0046d50 ffld 50ff e857 8ce3 ffff 5c8d 2024 70e8	
0046d60 fc66 alff f65c 0080 a083 011c 0000 8b00	
0046d70 6a07 8b01 ffcf 8b10 2474 8a14 8806 2444	
0046d80 8d10 2444 5010 c766 2444 3a15 e800 0294	325
0046d90 fffe 8559 74c0 8305 03f8 0d75 5868 80f5	
0046da0 8b00 e8c6 ca59 fffd e859 8295 fffe 358b	
0046db065c 0080 74ff 1424 c681 009c_0000 44e8 =	
0046dd0 8015 8500 9088 0000 8500 0fc9 e184 0000	
⁰⁰⁴ A colocition of molecular colocition to choice of	~
A selection of malware objuscation techniques	5
0046e00 0000 d88b 16e8 fdd6 8dff 2474 e81c d268	
0046e10 fffd 5cal 80f6 ff00 9cb0 0000 5700 7ce8	
0046e20 ff0f 59ff e859 8d15 fffe c085 840f 0090	
0046e30 0000 f633 7439 1824 0f75 448b 1424 5868	
0046e40 80f5 e800 c9b9 fffd a159 f65c 0080 3883	
0046e50 7501 3318 56db e856 2cca fffe 5959 f883	
0046e60 7401 4358 fb83 7c05 ebec 8d50 2444 5018	
0046e70 7cb8 7cf5 e800 796c ffff ff59 e830 d99a	
0046e80 fffd 8d59 2474 8b18 e8d8 dlec fffd db85	
0046e90 2275 448d 1824 e850 e3b4 fffd ff59 e830	
0046ea0 d978 fffd 8d59 2474 8b18 e8d8 d1ca fffd	
0046eb0 db85 0e74 e853 2dfa fffe 5759 71e8 fc33	
0046ec0 59ff 00 100 EISNE 748d 1424 a7e8	
004 Ged0 f dd1 5 f Contor Toobal ol Staff Mambar	
0046ef0 fffc ebf BM Basaarch - Haifa2	
0046f10 4485 082 December 2016 $024 0975$	
0046f20 448b 0424 eli7 10c2 5300 eli7 d88b 448b	
0046f30 0824 64f7 1424 d803 448b 0824 e1f7 d303	
0046f40 c25b 0010 cccc cccc cccc cccc cccc	
0046f50 5657 3353 8bff 2444 0b14 7dc0 4714 548b	
0046f60 1024 d8f7 daf7 d883 8900 2444 8914 2454	
0046f70 8b10 2444 0b1c 7dc0 4714 548b 1824 d8f7	
0046f80 daf7 d883 8900 2444 891c 2454 0b18 75c0	
0046f90 8b18 244c 8b18 2444 3314 f7d2 8bf1 8bd8 ©20	016 IBM Corporation
0046fa0 2444 f710 8bf1 ebd3 8b41 8bd8 244c 8b18	



Obfuscate

ob·fus·cate

/ˈäbfəˌskāt/ 🐠

render obscure, unclear, or unintelligible

render obscure, unclear, or unintelligible.

"the spelling changes will deform some familiar words and obfuscate their etymological origins" synonyms: obscure, confuse, make unclear, blur, muddle, complicate, overcomplicate, muddy, cloud,

befog

"mere rationalizations to obfuscate rather than clarify the real issue"

Why to be wilder (someone).

"it is more likely to obfuscate people than enlighten them"

synonyms: bewilder, mystify, puzzle, perplex, confuse, baffle, confound, bemuse, befuddle,

nonplus; informal flummox

"her work became more and more obfuscated by mathematics and jargon"

\rightarrow To make code hard to reverse engineer

LATIN	LATE LATIN	LATE LATIN	
fuscus			>>> obfuscate
dark		darkened	late Middle English

late Middle English: from late Latin obfuscat- 'darkened,' from the verb obfuscare, based on Latin fuscus 'dark.'



Stuff that malware does

- Taking advantage of vulnerabilities to infect
- Unpacking
- Anti-debugging, anti-research
- Obfuscation
- Malicious payload



About the examples

- Always, we have machine code
- However, to clarify, I've lifted some of them to c
 And simplified considerably



Example program

```
if (beingAnalyzed())
    fatal_error("this program is being analyzed");
else
    malicious();
```



Shift right

x = beingAnalyzed(); wasteTime(); /* but make it look like work */ if (x != 0) fatal_error("this program is being analyzed"); else malicious();



"Look Ma, no hands!"

hdc = GetDC(makeAnInvisibleWindow()); SetBkColor(hdc,beingAnalyzed()); wasteTime(); /* but make it look like work */ if (SetBkColor(hdc, RGB(6,7,8)) != 0) fatal_error("this program is being analyzed"); else

malicious();





And why make it easy?

hdc = GetDC(makeAnInvisibleWindow()); SetBkColor(hdc,beingAnalyzed()); wasteTime(); /* but make it look like work */ if (SetBkColor(hdc, RGB(6,7,8)) != 0) while(1) {wasteTime();}

else

malicious();





Infinite loops

Thread 1:

a: inc eax jmp a b: more code... c: inc eax jmp d d: more code... e: inc eax jmp e f: more code... Thread **17**:

Overwrite single byte 'e' with 'd'

etc. etc.





A way to waste time (but looks like work)

for (i=0; i<10000000; i++) {y = copyit(x); z = copyit(y); x = copyit(z);} int copyit(int i) { static int j=0, k=0; if (j>0) {j = i - 45; k = j + 10; } else if (k%7) {j = i - 100; k = j + 65;} **else** {j = i * 3; k = (j/3) - 35;} **return (k + 35);**



Another way to waste time

- Spawn lots of threads
- Pass events back and forth





Camouflaging API calls

Kernel32.GetModuleHandleA						
7dd71245 mov	edi, edi					
7dd71247 pus	h ebp					
7dd71248 mov	ebp, esp					
7dd7124a pop	ebp					
7dd7124b jmp	<pre>short GetModuleHandleA_0</pre>					

 So, call 7dd7124b directly (anonymous function)





Flouting coding conventions

pus ret	sh eax		p ji	oush eax mp <somesyste< th=""><th>mcal</th><th> ></th></somesyste<>	mcal	>
push eax (=a) push ebx (=b) push ecx (=c) ret		4				
		ecx (=c)		je a jmp b	call p	
C:	ret		b:	jge a		
b:	ret			jmp c 	p:	pop eax
a:			C:	jmp a		



Not in my backyard

CreateProcess ("c:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe", NULL,NULL,FALSE,0,NULL,NULL,&sinfo,&pinfo);

BYTE *startaddress = (BYTE *)

VirtualAllocEx

(pinfo.hProcess, 0, size, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

WriteProcessMemory

(pinfo.hProcess, startaddress, &localbuf, size, &byteswritten);

CreateRemoteThread

(pinfo.hProcess, 0, 0, (LPTHREAD_START_ROUTINE)startaddress.



Where's Wally?

ret = NtQuerySystemInformation(x, y, z, w);

typedef int (WINAPI*ftype)(PVOID a, PVOID b, PVOID c, PVOID d);

HINSTANCE lib = LoadLibrary(TEXT("ntdll.dll"));

ftype fp = (ftype) GetProcAddress(lib, "NtQuerySystemInformation");

ret = (fp) (x, y, z, w);

		_		_	
_	_	_	_	-	
-	-	_	_	_	
-		_			
				_	
		-	-	_	
				-	

Does anybody see the malicious code?

```
try {
 string s1 = "1.052033", s2 = "e+00", s3 = "3";
 for (int i = 0; i<1000; i++) {
  for (int j = 0; j<1000;j++) {
    long double f = stof(s1+s2+s3); s1 += s3;
  }
  int t = stoi(s3); t *= 2; s3 = to_string(t);
catch (...) { f(); }
```



Wasting time and camouflaging ops





Time stamp confusion

0xb8ddf1	push eax	

0046d50	ffld	50ff	e857	8ce3	ffff	5c8d	2024	70e8	
0046d60	fc66	alff	f65c	0080	a083	011c	0000	0048	
0046d70	6a07	8b01	ffcf	8b10	2474	8a14	8806	2444	
0046d80	8d10	2444	5010	c766	2444	3a15	e800	0294	
0046d90	fffe	8559	74c0	8305	03f8	0d75	5868	80f5	
0046da0	8b00	e8c6	ca 59	fffd	e859	8295	fffe	358b	
0046db0	f65c	0080	74ff	1424	c681	009c	0000	44e8	
0046dc0	fe45	8bff	59f8	ff85	840f	00f4	0000	5cal	
0046dd0	80f6	8b00	9c88	0000	8500	Ofc9	e184	0000	
0046de0	5100	b889	0098	0000	448d	2024	5057	cfe8	
0046df0	fe58	8bff	5c35	80f6	8300	Occ4	c681	0 0 9 4	
0046e00	0000	d88b	16e8	fdd6	8dff	2474	e81c	d268	
0046e10	fffd	5cal	80f6	ff00	9cb0	0000	5700	7ce8	
0046e20	ffOf	59ff	e859	8d15	fffe	c085	840f	0 0 9 0	
0046e30	0000	f633	7439	1824	0f75	448b	1424	5868	
0046e40	80f5	e800	c9b9	fffd	a159	f65c	0080	3883	
0046e50	7501	33		856	2cc	fe	5959	f883	
0046e60	7401	4358	33	7c05	eber	5 0	1	5018	
0046e70	7cb8	7cf5	0.0	79 f.c	ffi	- 59	630	89a	
0046e80	fffd	8d59	74	8h P	-8d8	dled	ffd	085	
0046e90	2275	448d	24	e8 0	3b4	fffd	f59	83	
0046ea0	d978	fffd	. a 59	24 4	o1 8	e8d8	ulca	ff 📕 🦳 🗋	
0046eb0	db85	0e74	e853	2dīa	iffe	5759	71e8	fc3	
0046ec0	59ff	006a	32e8	fell	59ff	748d	1424	a7e <mark>5</mark> –	
0046ed0	fddl	5fff	5b5e	e58b	c35d	8b55	8bec	0c45	
0046ee0	e883	7400	4815	0d74	7448	480f	0c74	b6e8	
0046ef0	fffc	ebff	e805	f797	ffff	c033	5d40	0cc2	
0046f00	cc00	25ff	900c	007c	CCCC	CCCC	CCCC	adaa	
0046f10	448b	0824	4c8b	1024	c80b	4c8b	0c24	0975	
0046f20	448b	0424	elf7	10c2	5300	elf7	d88b	448b	
0046f30	0824	64f7	1424	d803	448b	0824	elf7	d 3 0 3	
0046f40	c25b	0010	CCCC	CCCC	CCCC	CCCC	CCCC	adaa	
0046f50	5657	3353	8bff	2444	0b14	7dc0	4714	548b	
0046f60	1024	d8f7	daf7	d883	8900	2444	8914	2454	
0046f70	8b10	2444	0b1c	7dc0	4714	548b	1824	d8f7	
0046f80	daf7	d883	8900	2444	891c	2454	0b18	75c0	
0046f90	8b18	244c	8b18	2444	3314	f7d2	8bf1	8bd8	
0046fa0	2444	f710	8bf1	ebd3	8b41	8bd8	244c		

IBM



Backup slides



Packing





Packing/unpacking

Pack

-To compress an executable file

Unpack

-To run a second executable that:

- Decompresses the compressed file
- Runs it



Packing is not always malicious

- Reduce needed storage or download time
- Protect intellectual property through obfuscation
 - -Because packed code is harder to reverse engineer
 - -Especially if combined with encryption



But often it is malicious

Protect intellectual property through obfused intellectual

obfuscation

- -Because packed code is harder to reverse engineer
- -Especially if combined with encryption

Render signatures useless

-Because once my malware is identified, all I have to do is repack



And not always easy to understand



Example (based on Shylock/Caphaw)

- Take a piece of malware (call it A), encrypt it (giving B)
- Then, write a program (C) that generates the encrypted code
- Encrypt C, write it into your data section
- Write a program (**D**) that:
 - -Allocates space in memory
 - Decrypts its data section there (this will give you **C**)
 - -Allocates more space in memory, in which it writes code (E) that:
 - Erases (D)
 - Runs C to generate B (overwriting the zeroed-out D)
 - Allocates more memory
 - Decrypts **B** into the newly alloced memory (this will give you **A**)
 - Runs the result (A the original malicious code)
 - Runs E



Anti-debugging





Anti-debugging, anti-research

- Straightforward packed code is fairly easy to unpack

 Just watch what it does, e.g., in a debugger or in an emulator
 Watch carefully! (i.e., in a VM)
- Anti-debugging techniques are intended to make unpacking more difficult
 - -Exploit subtle differences between environments
 - Under debugger **vs.** independent run
 - In virtual vs. real machine
 - -Or just make it **confusing**, e.g.
 - Lots of jumps
 - Lots of threads



Exceptions

- Register an exception handler, then generate an exception on purpose
 - -Debuggers are given first chance to handle exception
 - So behavior is different under debugger vs. without debugger (there are workarounds – the point is just to make it hard for the reverser)
- Exception handler has access to all registers, including
 EIP
 - -So has full control over where it returns and in what state
 - -Lets you do funky stuff

Funky things you can do with exceptions

- Suppose EIP points here: XX YY ZZ TT UU VV WW
- And suppose operation beginning with XX consumes 3 bytes total, like this:



- Furthermore, assume operation has been written to generate an **exception**
- In the exception handler, **increment** EIP, so now the operation is this:



Or you can just overwrite the opcode





Funky things, continued

- Change the value of any general purpose register
- Or any flag
- Makes it hard to read the code, hard to debug, and hard for static disassemblers
- You can of course **nest** exceptions...



Checking for debugger, VM

- IsDebuggerPresent(), or access bit accessed by IsDebuggerPresent() (bit 3 of structure pointed to by FS:[30h] (Thread Information Block))
- NtQueryInformationProcess(), with certain parameters reveals debug port
- Check for presence of various files, windows, registry keys, e.g.
 - File "\\.\SICE": indicates SoftICE kernel debugger
 - Window of class "ollydbg" indicates OllyDbg binary debugger
 - Registry key named "Software\Wine" indicates Wine (un)emulator
- Check for environment variables, e.g.
 Presence of WLNumDLLsProt indicates to malware it is being watched
- Opcode 0f 00 (SLDT) returns 0 on Windows, non-zero on VMWARE
- Calculate md5 of code, compare to expected (finds software breakpoints)
- There are MANY other ways



Timing

- There are various ways to check the time
- Can be used to distinguish run under debugger or VM from independent run
- Phantom checks the time frequently, but then discards result
 - -Seems to be a way to **confuse** the human, who might waste time faking time