Advanced Anti-Deobfuscation

Bjorn De Sutter



ISSISP 2017 – Paris

About me



- Research domain: system software
 - compilers, binary rewriting tools, whole program optimization (binary & Java), virtualization, run-time environments
 - improve programmer productivity by means of automation
 - apply tools for different applications
 - obfuscation, diversity, mitigating side channels and fault injection, ...
 - protect against exploitation of vulnerabilities (multi-variant execution)
 - generating code for accelerators
- Also worked/spent time at
- Interrupts enabled





Lecture Overview

- 1. Basic Attacks
 - attacks on what?
 - basic attack tools & techniques
 - 2. Defenses
 - anti-anything
 - 3. Advanced Automated Attacks
 - generic deobfuscation
 - symbolic execution
 - 4. Defenses
 - anti-even-more

What is being attacked?

Asset category	Security Requirements	Examples of threats
Private data (keys, credentials, tokens, private info)	Confidentiality Privacy Integrity	Impersonation, illegitimate authorization Leaking sensitive data Forging licenses
Public data (keys, service info)	Integrity	Forging licenses
Unique data (tokens, keys, used IDs)	Confidentiality Integrity	Impersonation Service disruption, illegitimate access
Global data (crypto & app bootstrap keys)	Confidentiality Integrity	Build emulators Circumvent authentication verification
Traceable data/code (Watermarks, finger-prints, traceable keys)	Non-repudiation	Make identification impossible
Code (algorithms, protocols, security libs)	Confidentiality	Reverse engineering
Application execution (license checks & limitations, authentication & integrity verification, protocols)	Execution correctness Integrity	Circumvent security features (DRM) Out-of-context use, violating license terms

What is being attacked?



- 1. Attackers aim for assets, layered protections are only obstacles
- 2. Attackers need to find assets (by iteratively zooming in)
- 3. Attackers need tools & techniques to build a program representation, to analyze, and to extract features
- 4. Attackers iteratively build strategy based on experience and confirmed and revised assumptions, incl. on path of least resistance
- 5. Attackers can undo, circumvent, or overcome protections with or without tampering with the code

Basic Attack Techniques

- Static attack steps: without executing the code
 - symbolic information
 - graph representations of program
- Dynamic attack steps: observing execution
 - all kinds of hooks
 - start and intervene at interfaces
 - observe features and patterns of program execution (traces)
- Hybrid attack steps: combination of both
 - e.g.: build graphs of (unpacked) code observed during execution

- IDA Pro
- Binary Ninja
- angr
- Far from perfect
 - incomplete disassembly
 - incorrect graphs (control flow, call graphs)
- Flexible and interactive
 - linear sweep, recursive descent, heuristical and manual disassembly
 - GUI
 - code annotation
 - plug-ins and scripts

IDA - C:\Users\Daniel\Desktop\arch.idb (arch - Copy.exe)		
File Edit Jump Search View Debugger Options Windows Help		
📑 📻 🗔 🗇 🔿 🛱 🏝 🏝 斗 💫 🖬 🖾 🥥 📾 🏙 🕼 📌 🖛 🗶 🕨 💷 💭 Windbg debugger 🔷) 🗊 🕈 📽	
		•
F Functions window 🗗 🗶 🥂 IDA View-A, Exports, Imports 🛛 🧿 Hex View-A 🖂 🖪 Struct	ctures 🗵 🗄 Enums 🗵	
Function name 🔺 🗈 IDA View-A	🗗 🗙 📝 Exports	8 ×
f start f start f sub_0_401100 f short loc_0_401859 f sub_0_401730 f sub_0_401700 f sub_0_401700 mov ecx, [ebp+var_18] f sub_0_402520 mov ecx, [ebp+var_18] sub_0_4026D0 mov ecx, [ebp+var_28] mov ecx, [ebp+var_30] mov ecx, [ebp+var_30] kince for 103 mov ecx, [ebp+var_30] mov ecx, [ebp+var_30] sub eax, [ebp+var_14] liea ebx, [edx+ecx*4] mov ecx, [edx+ecx*4] f sub eax, [edx+ecx*4]	Name	► • • • • • • • • • • • • •
Image: Constraint of the second se	sub [ebp+var_10], edi F13C _dll_crt0 jmp loc_0_4017F2 F140 _exit sub_0_4017C0 endp Minoutorial _impure_ Minoutorial 0040F148 abort _minoutorial Minoutorial Minoutorial Minoutorial _exit Minoutorial Minoutorial _exit _impure_ Minoutorial 0040F148 abort _minoutorial Minoutorial Minoutorial _minoutorial _minoutorial Minoutorial _minoutorial _minoutorial _minoutorial Minoutorial _minoutorial _minoutorial _minoutorial Minoutorial _minout	₽0 ptr nternal ₽ ×
Python		
AU: idle Down Disk: 21GB		

- Static & hybrid attacks
- Rely on many underlying assumptions
- Library detection
 - F.L.I.R.T
- Diffing tools
 - BinDiff
- Custom tools
 - detect patterns
 - undo obfuscations
 - data flow analysis
- Supports code editing
- Interfaces with (remote) debuggers

	\sim · · ·	\mathbf{n}	<u> </u>										
ি IDA - E:\home\bcoppens\priva	ate\phd\presentation\before												
File Edit Jump Search View [Debugger Options Windows I	Help											
	🖦 🆦 🧎 🛵 🤅 🗛 🥥 :	at at i	🕇 🖈 🐲 🛋 🗙 🤅 🕨 🔲 🗖 No debugge	er 💌 😢 🛃	: 🗊 🕈 🏹								
				· · · · · · · · · · · · · · · · · · ·							þ. 4		~
Functions window	8 ×		A View-A	🕥 Statistics 🖂 🕥	Primary Linmatched 🖂 🛛 🕥	Secondary Linmatched 🖂 🛛 🖸 H	Hey View-A 🖾 🚺 Structures 🖾 🗮	Enums 🕅 🔯 Imp	orts	Exp	orts 🔲		
Eurotion name	Segment	cimilarit	confide change 50 primary		EA cocordary				atched b bac	chlock back	block/ matched	inct inctructions	a di in 🔥
init proc	joit lot	1,00	0.33 0000000 0.30	Jeğin Jeğin	U0UDU04U	_to_iter_begin	name nasi macuning		1 L L			anst instructions	· : 🕋
f start	tevt	1.00	0.99 08056280 _dl_receive	e_error	08056260	_dl_receive_error	name hash matching		1 1	1	27	27	
f call group start	text	1.00	0.99 08056710 _dl_debug_	_printf	080566F0	_dl_debug_printf	name hash matching		1 1	1	14	14	
f do dobal dtors aux	text	1.00	0.99 08056740 _dl_debug_	_printf_c	08056720	_dl_debug_printf_c	name hash matching		1 1	1	14	14	
frame dummy	text	1.00	0.99 080568E0 _dl_initial_e	error_catch_tsd	080568C0	_dl_initial_error_catch_tsd	name hash matching		1 1	1	5	5	
7 next user	text	1.00	0.99 0806BAE0gconv_g	get_modules_db	0806BAC0	gconv_get_modules_db	name hash matching		1 1	1	5	5	
7 reset user	text	1.00	0.99 0806BAF0gconv_g	get_alias_db	0806BAD0	gconv_get_alias_db	name hash matching		1 1	1	5	5	
f name and password match	.text	1.00	0.99 08072D40gconv_g	get_cache	08072D20	gconv_get_cache	name hash matching		1 1	1	5	5	
T check password	.text	1.00	0.99 08073790gconv_re	release_shlib	08073770	gconv_release_shlib	name hash matching		1 1	1	11	11	
7 main	.text	1.00	0.99 0807BBA0 vsscanf		0807BB80	vsscanf	name hash matching		1 1	1	30	30	
📝 libc start main	.text	1.00	0.99 080868D0 localtime		08086880	localtime	name hash matching		1 1	1	11	11	
📝 check one fd	.text	1.00	0.99 08087010 timelocal		08086FF0	timelocal	name hash matching		1 1	1	12	12	
📝 libc check standard fds	.text	1.00	0.99 08093980 _dl_reloc_b	bad_type	08093960	_dl_reloc_bad_type	name hash matching		1 1	1	29	29	
flibc_csu_init	.text	1.00	0.99 free_mem		08095CB0	free_mem	name hash matching		1 1	1	13	13	
🖌 _libc_csu_fini	.text	1.00	se_mem_	_5	08095EC0	free_mem_5	name hash matching		1 1	1	12	12	
ferrno_location	.text	1.00	V00	ic .	08096190	.term_proc	name hash matching		1 1	1	11	11	
f exit	.text				0804B100	valloc	name hash matching		17 17	17	64	64	
📝cxa_atexit	.text		ivalb.		0804B1D0	pvalloc	name hash matching		15 15	15	59	59	
📝new_exitfn	.text	_ 10	mallopt		08067D20	_IO_file_xsgetn	name hash matching		27 27	27	128	128	
📝 malloc_init_state	.text				0805EF80	parse_one_spec	name hash matching		13 113	3 113	437	437	
📝 malloc_atfork	.text	.00	exit		08049380	arena_get2	name hash matching		+2 42	42	129	129	
📝 free_atfork	.text				USU/AFEU	parse_one_spec_U	name hash matching		06 106	o 106	407	407	
📝 ptmalloc_lock_all	.tex)	.00	dl sysiafo inti	-80	0805A310	gettextlex	name hash matching		35 35	36	98	98	
📝 ptmalloc_unlock_all	.te;		y		08046820	mailopt	name nash matching		26 26	26	80	80	
📝 ptmalloc_unlock_all2	.te 1	00	mál LOPE		0804CDD8	_exit	name nash matching		1 1	1	3	3	
f next_env_entry	.te		macore		08040823	_u_systito_incou	name hash matching		25 25	25		2	
📝 ptmalloc_init_minimal	.ti 1	00	dLaux init		08040580	d aux init	name hash matching		11 17	20	24	24	
📝 ptmalloc_init	.te 🗖		TorTooxTune		00040300		name hash matching		02 0 ²	02	264	364	
📝 new_heap	.te	00	evecopf				name hash matching		5 5	52	30	39	
f grow heap	- Ha		Syscorii				name has in matching		5 5	5	50	50	2
		00	check pacewor	ed .)	_				
Contractor and a second second	U	1.90	check_basswor										
													в .
; Source File : 'dl-version.c	c'												^
01:35:26 sorting instructions	15												
01:35:26 reconstructing flows	graphs												
01:35:26 reconstructing funct	tions												
01:35:26 simplifying function	ns												
01:35:26 writing	roccoping												
01:35:26 CProtocolBufferWrite	er::write "C:/DOCUME~1/St:	ijn/LOCA	LS~1/Temp/zynamics/BinDiff/240/primar	ry/before.BinExport"									
01:35:27 before: 1.33 seconds	ls processing, 0.50 second:	s writir	a										
01:35:31 8 402 seconds for a	b functions with 82098 in: whorts	structio	ns in 1.85 seconds										
01:35:33 2.043 seconds for ma	atching.												_
01:36:10 Sending result to B:	inDiff GUI												~
Python													
AU: idle Down Disk:	1GB												





• Decompiler

<u>File A</u> nalyse <u>Vi</u>	ew <u>H</u> elp	
Instructions	0 ×	C++
8048094: push 8048095: mov 8048097: sub 804809a: cmp 804809e: jnz 80480a0: mov 80480a3: jmp 80480a3: jmp 80480a3: mov 80480a8: mov 80480a8: mov 80480ba: mov 80480b0: mov 80480b0: mov 80480b0: mov 80480b6: mov 80480b6: mov 80480b6: mov 80480b6: ret	ebp ebp, esp esp, 0x18 [ebp + 0xc]:32, 0x0 0x80480a5 eax, [ebp + 0x8]:32 0x80480c1 eax, [ebp + 0x8]:32 edx, eax edx, 0x1f [ebp + 0xc]:32 eax, edx [esp + 0x4]:32, eax eax, [ebp + 0xc]:32 [esp]:32, eax 0x8048094	<pre>int32_t gcd(int32_t arg1, int32_t arg2) { int32_t eax1; if (arg2 != 0) { eax1 = gcd(arg2, arg1 % arg2); } else { eax1 = arg1; } return eax1; }</pre>

Debuggers - 1

- GDB
- OllyDbg
- Scriptable
- Support tampering
 - alter processor state (incl. program counter)
 - alter memory contents
 - alter code
 - used for out-of-context execution

Debuggers - 1

🔆 OllyDbg - 6ceed14d16b89860eabe9eadd918a2bf.exe - [CPU - main thread, module 6ceed14d]					
C File View Debug Plugins Options Window Help					
→	 ?				
	▲ Registers (FPU) < < < <				
004220848 . 68 FF PUSH -1 004220848 . 68 FF PUSH -1 004220847 . 68 78054300 PUSH 6ceed14d.00430578 004220848 . 64:81 0000000 MOV EAX,DWORD PTR FS:[0] SE handler installation 004220884 . 64:81 0000000 MOV EAX,DWORD PTR FS:[0] SE handler installation 004220884 . 64:81 0000000 MOV EAX,DWORD PTR FS:[0] SE handler installation 004220884 . 64:8925 00000 MOV DWORD PTR FS:[0],ESP SE handler installation 004220885 . 53 PUSH EAX SE handler installation	ERX 00000000 ECX 0012FFB0 EDX 7C91E4F4 ntdll.KiFastSystemCallRet EBX 7FFDE000 ESP 0012FFC4 EBP 0012FFFC4 ESI FFFFFFF EDI 7C920208 ntdll.7C920208				
0042D8C6 .56 PUSH ESI 0042D8C7 .57 PUSH EDI 0042D8C8 .8965 E8 MOV DWORD PTR SS:[EBP-18],ESP 0042D8C8 .FF15 48004300 CALL DWORD PTR DS:[<&KERNEL32.GetVersion 0042D8C8 .33D2 XOR EDX,EDX 0042D8C8 .84D4 MOV DU,AH 0042D8D5 .8915 288D4300 MOV DWORD PTR DS:[438D28],EDX 0042D8D5 .8915 288D4300 MOV DWORD PTR DS:[438D28],EDX 0042D8D5 .81E1 FF000000 AFF	EIP 0042D8A5 6ceed14d. <moduleentrypoint> C 0 ES 0023 32bit 0(FFFFFFF) P 1 CS 001B 32bit 0(FFFFFFF) A 0 SS 0023 32bit 0(FFFFFFF) Z 1 DS 0023 32bit 0(FFFFFFF) S 0 FS 003B 32bit 7FFDD000(FFF) T 0 GS 0000 NULL D 0 0 0 LoctEmp EPPOR SUCCESS (00000000)</moduleentrypoint>				
0042D8E3 . \$900 24804300 MOU DWORD PTR DS:[438D24],ECX 0042D8E0 . CIE1 08 SHL ECX,8 ADD ECX ECX 0042D8E4 . 890D 208D4300 MOU DWORD PTR DS:[438D20],ECX SMR EAX,10 0042D8F7 . CIE8 10 SHR EAX,10 EAX,10 SHR EAX,10 0042D8F7 . 33F6 XOR ESI,ESI PUSH ESI EAX PUSH ESI EAX 0042D8F7 . 56 PUSH ESI CALL 60e42DEFC 6042DFC FS F8050000 CALL 60e42DEFC EA FS F00F ECX FDV 0042D8F6 . 55 . 56 PUSH ESI EA F00F ECX FDV FDV 0042D964 . 59 . 76 TEST EDX FDV FDV FDV 0042D8F7 . 56 . 700F ECX FDV FDV FDV FDV 0042D8F6 . 59 . 700F ECX FDV FDV FDV FDV 0042D8F6 . 59 . 700F ECX	0 0 Lasterr ERKUF_SUCLESS (00000000) EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE) ST0 empty 0.0 ST1 empty 0.0 ST2 empty 0.0 ST3 empty 0.0 ST4 empty 0.0 ST5 empty 0.0 ST6 empty 0.0 ST6 empty 0.0 ST7 empty 0.0				
EBP=0012FFF0	S 2 1 0 E S P U O 2 D 1 FST 0000 Cond 0 0 0 Err 0 0 0 0 0 0 0 (GT) FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1				
Cceed14d. <moduleentrypoint></moduleentrypoint>	MILEERA 70817067 RETURN to keypel32,70817067				
Haddress Hex dupp Hstill 00431000 00	• •				
Analysing Sceed14d: 73 heuristical procedures, 106 calls to known, 7 calls to guessed functions Paused					

Debuggers - 2

- Used for program understanding
- Used for zooming in on relevant code
 - Continuous iterative refinement of scripts
- Low overhead with hardware breakpoints
- High overhead with software breakpoints
 - Requires tampering

Emulation & Instrumentation

- QEMU
- Pin
- Valgrind
- DynInst
- Itrace
- Used to collect traces
 - To identify patterns and points of interest
- Used like a debugger
 - Iterative refinement of scripts
 - But not interactive

Software Tampering

- Editing the binary
- Alter running process state (CPU, memory)
- Intervene at interfaces
 - system calls
 - library calls
 - network activities
 -
- Custom binaries to invoke library APIs
- Aforementioned tools
- Cheat Engine
 - all kinds of reverse engineering aids (pointer chaining)

struct player

bool visible

struct player

bool visible



((ESP(play())-0x16)+0x4)+0x28

🐔 Change address 🛛 🔀				
Give the new address:				
019FACE0				
🗹 Pointer				
Add pointer Remove				
This pointer points to address 019FACC8	The offset you chose brings it to 019FACE0			
Address of pointer 00087488	Offset (Hex) 18			
This pointer points to address 000874B8	The offset you chose brings it to 000874B8			
Address of pointer 019FAC3C	Offset (Hex) 0			
This pointer points to address 019FAC28	The offset you chose brings it to 019FAC3C			
Address of pointer 019FAB94	Offset (Hex) 14			
This pointer points to address 019FAB88	The offset you chose brings it to 019FAB94			
Address of pointer 005743A0	Offset (Hex)			
OK Cancel				

Lecture Overview

- 1. Basic Attacks
 - attacks on what?
 - basic attack tools & techniques

2. Defenses

- anti-anything
 - 3. Advanced Automated Attacks
 - generic deobfuscation
 - symbolic execution
 - 4. Defenses
 - anti-even-more

Anti-tampering

- Code guards (code integrity)
 - hashes over code regions
- State inspection
 - check for existing invariants
 - inject additional invariants
 - for data integrity and control flow integrity
- Basic control flow integrity
 - check return addresses
 - check stack frames



Remote attestation



The attestator routine is invoked.

Anti-disassembly

- Hide code
 - packers, virtualization, download code on demand, self-modifying code
- Junk bytes
- Indirect control flow transfers
- Jumps into middle of instructions
- Code layout randomization
- Overlapping instructions
- Exploit known heuristics
 - continuation points
 - patterns for function prologues, epilogues, calls, ...

Often, wrong information is worse than no information.

Anti-disassembly examples

Example 1



Anti-decompilation

Exploit semantic gap between source code and assembly code or bytecode

- strip unnecessary symbol information
- rename identifiers (I,I,L,1)
- goto spaghetti
- disobey constructor conventions
- disobey exception handling conventions

Anti-decompilation example



Batchelder, Michael, and Laurie Hendren. "Obfuscating Java: the most pain for the least gain." In *Compiler Construction*, pp. 96-110. Springer Berlin Heidelberg, 2007

Anti-debugging

- Option 1: check environment for presence debugger
- Option 2: prevent debugger to attach
 - OS & hardware support at most one debugger per process
 - occupy one seat with custom "debugger" process
 - make control & data flow dependent on custom debugger
 - anti-debugging by means of self-debugging









Anti-emulation

- Emulators are buggy incomplete
- Virtual environments are not real
- Johanna Rutkowska
 - Blue pill
 - Red pill

Lecture Overview

- 1. Basic Attacks
 - attacks on what?
 - basic attack tools & techniques
 - 2. Defenses
 - anti-anything
 - 3. Advanced Automated Attacks
 - generic deobfuscation
 - symbolic execution
 - 4. Defenses
 - anti-even-more

- no obfuscation-specific assumptions
 - treat programs as input-to-output transformations
 - use semantics-preserving transformations to simplify execution traces
- dynamic analysis to handle runtime unpacking





- *Quasi-invariant locations*: locations that have the same value at each use.
- Their transformations:
 - Arithmetic simplification
 - adaptation of *constant folding* to execution traces
 - consider quasi-invariant locations as constants
 - controlled to avoid over-simplification
 - Control simplification
 - E.g., convert indirect jump through a quasi-invariant location into a direct jump
 - Data movement simplification
 - use pattern-driven rules to identify and simplify data movement.
 - Dead code elimination
 - need to consider implicit destinations, e.g., condition code flags.



Symbolic Execution



effective because most obfuscations implement semantics that do not involve input

Lecture Overview

- 1. Basic Attacks
 - attacks on what?
 - basic attack tools & techniques
 - 2. Defenses
 - anti-anything
 - 3. Advanced Automated Attacks
 - generic deobfuscation
 - symbolic execution
 - 4. Defenses
 - anti-even-more

Anti-taint analysis

- tainting all data with artificial computations
- hiding data dependencies through covert channels
 - time
 - system state
 - anything not normally checked by analysis

Obfuscations with varying, input-dependent behavior (Banescu et al, ACSAC 2016)

```
1 unsigned char *str = argv[1];
2 unsigned int hash = 0;
3 for(int i = 0; i < strlen(str); str++, i++) {
4 hash = (hash << 7) ^ (*str);
5 }
6 if (hash == 809267) printf("win\n");
```

1. RANGE DIVIDER

```
unsigned char *str = argv[1];
1
     unsigned int hash = 0;
2
3
     for(int i = 0; i < strlen(str); str++, i++) {</pre>
4
       char chr = *str;
5
       if (chr > 42) {
6
         hash = (hash << 7) ^ chr;
7
       } else {
         hash = (hash * 128) ^ chr;
8
9
       }
     }
10
     if (hash == 809267) printf("win\n");
11
```

Obfuscations with varying, input-dependent behavior (Banescu et al, ACSAC 2016)

2. INPUT INVARIANTS

- 1. inject extra inputs into programs
- 2. let correct execution depend on invariant properties of those inputs

for example: feed program extra key to decrypt bytecode (how to get these to the user ???)