UNDERSTANDING SWIZZOR’S OBFUSCATION

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

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

  1. 98be86967...
  2. tp_map16
  3. staA

- 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...
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:
  - **Tracing step**: once per binary, it outputs two files:
    - Improved trace: detailed view.
    - Events file: high level view.
  - **Analysis step**: 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 [http://www.pintool.org/](http://www.pintool.org/)

- 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.
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:
    
    ```
    INT D4
    CHAR* SA
    PDWORD I4
    ...
    ```

  - Complex types:
    
    ```
    SECURITY_ATTRIBUTES D[DWORD, LPVOID, BOOL]
    LPSECURITY_ATTRIBUTES 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

PIN TOOL POINT OF VIEW

<table>
<thead>
<tr>
<th>EXECUTED</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTRUCTION1</td>
<td>1</td>
</tr>
<tr>
<td>INSTRUCTION5</td>
<td>2</td>
</tr>
<tr>
<td>INSTRUCTION6</td>
<td>3</td>
</tr>
<tr>
<td>INSTRUCTION2</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>INSTRUCTION3</td>
<td>5</td>
</tr>
<tr>
<td>INSTRUCTION5</td>
<td>6</td>
</tr>
<tr>
<td>INSTRUCTION6</td>
<td>7</td>
</tr>
</tbody>
</table>

This is not a loop! So what’s a loop?
Loops: How to detect them? (3)

(SIMPLIFIED) STATIC POINT OF VIEW

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 at least 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] 0x00000001
    | A3: [LPCWSTR] "XJLFOQ"
    | RV: [HANDLE] 0x00000000
...
[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] 0x00000018
    | 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]
  | M: calcul4
  | W: 0x0012FB8C

[=> EVENT: API CALL <=][TIME: 299][@: 0x00402FC2]
  | F: malloc
  | A1: [SIZE_T] 0x00002A84
  | RV: [VOID*] 0x023A6E38

[=> EVENT: LOOP <=][START:634 - END:1381][LEAD@:0x0040F62A - TAIL@:0x0040F41C]
  | 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]
  | 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: 
  
SMALL UNIT OF TIME

BIG UNIT OF TIME

TIME
**Loop2**

[Lead@: 0x0040F62A][Queue@: 0x0040F41C][Start: 634][End: 1381]

| TURN: 57 |
| READ ZONES: [0x0042A8A5-0x0042A8EC: 72 B] |
| [0x0042A579-0x0042A5F4: 124 B] |
| [0x00426234-0x0042623F: 12 B] |
| [0x00428440-0x00428447: 8 B] |
| [0x004330C0-0x004330C3: 4 B] |
| [0x004286C9-0x004286D4: 12 B] |
| [0x004286F1-0x0042876C: 124 B] |
| [0x00432F20-0x00432F23: 4 B] |

| WRITE ZONES: [0x0042A8A5-0x0042A8EC: 72 B] |
| [0x0042A579-0x0042A5F4: 124 B] |
| [0x00426234-0x0042623F: 12 B] |
| [0x00428440-0x00428447: 8 B] |
| [0x004286C9-0x004286D4: 12 B] |
| [0x004330C0-0x004330C3: 4 B] |
Enough with the tools, what about the packer?
Era 0: FUD

Useless malloc!
Era 1: Prepare the packer

Example of simple loop
Era 1: Example of simple loop (2)

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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Read</th>
<th>Write</th>
<th>Call/Jmp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4284be</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x4284ca</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x4284ce</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x42850a</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x4284a8</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x42951f</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x428212</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x428499</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x42a360</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x42a678</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Era 1: Example of simple loop (3)
Era 1:
More original loops

- Read clusters jump over 3 bytes!

- Big write zone.
Era 1: More original loops (2)

- Check the code:

```assembly
loc_403029:
lea   esi, [esi+2]
mov   [ebp-5Ch], eax
mov   [ebp-4Ch], edi
mov   edi, dword_433029
xor   eax, eax
or    eax, [esi+edi]
sub   edx, 3
jg    loc_403ABA

true

loc_403ABA:
lea   esi, [esi+4]
mov   dword_433004, edx
mov   edx, dword_433068
mov   [ecx+edx], eax
mov   edx, [esi+edi]
lea   ecx, [ecx+4]
add   esi, 4
mov   eax, dword_4330C8
mov   [ecx+eax], edx
lea   ecx, [ecx+4]
mov   edx, dword_4330D4
mov   eax, [ebp-5Ch]
mov   edi, [ebp-4Ch]
jmp   short loc_403029
```

Simple, no?
Era 1: 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.
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:

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Hex_dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>00430AA0</td>
<td>FF 48 09 00 00 08 FF A0 06 00 00 FF 60 01 00 00</td>
</tr>
<tr>
<td>00430AB0</td>
<td>10 10 10 10 24 18 04 08 04 EC 08 04 04 04 04</td>
</tr>
<tr>
<td>00430AC0</td>
<td>04 04 04 04 0C FF 98 04 00 00 04 A4 00 00 00 00</td>
</tr>
</tbody>
</table>
```
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. **Dropper** creates registry entries with affiliate ID and software version
2. **Dropper** launches ** updater**
3. **Updater** downloads **second stage** according to affiliate ID
4. **Second stage** is responsible for ad delivery
Typical Install Process

- Adware Delivery
- Updater
- Dropper
Code Injection

Diagram:
- Deobfuscated code start
  - GetCommandLineA
  - Get name of Internet Explorer from registry
    - Compare
      - Start Internet Explorer
      - Execute payload
        - WriteProcessMemory
          - Patch process
            - Start injected binary in Internet Explorer
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

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

647B644E9BB73ED09CFC6721AE0D19196E
EB186D66B9B204B8D3FDA4700F87FB6EF9

70000019:5.61msn:United States
Advertisement Delivery

```plaintext
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=E17B028
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
```
### WHY PAY MORE FOR NORTON OR MCAFEE?

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Shield Deluxe 2010</th>
<th>Norton Antivirus 2010</th>
<th>McAfee VirusScan Plus 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antivirus Protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Anti Spyware Protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Antiphishing Protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Browser Protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Email Protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1 Year of Free Antivirus Updates</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1 Year of Free AntiSpyware Updates</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FREE Tech Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Priced Under $30.00</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**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

- 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

Tin Foil Hat Area
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