00	0400	000
0000	c645	£701	C645	f800	0000	7501	105	4601	0008	£701	000
£604	807d	fc00	7410	807d	0000	0000	0100	£204	0074	fc00	741
48bf	0e08	0700	0000	807d	£645	7000	0705	1004	8070	0700	000
ffee	740f	c705	48bf	60.08	fc.00	744	607d	Con Con	7405	6705	495
0000	00e9	9900	0000	C645	0600	0000	0070	00000	0000	0000	400
c645	f966	c645	fa05	8070	6701	C645	feee	664	foee	C645	f 20
fe00	750a	c705	48bf	0e08	100	7410	807d	6-00	1900	-705	485
fc00	750a	c705	48bf	6e08	0000	0000	8074	5-00	Xe-	-705	400
fe00	7586	807d	FF00	740c	0000	0000	074	f=00	700	0074	460
0600	0000	eb4b	cb49	c645	e705	ODEE	1.00	0600	0000	ab/b	ahd
c645	f901	c645	faez	807d	£703	S	4000	0000	f0000	C040	604
Sele 3	55.80	-5-7	0540	h.£0-	0010	EAN	1000	C045	5500	33	05.44
1800	00.00	Sdc3	5580	=5c7	0008	5400	0000	4999	3389	esox.	5591
3000	0000	4500	3365	Edc3	0612	6666	Nopa.	4800	0000	2002	228
5000	0000	4500	aabe	5005	0540	Drue	3	6666	000		000
DFC7	0621	bf@o	00000	6666	5589	esc/	054	Dree	064	ise	0001
Edc2	5590	OF92	0022	C705	0000	5003	2282	Sc!	054		0821
3003	-140	6282	0000	6000	0008	4900	0000	2004	228	e583	ecit
2600	0540	-10-	1190	-0-6	4801	0008	0100	0000	8148	brue	088.
30-6	45 40	00-6	465-	00-7	0187	0002	0000	BDUA	8548	e105	081
3000	4519	door	4310	66C7	4517	0006	4518	0006	519	0006	4514
5000	foee	0901	5000	0074	6548	Dree	6862	0000	0000	0501	0001
4055	1900	0200	1001	8070	1701	C645	1866	C645	1966	C645	Ta0:
4801 Ec.00	75.00	0300	4000	8070	TDOO	750a	C705	48bf	0668	0300	0001
6.00	7508	C/05	4801	0008	TDOO	7410	807d	TCOO	750a	C705	48b
FC00	74-15	807d	1000	740f	0900	0000	807d	fc00	7415	807d	±Ρ0(
- N.C. (N.C.)			A1 1 (A) (A)	HHA-9	c705	49hF	8668	0600	0000	~000	0100





EXPERIMENTAL EVALUATION

GUEB + manual analysis [j. comp. virology 14]

- tiff2pdf : CVE-2013-4232
- openjpeg : CVE-2015-8871
- gifcolor : CVE-2016-3177
- accel-ppp
- GUEB + BINSE/SE [ssprew16]
 - Jasper JPEG-2000 : CVE-2015-5221

On these examples:

- Better than DSE alone
- Better than blackbox fuzzing
- Better than greybox fuzzing with no seed

	0000	5dc3	2288	e5c7	0812	0000	0058	4800	0000	5dc3	558
	0058	4500	0000	1			820	0000	00b8	4500	000
	0821	0000	0058	1 Er	ntrv p	oint	540	bfΘe	0821	0000	00b
	0540	b†0e	0822	•			5 9	e5c7	0540	bf0e	082
	5589	e583	ec10	C705	0658	4900	00 0	5dc3	5589	e583	ec1
	a148	bf0e	0883	1809	48bf	0e08	0100	0000	a148	bf0e	088
	8548	e10b	08ff	e0c6	0f87	0002	0000	8604	8548	e10b	08 f
	4519	00~6	45fa	00c7	45f7	00c6	45f8	00-6	1019	00 :6	45 f
	00c9	d901	0000	c645	0548	bfee	-100 K	0000	00c9	d9.51	996
	f900	C645	fa01	8074	1701	C645	f860	c645	f900	C 45	fa0
	0e08	0300	0000	8070	fbee	750a	c755	48bf	0e08	3 00	000
	750a	C705	4861	0e08	fb00	7410	897d	fc00	750a	C705	48b
	7415	207 d	1 600	740f	0900	0000	807d	fc00	741	807d	fb0
	0009	c988	0100	00e9	c705	4857	0e68	0600	00.00	e988	010
	C646	f 800	C645	f900	8301	0000	c645	f701	645	f 800	c64
	740f	C705	48bf	0008	c645	fa02	807d	fc00	40f	c705	48b
	90e9	5991	0006	c645	0400	0000	e95e	0100	00e9	5901	000
1	F966	C645	fa03	807d	f701	C645	f860	C646	f900	C645	fa0
1	750	C705	48bf	0e08	fdee	7410	807d	felle	750a	C705	48b
	75	C705	48bf	0e08	0500	0000	807d	f£00	750a	C705	48b
	746 F	c705	48bf	0e08	0300	0000	807d	fe00	740f	c705	486
	6	901	0000	c645	0666	0000	egge	0100	00e9	0901	000
	Tree	.045	Fa01	807d	f701	643	T860	c645	f901	c645	fa0
	-	400	0000		NUMBER	7501	<705	48bf	0c08	0400	000
_	C645	£701	C615	f800	0000	00c9	d100	0000	c645	f701	C64
	807d	fc00	7410	807d	c645	K 900	c645	fa04	807d	fc00	741
	0e08	0700	0000	8070	ff00	750a	c705	48bf	0e08	0700	000
	740f	C705	48bf	00.08	fc00	7416	807d	N E00	740f	C705	48b
	00e9	9900	0000	C645	0600	0000	e99e	0000	00e9	9900	000
	1966	C645	1405	8070	£701	C645	f860	c643	f900	C645	fa0
_	750a	c705	4861	0e08	FN00	7410	807d	fe00	250a	c705	486
	-		Sec.	0e08	0800	0000	807d	fc00	7 30 a	c705	48b
ł		State of the	N NOT	740c	0900	0000	807d	fe00	7566	807d	ff0
-	100 1	Contractory of	1	c645	c705	ABbf	6c68	0600	0000	cb4b	eb4
E		SUN C		807d	f701	C045	f866	C645	f901	645	fa0
i	1	-		bf0e	0058	5400	0000	5dc3	5589	e507	0544
2		1	The second	e5c7	0812	0000	8400	4800	0000	5dc3	558!
ŝ				5dc3	0540	bf0e	0.20	0000	00L		0.001
V	CALCUMPTION OF	150	West-	5800	5589	e5c7	054	bf 0e	082 L	ise 🕨	0051
h	1000	100	X	6666	0000	5dc3	5589	\$5c7	054		082
	(and		In	c705	00b8	4900	0000	3003	558	e583	ec10
	AL.	EN I	V A	f809	48bf	0e08	0100	0000	a148	bf0e	088
	Sec.		λ / λ	e0c6	0f87	0002	0000	8b0-	8548	e10b	08f:
3		X	ARE	00c7	45f7	00c6	45f8	00c6	USF9	00c6	45f;
1	7360	6 B	13	C645	0548	bf0e	080Z	0000	0000	1000	0001
		NY I	A.	807d	f701	C645	f800	C645	f900	C645	fa0:
	21	10		807d	fb00	750a	c705	48bf	0e68	0300	0001
-	PE	Sec.	SWIE .	0e08	fb00	7410	807d	fc00	750a	C705	48b1
-	71			740F	0900	0000	807d	fc00	7415	807d	fb0(
	J. L			60e9	c705	48bf	0e68	0600	0000	e988	010(
	11	123	ON C								
	1 17	Contraction of the									

4800

bf0e e5c7 5dc3 0000 8b04 00c6 0000 c645 48bf fc00 fc00

0600 f701 fc00 0100

c645

9000 1801

ff00 0000 c645 fe00





APPLICATION: MALWARE DEOBFUSCATION [S&P 2017, with LORIA]

APT: highly sophisticated attacks

- Targeted malware
- Written by experts
- Attack: 0-days
- Defense: stealth, obfuscation
- Sponsored by states or mafia

The day after: malware comprehension

- understand what has been going on
- mitigate, fix and clean
- improve defense



USA elections: DNC Hack







Goal: help malware comprehension

- Reverse of heavily obfuscated code
- Identify and simplify protections



list ^{Ceatech}

REVERSE CAN BECOME A NIGHTMARE (OBFUSCATION)





EXAMPLE: OPAQUE PREDICATE

Constant-value predicates

(always true, always false)

• dead branch points to spurious code

• goal = waste reverser time & efforts

eg: **7y² - 1 ≠ x**²

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

Т

	¥	
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< td=""><td>d_addr></td></dead<>	d_addr>





EXAMPLE: STACK TAMPERING

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

- hide the real target
- return site may be spurious code

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



list Ceatech

STANDARD DISASSEMBLY TECHNIQUES ARE NOT ENOUGH





Dynamic analysis

robust vs obfuscation

jmp

eax

CARNOT

universite

too incomplete

list

DYNAMIC SYMBOLIC EXECUTION CAN HELP (Debray, Kruegel, ...)




YET ... WHAT ABOUT INFEASIBILITY QUESTIONS?

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?

And more

- does this malicious ret always go there?
- does this expression always evaluate to 15?
- does this self-modification always write this opcode?
- does this self-modification always rewrite this instr.?



Not addressed by DSE

Cannot enumerate all paths





OUR PROPOSAL: BACKWARD-BOUNDED SYMBOLIC EXECUTION

Insight 1: symbolic reasoning

- precision
- But: need finite #paths

Low FP/FN rates in practice

• ground truth xp



- pre_k(c)=0 => c is infeasible
- finite #paths
- efficient, depends on k
- But: backward on jump eax?

Insight 3: dynamic partial CFG

- solve (partially) dyn. jumps
- robustness



- can miss infeasibility
- why: k too small (miss Λ-constraints)

False positive (FP)

- wrongly assert infeasibility
- why: CFG too partial (miss V-constraints)



mov edx, 0

call XX

add [esp], 9

cmp edx, [esp+4]

inc edx



FORWARD & BACKWARD SYMBOLIC EXECUTION





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





EXPERIMENTAL EVALUATION







CONTROLLED EXPERIMENTS

- Goal = assess the precision of the technique
 - ground truth value
- Experiment 1: opaque predicates (o-llvm)
 - 100 core utils, 5x20 obfuscated codes
 - k=16: 3.46% error, no false negative
 - robust to k
 - efficient: 0.02s / query
- Experiment 2: stack tampering (tigress)
 - 5 obfuscated codes, 5 core utils
 - almost all genuine ret are proved (no false positive)
 - many malicious ret are proved « single-targets »

	k	OP (5556)		Genuine (5183)		TO Error rate		Time	avg/query
	ĸ	ok	miss	ok	miss		(FP+FN)/Tot	(s)	(s)
			(FN)		(FP)		(%)		
	2	0	5556	5182	1	0	51.75	89	0.008
	1	002	4650	5150	20	0	42 61	96	0.009
						14	9	120	0.011
very precise results						S	152	0.014	
						5	197	0.018	
Sooms officient						5	272	0.025	
				CIU				384	0.036
	32	5552	4	4579	604	25	5.66	699	0.065
	40	5548	8	4523	660	39	6.22	1145	0.107
	50	5544	12	4458	725	79	6.86	2025	0.189

	runtime genuine			runtime violation			
Sample	#mat t	proved	proved	#rot t	proved	proved	
	#ret .	genuine	a/d	#ret '	a/d	single	
obfuscated pr	rograms						
simple-if	6	6	6/0	9	0/0	8	
bin-search	15	15	15/0	25	0/0	24	
bubble-sort	6	6	6/0	15	0/1	13	
mat-mult	31	31	31/0	69	0/0	68	
huffman	19	19	19/0	23	0/3	19	
non-obfuscate	non-obfuscated programs						
ls	30	30	30/0	0	-	-	
dir	35	35	35/0	0	-	-	
mktemp	21	20	20/0	0	-	-	
od	21	21	21/0	0	-	-	
vdir	49	43	43/0	0	-	-	



CASE-STUDY: PACKERS





Packers: legitimate software protection tools (basic malware: the sole protection)



CASE-STUDY: PACKERS (fun facts)

Several of the tricks detected by the analysis

		_	Cor In Asr dek
idium	OP in ACProtect		10043a9 mov [ebp+0x3a8], eax
ACK Upack	1018f7a js 0x1018f92	OP in Armadillo	10043af popa 0x10043bb
Packman	1018f7c jns 0x1018f92	10330ae xor ecx, ecx	10043b0 jnz 0x10043ba
stector SVk	<pre>(and all possible variants ja/jbe, jp/jnp, jo/jno)</pre>	10330b0 jnz 0x10330ca	Enter SMC Layer 1
Crypter	575757757757		10043ba push 0x10011d7
oleBox		CST in ACProtect	10043bf ret
Yoda's Protector Pack		1001000 push 16793600	OP (decov) in ASPack
BoxedApp	CST in ACProtect	1001005 push 16781323	
gma Themida	1004328 call 0x1004318	100100a ret	10041c0: cmp bl, 0x1
Pack	1004318 add [esp], 9	100100b ret	ZE = 0 ZE = 1 at runti
	100431c ret	100/162.	mp_0x100416d 1004105: inc_[obp+0x0
		[]	
		Séh	astien Bardin ISSISP 2017 79

74

CST in ASPack



CASE-STUDY: THE XTUNNEL MALWARE (part of DNC hack)



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

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

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



list ceatech

SECURITY ANALYSIS: COUNTER-MEASURES (and mitigations)

- Long dependecy chains (evading the bound k)
 - Not always requires the whole chain to conclude!
 - Can use a more flexible notion of bound (data-dependencies, formula size)
- Hard-to-solve predicates (causing timeouts)
 - A time-out is already a valuable information
 - Opportunity to find infeasible patterns (then matching), or signatures
 - Tradeoff between performance penalty vs protection focus
 - Note: must be input-dependent, otherwise removed by standard DSE optimizations
- Anti-dynamic tricks (fool initial dynamic recovery)
 - Can use the appropriate mitigations
 - Note: some tricks can be circumvent by symbolic reasoning

Current state-of-the-art

- push the cat-and-mouse game further
- raise the bar for malware designers

Also

- « Probabilistic obfuscation »
- Covert channels





- Why binary-level analysis?
- Some background on source-level formal methods
- The hard journey from source to binary
- A few case-studies
- Conclusion





SUMMARY

	Feasibility	Infeasibility	Efficient	Robust
Static syntactic	Х	Х	OK	Х
Dynamic		Х	OK	OK
DSE	OK	Х	Х	OK
Static semantic	Х	OK	Х	Х
BB-DSE	Х	OK (fp,fn)	OK	OK





CONCLUSION

- Semantic analysis can change the game of binary-level security
 - Current syntactic and dynamic methods are not enough
 - [complement existing approaches and help the expert, not replace everything]
 - Explore more, Prove invariance, Simplify

- Yet, challenging to adapt from source-level safety-critical
 - Need robustness, precision and scale!!
 - « Correct-enough » and « Complete-enough » are enough (room for better definition!)
 - DSE much easier to adapt than AI
 - New challenges and variations, so much to do







FUTURE DIRECTION





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