Pierre-Marc Bureau – bureau@eset.sk Joan Calvet - j04n.calvet@gmail.com

UNDERSTANDING SWIZZOR'S OBFUSCATION





Swizzor

- Present since 2002 !
- AV companies receive hundreds of new binaries daily.
- Nice icons :



Little publicly available information.

Presentation Outline

Introduction

The packer

The heart of Swizzor

Conspiracy theories

Welcome in Swizzorland !

At first sight :

- Standard Win32 binary
- Clean compiler signature with a nice "WinMain()"
- Long list of imports
- Statically linked with the C standard library (msvcrt)

Sounds cool! But if you try to disassemble it and dig deeper, you could see...

00403258	push	edi	
00403259	mov	edx. [esp+4+arg 0] 👗	
0040325D	sub	edx. 13Eh	
00403263	lea	edi. unk 433564	
00403269	mov	eax. off 420554[edx]	
0040326F	mnu	ecx. 28B3b	
00403274	add	ecx. eax	X
00403276	sub	ecx. [edi]	N
00403278	mov	eax. ecx	No.
0040327A	DOD	edi	
0040327B	retn	4	
<u>A</u> .		Å	
T		7 1	
201			
200			
00408F	ZE MOV	dword_433660, eax 🗨	17 - 12 - 12 - 12 - 12 - 12 - 12 - 12 -
00408F	83 push	19Ah	Same and
00408F	88 call	sub_403258	1
00408F	8D mov	dword_433668, eax	
00408F	92 push	14Ah	1
00408F	97 call	sub_403258	
00408F	9C jmp	1oc_4090F6	<i>\</i>
		anti-	NUTRICALCALICA
1			etantertantertante Militature
00402BF:	3 push	edi	anconcone,
00402BF3 00402BF4	3 push 4 mov	edi edx, [esp+4+arg_0]	ann an
00402BF 00402BF 00402BF	3 push 4 mov 3 sub	edi edx, [esp+4+arg_0] edx, 1B4h	
00402BF3 00402BF4 00402BF4 00402BF4	3 push 4 mov 3 sub 1 ea	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534	anconcourse
00402BF3 00402BF4 00402BF4 00402BF6 00402BF6 00402C04	3 push 4 mov 3 sub 5 lea 4 mov	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_4359D8[edx]	anconcone,
00402BF3 00402BF4 00402BF4 00402BF6 00402C04 00402C04	3 push 4 mov 3 sub 5 lea 4 mov 4 mov	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_4359D8[edx] edi, 19B7DBh	and the second s
00402BF3 00402BF4 00402BF4 00402BF6 00402C04 00402C04 00402C04	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 5 sub	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_4359D8[edx] edi, 19B7DBh edi, eax	- Aller
004028F 004028F 004028F 004028F 00402C9 00402C9 00402C0 00402C01	3 push 4 mov 3 sub 5 lea 4 mov 4 mov 7 sub 1 add	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 19870Bh edi, eax edi, [ecx]	
004028F; 004028F; 004028F; 004028F; 00402264; 00402204; 00402209; 00402201; 0040221;	3 push 4 mov 3 sub 5 lea 4 mov 4 mov 7 sub 1 add 3 mov	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 1987DBh edi, eax edi, [ecx] eax, edi	1
004028F3 004028F3 004028F3 004028F3 00402C97 00402C97 00402C97 00402C13 00402C13	3 push 4 mov 3 sub 5 lea 4 mov 4 mov 7 sub 5 sub 1 add 3 mov 5 pop	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_4359D8[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi	All and a second
0040285 0040285 0040285 0040285 00402261 0040220 0040220 0040220 0040221 0040221 0040221	3 push 4 mov 3 sub 5 lea 4 mov 4 mov 5 sub 1 add 3 mov 5 pop 5 retn	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_4359D8[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi 4	
0040285; 00402854 00402856 00402856 00402269 00402209 00402209 00402201 00402211 00402211 00402211	3 push 4 mov 3 sub 5 lea 4 mov 4 mov 7 sub 1 add 3 mov 5 pop 5 retn	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi 4	
0040285 0040285 0040285 0040285 00402269 00402209 0040220 0040221 0040221 0040221	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 4 mov 5 sub 1 add 3 mov 5 pop 5 retn E push	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 19870Bh edi, eax edi, [ecx] eax, edi edi 4	
004028F3 004028F4 004028F4 004028F4 00402C94 00402C91 00402C13 00402C13 00402C13 00402C14 00402C14 00402C14 00402C14 00402C14	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 5 sub 1 add 3 mov 5 pop 5 retn E push 6 nush	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 1987DBh edi, eax edi, [ecx] eax, edi edi 4 edi	K
00402853 00402854 00402856 00402856 00402004 00402004 00402004 00402013 00402013 00402013 00402013 00402013 00402013 0040208 0040208	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 5 sub 1 add 3 mov 5 pop 5 retn E push 6 mou	edi edx, [esp+4+arg_8] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 1987DBh edi, eax edi, [ecx] eax, edi edi 4 edi edi ebx eby [esp+8+arg_8]	K
0040285 0040285 0040285 0040285 00402265 00402209 00402209 0040221 0040221 0040221 0040221 0040221 0040221 0040228 0040228 0040228 0040228	3 push 4 mov 3 sub 5 lea 4 mov 7 sub 1 add 3 mov 5 pop 5 retn E push F push 0 mov 9 sub	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi edi edi edi ebx ebx, [esp+8+arg_0] ebx, 30b	~
0040285 00402854 00402856 00402856 0040200 0040200 0040200 0040201 0040201 0040201 0040201 0040201 0040201 0040201 0040208 0040208 0040209 0040209	3 push 4 mov 3 sub 5 lea 4 mov 7 sub 1 add 3 mov 5 pop 5 retn E push F push 9 mov 4 sub 7 mov	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi edi 4 edi ebx ebx, [esp+8+arg_0] ebx, 30h eax dword_b2256[ebx]	6
0040285; 00402854 00402854 0040269 0040209 0040209 0040209 0040201 0040201 0040201 0040201 0040208 0040208 0040208 0040209 0040209	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 7 sub 1 add 3 mov 5 pop 5 retn 6 push 6 mov 4 sub 7 mov 9 loa	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 19870Bh edi, eax edi, [ecx] eax, edi edi edi edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi unk h2256C	
0040285 0040285 0040285 0040268 0040209 0040209 0040201 0040201 0040201 0040201 0040201 0040201 0040208 0040208 0040209 0040209 0040209	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 5 sub 1 add 3 mov 5 pop 5 retn E push 6 mov 4 sub 7 mov 0 lea 2 mov	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 1987DBh edi, eax edi, [ecx] eax, edi edi 4 edi 4 ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ocx 2116Eb	
00402853 00402854 00402854 00402854 0040204 0040204 0040204 00402013 00402013 00402013 00402013 00402013 00402013 004020208 0040208 0040209 0040209 0040209 0040209	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 5 sub 1 add 3 mov 5 pop 5 retn E push 6 mov 4 sub 7 mov 0 lea 3 mov 9 sub	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 1987DBh edi, eax edi, [ecx] eax, edi edi 4 edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ecx, 2416B5h ecx, 242	
0040285 0040285 0040285 0040285 00402269 00402209 00402209 0040221 0040221 0040221 0040221 0040221 0040221 00402209 0040209 0040209 0040209 0040209 0040209	3 push 4 mov 3 sub 5 lea 4 mov 7 mov 5 sub 1 add 3 mov 5 pop 5 retn 6 push 6 mov 4 sub 7 mov 0 lea 3 mov 8 sub 8 sub	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi edi edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ecx, 2416B5h ecx, eax ecx, fordi]	
0040285; 00402854 00402856 00402261 00402201 00402201 0040221; 0040221; 0040221; 0040221; 0040221; 00402203 00402203 00402209 0040209 0040209 0040209 0040209 00402004	3 push 4 mov 3 sub 5 lea 4 mov 7 mov 7 sub 1 add 3 mov 5 pop 5 retn 8 mov 4 sub 7 mov 9 lea 3 mov 8 sub 8 sub A add	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_4359D8[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ecx, 2416B5h ecx, eax ecx, [edi]	
0040285; 00402854 00402854 00402694 00402094 00402094 00402094 0040201; 0040201; 0040201; 0040201; 0040201; 004020208 0040209 0040209 0040209 0040209 0040209 0040204 004020A 004020A	3 push 4 mov 3 sub 5 lea 4 mov 4 mov 5 sub 1 add 3 mov 5 retn 6 retn 6 mov 4 sub 7 mov 0 mov 4 sub 7 mov 0 a mov 8 sub A add C mov 7 poc	edi edx, [esp+4+arg_0] edx, 184h ecx, unk_433534 eax, dword_435908[edx] edi, 1987DBh edi, eax edi, [ecx] eax, edi edi edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ecx, 2416B5h ecx, eax ecx, [edi] eax, ecx ebx	
0040287 0040287 0040287 00402687 0040209 0040209 0040209 0040201 0040201 0040201 0040201 0040208 0040208 0040209 0040209 0040209 0040209 0040209 0040209 0040204 004020A 004020A	3 push 4 mov 3 sub 2 lea 4 mov 4 mov 4 mov 5 sub 1 add 3 mov 5 retn 6 retn 7 mov 0 lea 3 mov 4 sub 7 mov 0 lea 3 mov 8 sub A add C mov E pop	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi 4 edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ecx, 2416B5h ecx, eax ecx, [edi] eax, ecx ebx edi	
00402853 00402854 00402856 00402269 00402209 00402209 00402201 00402213 00402213 00402213 00402213 00402208 00402208 00402208 00402209 00402209 00402204 00402204 00402204 00402204	3 push 4 mov 3 sub 5 lea 4 mov 7 sub 1 add 3 mov 5 pop 5 retn 6 mov 5 pop 5 retn 7 mov 0 lea 3 mov 0 lea 3 mov 0 lea 3 mov 0 lea 3 mov 0 lea 3 mov 0 lea 5 cmov 0 cmov E pop F pop	edi edx, [esp+4+arg_0] edx, 1B4h ecx, unk_433534 eax, dword_435908[edx] edi, 19B7DBh edi, eax edi, [ecx] eax, edi edi edi edi edi ebx ebx, [esp+8+arg_0] ebx, 30h eax, dword_4327EC[ebx] edi, unk_43356C ecx, 2416B5h ecx, eax ecx, [edi] eax, ecx ebx edi	

	~	00402DB3 push 00402DB4 push	esi ebx	
	1	00402DB5 mov	ebx, [esp+8+arg_0]	
00408365 mov [esp+18h+arg_58], eax		00402DB9 sub	ebx, OFDh	
00408309 push 119h	/	004020BF MOV 004020C5 lea	edx, uworu_437298[eDx] edx_dword_433530	
00400302 Call Sub_402003	r	00402DCB mov	esi. 2FBDh	
00408378 push 1		00402DD0 xor	esi, eax	
0040837A call sub_40803A		00402DD2 add	esi, [edx+18h]	
0040837F mov [esp+18h+arg_4C], eax		00402DD5 sub	esi, [edx+44h]	
00408383 mov edi, dword_433640		00402DD8 mov	eax, esi	
00408389 sub esp, 8			ebx	
00400386 push eur		00402006 pop 0040200C reto	езі h	
88488392 mou dword 433654, eax		88482000 TCCI	-	
00408397 push 125h				
0040839C call sub_402DB3				
004083A1 jmp loc_408D0B	128	0040390E push	edi	
	and and an	0040390F push	ebp	
		00403910 mov	ebp, esp	
EN UL		00403912 MOV 006002017 cub	edi, 55EV427HN	
00408D0B ; START OF FUNCTION CHUNK FOR sub_408CB4		00403917 SUD 00403016 lea	edi [edi+638A6b]	
00408D 0B		00403920 add	edi. (offset loc 403005+3	3)
19949809B 10C_49809B:		00403926 mov	eax, edi	1
00408000 muv [esp+20n+ary_36], eax		00403928 pop	ebp	
88488017 call sub 48398F		00403929 pop	edi	
00408D1C push 20h]	0040392A retn	4	
00408D1E call sub_408F7E				
00408D23 mov dword ptr [ebp-380h], 1D0h				
00408D2D sub esp, 0Ch	name and		1	e.
00408D30 call sub_402BF3			esi edu fospelueuro el	L
00408035 MUV UWUru_433004, eax	1	00403830 MOV	edx, [esp+4+ary_0] edx, 2Ab	L
8848803F call sub 48383C		00403844 lea	ecx. unk 433594	L
00408D44 mov [esp+8+arg 4], eax		0040384A mov	eax, dword 435F38[edx]	L
00408D48 push dword_433670	and the second s	00403850 mov	esi, 134E3Ch	L
00408D4E call sub_403258		00403855 sub	esi, eax	L
00408D53 mov dword_433648, eax		00403857 add	esi, [ecx]	L
		00403859 duu 00103859 duu	esi, Offset 100_403E90	L
		00403851 MOV	eda, esi	L
00408D66 push 3Ch		00403862 retn	4	L
00408D68 call sub 402D8E		-		
00408D6D mov dword_433658, eax				
00408D72 push 96h	ALL CONTRACTOR OF CARD			
00408D77 call sub_40383C	- gen	00402DFF push	esi	
00408076 MUV [250+8+ary_58], eax		00402E00 push	ebp	
00408085 call sub 4020FF		00402E01 MOV	eop, esp 	
for any any and any any and and any and and any and any and any and any and any and any any any any any any any		00402E03 MOV	ecx, [ebn+arg A]	
		00402E0B 1ea	esi, dword 433530	
		00402E11 mov	edx, [esi+2Ch]	
▼		00402E14 xor	ecx, edx	
		00402E16 mov	eax, [esi+54h]	
		00402E19 xor	ecx, eax	
		00402E18 SUD	etx, [ESI+OUN] acy offset los h09000	
		00402E24 mou	Pax. PCX	
		00402E26 pop	ebp	
		00402E27 pop	esi	
		00402E28 retn	4	Ø





This is the packer !

- Between 40 M and 100 M CPU instructions.
- Objective : protect the original code which is the heart of Swizzor against:
 - Manual reverse-engineering
 - Detection by security products

Problem

- We want to understand what's is going on inside :
 The packer
 - The heart of Swizzor (original executable)
- But:
 - It seems difficult (cf. previous slides)
 - We are newbies

First step : the packer

Context:

- Mono-thread, 32 bits binary.
- Less than 1% of API calls :
 - → Not enough to understand API calls, need to think at assembly level.
- Only one layer of code : no dynamic code before the unpacked binary.
- The packer layer for one binary will have the same behavior over multiple executions :
 - → The addresses are the same inside the main module (in particular the ones used to access the data section)

Proposed solution (1)

- Set of tools:
 - A tracing engine which is going to collect « information » for us
 - Some tools to exploit the collected information:
 - Visualization to quickly identify interesting patterns or recognize already seen behaviors.
 - Heuristic engine based on previous knowledge.

Proposed solution (2)

- Work process:
 - <u>Tracing step</u>: once per binary, it outputs two files:
 - Improved trace : detailed view.
 - Events file : high level view.
 - <u>Analysis step</u>: standard RE work but directed by the previously collected information.

Tracing engine

- Pin : dynamic binary instrumentation framework:
 - Insert arbitrary code (C /C++) in the executable (JIT compiler).
 - Rich library to manipulate assembly instructions, basic blocks, library functions...
 - Deals with self-modifying code.
- Check it at <u>http://www.pintool.org/</u>

But what information do we want to gather at run-time ?

1. Memory Access

 Swizzor binaries have a data section of more than 10KB and weird stuff inside.

 It would be interesting to see the actual access made by the code in this section.

- Easy to do with PIN, cf. documentation.
- BTW, most of these access are hard to decide statically.

2. API calls (1)

- PIN provides an API to deal with system calls, but we are more interested in the APIs functions that actually perform system calls...
- Detection of API calls:
 - Dynamic linked library : PIN functions like RTN_FindNameByAddress()
 - Statically linked library: use IDA Flirt.

API calls (2)

- Detecting is cool, but we can do better : dump arguments and return values!
 - Function prototypes given in entry of the PIN tool:

HMODULE GetModuleHandleA(IN LPCSTR); BOOL GetThreadContext(IN HANDLE,IN_OUT LPCONTEXT); WCHAR_T* wcschr(IN WCHAR_T*,IN WCHAR_T);

- Instructions for dumping:
 - Basic types:



Complex types:

SECURITY_ATTRIBUTES LPSECURITY_ATTRIBUTES D[DWORD,LPVOID,BOOL]
I[SECURITY_ATTRIBUTES]

3. Loops

- Why is it interesting ?
 - Most of the time, a loop does one thing: decrypting data, resolving imports, containing other loops...
 - In a « divide and conquer » approach, a loop can thus be considered as an independent sub-problem.

Loops in Swizzor!



More than 95% of the packer code is in loops !

Loops: How to detect them ? (1)

(SIMPLIFIED) STATIC POINT OF VIEW

PIN TOOL POINT OF VIEW



When tracing a binary, can we define a loop as the repetition of an instruction ?

Loops: How to detect them ? (2)

(SIMPLIFIED) STATIC POINT OF VIEW

PINTOOL POINT OF VIEW



This is not a loop ! So what's a loop ?

Loops: How to detect them ? (3)

(SIMPLIFIED) STATIC POINT OF VIEW

PIN TOOL POINT OF VIEW



EXECUTED	TIME
INSTRUCTION1	1
INSTRUCTION ₂	2
INSTRUCTION3	3
INSTRUCTION1	4
INSTRUCTION ₂	5
INSTRUCTION3	6
INSTRUCTION1	7

What actually define the loop, is the back edge between instructions 3 and 1.

Loops: How to detect them ? (4)

- In our dynamic world a back edge is an instruction pair (Leader, Tail) where:
 - The Leader has been first executed.
 - The Tail is executed just before the Leader <u>at least</u> two times.
- Thus we detect on the fly the (Leader, Tail) pair, i.e. the loops.
- Detecting loops is cool but we can do better : collect the addresses that have been read and written by the loop !

4. Exceptions

 Between 5 and 10 exceptions in a standard Swizzor packer.

- Detect them by instrumentation of KiUserExceptionDispatcher()
- Dump the error code of the exception with the fault address.

5. Dynamic code

 If code is executed outside of either the main module or shared libraries, we detect it as dynamic code (remember : no dynamic code inside the main module for Swizzor!)

 Identify the instruction which transfers control to new code.

6. Swizzor "calculus"

- A "calculus" is a small block of code which makes calculations on its argument and returns the result (no memory modification, no API, etc).
- We detect them with a simple heuristic in our PIN tool :
 - Between 7 and 20 instructions.
 - More than 40% of arithmetic instructions (XOR/ADD/SUB).
 - Ends with a RETURN instruction.
- We store where the result is written.

Output 1: improved trace

[6][00404117] mov dword ptr [ebp-0x40], eax W 0x0012FBF0 [7][0040411A] callAPI OpenMutexW A1: [DWORD] 0x001F0001 A2: [BOOL] 0x0000001 A3: [LPCWSTR] "XJLFOQ" RV: [HANDLE] 0x0000000 [59][004041D2] callM calcul1 [60][004041D7] mov ecx, eax [93][0040310F] callAPI __snwprintf A2: [SIZE_T] 0x00000190 A3: [WCHAR_T*] "%4u ange %04x (%x" RV: [INT] 0x0000018 A1: [WCHAR T*] "1216 ange f92c6aeb (16c" [94][00403114] add esp, 0x18 [95][00403117] push dword ptr [ebp-0x28] R 0x0012FC08 [1490][0040C136] mov dword ptr [edi], 0x6 W 0x000003E8 **!! EXCEPTION !!** • • •

(Easy to look for regular expressions inside the trace!)

Output 2: events file

[=> EVENT: CALCULUS <=][TIME: 294][@: 0x00402E3A]</pre> M: calcul4 W: 0x0012FB8C [=> EVENT: API CALL <=][TIME: 299][@: 0x00402FC2]</pre> F: malloc A1: [SIZE_T] 0x00002A84 RV: [VOID*] 0x023A6E38 [=> EVENT: LOOP <=][START:634 - END:1381][LEAD@:0x0040F62A - TAIL@:0x0040F41C]</pre> **TURN: 57 READ ZONES:** [0x0042A8A5-0x0042A8EC: 72 B] [0x0042A579-0x0042A5F4: 124 B] [0x00426234-0x0042623F: 12 B] WRITE ZONES: [0x0042A8A5-0x0042A8EC: 72 B] [0x0042A579-0x0042A5F4: 124 B] [0x00428440-0x00428447: 8 B] [=> EVENT: EXCEPTION <=][TIME: 1490][@: 0x0040C136]</pre> **EXCEPTION CODE:** 0xc0000005 (STATUS_ACCESS_VIOLATION)

Output 2: timeline!

Between 400 and 600 events in a standard
 Swizzor packer.

- Not easy to read in a plain text file.
- Build a "timeline" by using the Timeline widget from the MIT :

http://www.simile-widgets.org/timeline/

	:								1		
		o cal	lcul1 🍙srand()	ocalcul2 ocal	lcul5 🖕 calcul1 🍚	calcul	l3 🖕 calcul6	ocalcul3 ocalcu	ll _O calcu	13 Loop2	
		otime()) 💿 calcul2 🍙	calcul3 🔾 calcul2	ocalcul1 ocalcul	.7 🥥	calcul4 🖕 c	alcul4 ocalcul9	🕘 😋 calcul	L2 Loop1	
	o Oper	nMutexW() 🥥	calcul1 o_snwpri	.ntf() ocalcul2 (calcul7 calcul7	′ <u></u> ⊂ ⊂	alcul9 🖕 ca	lcul9 ocalcul3	○ calcul2	🖕 calcul9	
		😑 calcul1	ocalcul2 oc	alcul1 🕥 Ca	alcul7 🖕 calcul9	o ca	lcul4 _o cal	cul9 🖕 calcul3	calcul6	🖕 calcul8	
		🔾 W	/cschr() osnwp	orintf() 👌 calcu	15 🖕 calcul9	🔵 cal	cull _O calcu	ul9 👩 calcul9 🥥	calcul3	🖕 calcul5	
		⊖wcs]	len()	🔵 calcul	1 🥥 calcul9 🕥 ca	alcu15	🔵 calcu	l1 👌 calcul3 👌 c	alcul9	ocalcul7	
		🔵 GetCo	ommandLineW()	ocalcul5	ocalcul2 ocal	cu15	🔵 calcul2	2 👌 calcul2	lcul9	🖕 calcul4	
		🔵 calcul2	2 oGetCurrentT	hreadId()	ocalcul5		ocalcul1	ocalcul9 omal	loc()	⊖malloc()	
		⊖GetModul	eFileNameW()	0	calcul8		🔾 calcul8	○ CloseHandle	()	_O calcul3	
	G	GetCurrent	ProcessId()		0		0	malloc()		_O calcul1	
$\left \right $	q		••			8				omalloc()	
	👝 Ge	_snwprint	:f()							_O calcul7	
		API CALL [@	0x004045DB][Tim	e: 138]						_O calcul6	
			71 0 00000000							_O calcul6	
		A2: [SIZE Δ3· [WCHΔ	_] 0x00000080 R T*l "%i coren '	%d olagm %i"						🖕 calcul6	
		RV: [INT]	0x00000026	ou olugii /ol						🖕 calcul4	
		A1: [WCHA	R_T*] "1216 core	n -114529557 olag	gm -253932429"					🖕 calcul4	
										ocalcul3	
										ocalcul1	
										ocalcul6	
	0.00									🖕 calcul1	
_	Øhr		1	2	3	4		5	6	△ calcul1	
								<u></u>	<u> </u>		L
\exists					START						
						1970					

SMALL

TIME

UNIT OF

BIG UNIT

OFTIME

TIME



Enough with the tools, what about the packer?

Era 0: FUD



Era 1: Prepare the packer

Example of simple loop

⊚mall	.oc()	Loop2	2		Loc
Loop2					8
[Lead@ [Start): 0x0040 :: 634][E	F62A][Que nd: 1381]	ue@: 0x0	0040F41C]	
TURN READ	1: 57 20NES:	0x0042A8	A5-0x004	12A8EC: 7	2 B
0x004 0x004	26234-0x 28440-0x	0042623F: 00428447:	12 B 12 B 8 B		
0x004 0x004 [0x004	286C9-0x	004286D4: 0042876C:	12 B 124 B]	1	
WRIT	E ZONES:	0x0042A	8A5-0x00	42A8EC:	72 B
0x004 0x004 0x004	26234-0x 28440-0x	0042623F: 0042623F: 00428447:	124 B 12 B 8 B		

0x004330C0-0x004330C3: 4 B

omalloc()	Loop2	Loo
eHandle()	Loop1	Loop
Loop1		®
[Lead@: 0x0040F605 [Start: 624][End: 1][Queue@: 0x0040F671] 1338]	
TURN: 4 READ ZONES: [0x00 [0x00428739-0x0042] [0x00428755-0x0042] [0x0042876D-0x0042] [0x00426240-0x0042] [0x0012FB40-0x0012] [0x0012FB0C-0x0012] [0x004330C0-0x0043] [0x004330D4-0x0043] [0x004280F9-0x0042] [0x00428448-0x0042] [0x004286F9-0x0042] [0x004286F9-0x0043]	042A8ED-0x0042A8F0: 4 873C: 4 B] A5F8: 4 B] 6243: 4 B] FB43: 4 B] FB0F: 4 B] 30C3: 4 B] 30D7: 4 B] FB7F: 4 B] 844B: 4 B] 8700: 8 B] 2F23: 4 B]	B]
[0x004286D9-	0x004286E8: 16	В]
0X00429745-	0X00429754: 16	В]
WRITE ZONES: [0x/ [0x0042A5F5-0x0042/ [0x00426240-0x0042/ [0x0012FBE8-0x0012/ [0x004330D4-0x0043] [0x00428448-0x0042/ [0x004330C0-0x0043] [0x0012FB7C-0x0012/ [0x0012FB40-0x0012/ [0x0012FB0C-0x0012/	0042A8ED-0x0042A8F0: 4 A5F8: 4 B] 6243: 4 B] FBEB: 4 B] 30D7: 4 B] 844B: 4 B] 30C3: 4 B] FB7F: 4 B] FB43: 4 B] FB0F: 4 B]	В]

Era 1: Example of simple loop (2)

Memory profile : [#Read, #Write, #Call/Jmp]



Era 1: Example of simple loop (3)

malloc()	Loon	seHandle()	Loop1	Loop
	Loop2	Loop1		ම
Loop2		ی ۲ [Lead@: 0x004 [Start: 624][0F605][Queue@: 0x0040F6 End: 1338]	671]
[Lead@: 0x0040F6	52A][Queue@: 0x0040F41C	TURN: 4	[0x004249ED_0x004249E	a. 1 D1
[Start: 634][End	1: 1381]	[0x00428739-0	x0042873C: 4 B]	0.4Dj
		[0x0042A5F5-0 [0x0042876D-0	0x0042A5F8: 4 B] 0x00428770 4 B]	
		[0x00426240-0	0x00426243: 4 B]	
READ ZONES: 0	0X0042A8A5-0X0042A8EC:	[0x0012FB40-0	x0012FB43: 4 B]	
0x0042A579-0x00)42A5F4: 124 B	[0x0012FB0C-0	0x0012FB0F: 4 B]	
0x00426234-0x00	042623F: 12 B	[0x004330C0-0	0x004330D7: 4 B]	
0x00428440-0x00	0428447: 8 B	[0x0012FB7C-0	0x0012FB7F: 4 B]	
0x004330C0-0x00	4330C3 · 4 B	[0x00428448-0	x0042844B: 4 B]	
0x00439600_0x00	4286D4 12 P	[0x004286F9-0	0x00428700: 8 B]	
0x00428009-0x00	42076C 424 P	0x00432F20-0	0x00432F23: 4 B	46.01
[0X004286F1-0X00	042876C: 124 BJ	[0x004280	D9-0X004286E8:	16 B]
[0x00432F20-0x00	0432F23: 4 B	[0x004297	745-0x00429754:	16 B]
WRITE ZONES:	0x0042A8A5-0x0042A8FC:	2 B		
0x00424579_0x00	M205E4 · 124 B	WRITE ZONES	5: [0x0042A8ED-0x0042A8	F0: 4 B]
0x0042A373-0x00	A2622E, 12 P	[0x0042A3F3-0	0x00426243: 4 B]	
0x00426234-0x00	42623F: 12 B	[0x0012FBE8-0	x0012FBEB: 4 B]	
0x00428440-0x00	0428447: 8 B	[0x004330D4-0	x004330D7: 4 B]	
0x004286C9-0x00	4286D4: 12 B	[0x00428448-0	x0042844B: 4 B]	
0x004330C0-0x00	04330C3: 4 B	[0x004330C0-0	0x004330C3: 4 B]	
0		[0x0012FB/C-0	0X0012FB/F: 4 B]	
			00120043. 4 D]	

_ malloc()

Loop2

keyHeuristic... Done 0x426234-0x426244 [DECRYPTED BY KEY 0] on 16 bytes 0x428440-0x42844c [DECRYPTED BY KEY 0] on 12 bytes 0x4286c9-0x4286d5 [DECRYPTED BY KEY 0] on 12 bytes 0x4286d9-0x4286ed [CONTROL STRUCTURES FOR KEY 0x4286f1] 0x4286f1-0x428771 [KEY 0] 128 bytes 0x429745-0x429759 [CONTROL STRUCTURES FOR KEY 0x4286f1] 0x42a579-0x42a5f9 [DECRYPTED BY KEY 0] on 128 bytes 0x42a8a5-0x42a8f1 [DECRYPTED BY KEY 0] on 76 bytes

Era 1:

More original loops

Read clusters jump over 3 bytes !



Big write zone.

Loop28

[Lead@: 0x00408B29][Queue@: 0x00408AEC] [Start: 4051025][End: 4215937]

TURN: 7496 READ ZONES:	[0x004119]	3E-0x6	004119 <u>45:</u>	8
B			+2	
[0x00411948-0x	00411 <u>94F</u> :	8 B]	. 2	
[0x00411952-0x	00411959:	8 B]	+3	
[0x0041195C-0x	00411963:	8 B]		
[0x00423DC8-0x	00423DCF:	8 B]		
[0x00423DD2-0x	00423DD9:	8 B]		
[0x00423DDC-0x	00423DE3:	8 B]		
[0x00423DE6-0x0	00423DED:	8 B]		

[0x0012F750-0x0012F753: 4 B] [0x004333D4-0x004333D7: 4 B] [0x004333C8-0x004333CB: 4 B] [0x00423E04-0x00423E0B: 8 B] [0x00433020-0x00433023: 4 B]

WRITE ZONES: [0x0015FA2F-0x0016E46E:
59968 B]
[0x004333D4-0x004333D7: 4 B]
[0x0012F760-0x0012F763: 4 B]

Era1: More original loops (2)

Check the code:



Simple, no?

Era1: More original loops (3)

Check this one :

Seems more complicated!



Era 1:

More original loops(4)

But here are the characteristics we gathered.

Exact same type of algorithm!

We only care about the write zone.

Loop3
[Lead@: 0x00406674][Queue@: 0x00406869] [Start: 1521][End: 495021]
TURN: 2625 READ ZONES: [0x0042B2B9-0x0042B <u>2C0</u> : 8 B]
$[0x0042B2C2-0x0042B2C9: 8 B]^{+2}$
[0x0042B2CB-0x0042B2D2: 8 B] +2
[0x0042B2D4-0x0042B2DB: 8 B]
[0x0042B2DD-0x0042B2E4: 8 B]
[0x0042B2E6-0x0042B2ED: 8 B]
[0x0042B2EF-0x0042B2F6: 8 B]
[0x0042B2F8-0x0042B2FF: 8 B]
[0x0012FAA0-0x0012FAA3: 4 B]
[0x0012F884-0x0012F887: 4 B]
[0x0012F6C8-0x0012F6CB: 4 B]
-
WRITE ZONES: [0x0012FBAC-0x0012FBAF: 4
[0x0042B2B8-0x004304BF: 21000 B]
[0x00433154-0x0043315B: 8 B]
[0x004330F8-0x004330FB: 4 B]
[0x0043313C-0x00433147: 12 B]
[0x0012F404-0x0012F413: 16 B]
[0x00433128-0x0043312F: 8 B]
[0x00433110-0x00433117: 8 B]
[0x0012F3F0-0x0012F3FF: 16 B]

Era 2: Set up the unpacked code



Remember that ?

Era 2: Set up the unpacked code (2)

Let's take a closer look:



A binary tree where the path is built with successive addition plus JZ/JB.

Era 2: Setup the unpacked code (3)

- It has the shape of a binary tree.
- At each node, a 4-bytes value (the counter) is added with itself, then it checks if the result:
 - Is zero (JNZ/JZ)
 - Has overflowed (JB/JNB)
- If the result is zero it takes the next 4-bytes value.
- Somewhere in the function, there are some loops that calculate one byte depending also of the counter (ADC), this is the decrypted byte.
- These functions is implemented differently three times in one Swizzor binary for data, rdata and text sections, but that stays the exact same algorithm!

Era 2: Set up the unpacked code (4)



Era 2: Set up the unpacked code (5)

- As the unpacked binary is normally mapped at 0x400000, it needs to patch all the absolute address.
- A patch table for each dynamic area:

Address	Hee	: du	IMP													
00430AA0	FF	48	69	00	88	08	FF	A0	06	00	00	FF	60	01	00	00
00430AB0	10	10	10	10	24	18	64	08	04	EC	08	04	04	04	04	04
00430AC0	04	04	04	Ø4	0C	FF	98	04	00	00	04	A4	00	00	00	00

Packer miscellaneous

- Checks the kernel32 timestamp against the Windows 95 explorer.exe timestamp!
- Checks the first 4 bytes of the return value of *RtlDecodePointer()* against hardcoded values.
- Looks for certain functions in kernel32 export table by means of signatures and deal with forward exports.
- Looks also in the import table of some modules! For example the ADVAPI32 functions are found in the import table of RPCRT4.

SWIZZOR'S UNPACKED CODE

Hidden Code

- Millions of different files
- Probably all produced by the same gang
 - Droppers
 - Updaters
 - Advertisement delivery
- Many common characteristics

Typical Installation

- 1. <u>Dropper</u> creates registry entries with affiliate ID and software version
- 2. <u>Dropper</u> launches <u>updater</u>
- 3. <u>Updater</u> downloads <u>second stage</u> according to affiliate ID
- 4. <u>Second stage</u> is responsible for ad delivery

Typical Install Process



Code Injection



Code Injection

- str1 = RegQueryValueA(
 - "InternetExplorer.Application");
- str2 = GetModuleFileNameA(NULL);
- str1 = GetShortPathName(str1);
- str2 = GetShortPathName(str2);

if(strcmpA(str1, str2) != 0)
inject_and_exit();

String Encryption



- All strings are encrypted (xor)
- Decrypted "on the fly" before usage
- The first character of the key is indicated by the first 2 chars of the encrypted string
- Same string = multiple encrypted versions

String Decrypting

647B644E9BB73ED09CFC6721AE0D19196E EB186D66B9B204B8D3FDA4700F87FB6EF9

70000019:5.61msn:United States

- Used to encrypt network communication
- XOR key is always the same

Advertisement Delivery

POST /tba/p HTTP/1.1 Content-Length: 289 Content-Type: application/x-www-form-urlencoded User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; 6.0 Accept-Encoding: gzip Host: ads.range159-195.com

guid=29235442840985DE819F8A4B73AA8FC3334E&version= le=F94122913C22&session=B10B&activeWindows=E17B02& B3A7DB6A7C62&launchCount=9E3962HTTP/1.1 200 OK Server: Resin/3.0.18 Content-Language: en-CA Content-Type: application/octet-stream Connection: close Transfer-Encoding: chunked Date: Tue, 15 Jun 2010 15:01:40 GMT

Advertisement

WHY PAY MORE FOR NORTON OR MCAFEE?



The Shield Deluxe 2010 provides superior protection at half the price!

	The Shield Deluxe 2010	Norton AntiVirus 2010	McAfee VirusScan Plus 2010
Antivrius Protection	1	1	1
Anti Spyware Protection	1	1	1
Antiphishing Protection	1	1	1
Browser Protection	1	1	1
Email Protection	1	1	1
1 Year of Free Antivirus Updates	1	1	1
1 Year of Free AntiSpyware Updates	1	1	1
FREE Tech Support	1		
Priced Under \$30.00	1		

PRODUCT SUMMARY

The Shield Deluxe 2010 provides superior Proactive Protection from Viruses, Spyware, and other e-Threats ... that won't slow your PC down!

The Shield Deluxe 2010, powered by BitDefender award winning Antivirus engine, provides advanced proactive protection against viruses, spyware, phishing attacks and identity theft. Stay one step ahead of the latest e-Threats while maintaining superior performance that keeps your PC running smoothly. The Shield Deluxe is simple to install and set up, while offering advanced users a range of versatile settings for fine-tuning the program.



Five new malware samples are found every 2 minutes. If your security software expired yesterday, you are

Updater

http://%s/bins/int/7k42_up2.int

- References to all affiliate IDs
- Generate unique installation ID
- Contacts LOP servers



Host File Modifications

"" seg000:0014E270	00000061	С	host255-255-255-0.com
"" seg000:0014E2D8	00000059	С	host192-168-1-2.com
"" seg000:0014E338	00000051	С	host127-0-0-1.com
"" seg000:0014E38C	00000031	С	host-domain-lookup.com
"" seg000:0014E3C0	00000025	С	netbios-wait.com
"" seg000:0014E3E8	000000	С	cidhelp.com
"" seg000:0014E404	00000023	С	dns-look-up.com
"" seg000:0014E428	000000	С	startnow.com
"" seg000:0014E448	000000	С	cc214142.com
"" seg000:0014E468	00000021	С	zone-media.com
"" seg000:0014E48C	000000	С	revenue.net
"" seg000:0014E4A8	00000029	С	adintelligence.net
"" seg000:0014E4D4	000000	С	msgplus.net
"" seg000:0014E4F0	000000	С	patchou.com
"" seg000:0014E50C	00000021	С	look-today.com
"" seg000:0014E530	0000001F	С	search200.com
"" seg000:0014E550	00000029	С	mastersearcher.com
"" seg000:0014E57C	000000	С	ifsearch.com
"" seg000:0014E59C	00000025	С	searchhotsex.com
"" seg000:0014E5C4	0000001F	С	searchexe.com
"" seg000:0014E5E4	0000001F	С	searchbee.net
"" seg000:0014E604	00000025	С	prosearching.com
"" seg000:0014E62C	00000021	С	opensearch.org
"" seg000:0014E650	00000023	С	omegasearch.com
"" seg000:0014E674	00000027	С	netsearchsoft.com
"" seg000:0014E69C	00000025	С	iwantosearch.com
"" seg000:0014E6C4	00000023	С	isearchhere.com
"" seg000:0014E6E8	00000025	С	intelesearch.com

- Upon installation, etc/host file is modified
- Domain blacklist is removed
- If you can decrypt the strings, you have a complete list of domains related to this company

Dark Connections



C2 Media / LOP.com

- Advertising:
 - Pop ups
 - Toolbars
 - Search engine
- All software delivered by this company uses Swizzor type obfuscation (even their uninstaller)

GodLikeProductions.com

- Conspiracy theorist discussion forum
- Bought by lop.com, probably to distribute advertisement and attract traffic
- Change post contents
 - Bunny = lop.com
 - Flower = spyware
- Reachable from lop.com (chat page)

Conclusions

- Complex target
 - Millions of (sometimes useless) instructions
 - Multiple binaries per installation
- Solutions
 - Enhanced tracing
 - Visualization
- Fun!

Pierre-Marc Bureau – bureau@eset.sk Joan Calvet - j04n.calvet@gmail.com





