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

UNDERSTANDING SWIZZOR'S OBFUSCATION

ÉCOLE POLYTECHNIQUE MONTRÉAL

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…

```
00408365 mov     [esp+18h+arg_58], eax
00408369 push    119h
0040836E call    sub_402DB3
00408373 mov     dword_43363C, eax
00408378 push    1
0040837A call    sub_40803A
0040837F mov     [esp+18h+arg_4C], eax
00408383 mov     edi, dword_433640
00408389 sub     esp, 8
0040838C push    edi
0040838D call    sub_403258
00408392 mov     dword_433654, eax
00408397 push    125h
0040839C call    sub_402DB3
004083A1 jmp     loc_408D0B
```

```
00408D0B ; START OF FUNCTION CHUNK FOR sub_408CB4
00408D0B
00408D0B loc_408D0B:
00408D0B mov     [esp+20h+arg_5C], eax
00408D12 push    55F05307h
00408D17 call    sub_40390E
00408D1C push    20h
00408D1E call    sub_408F7E
00408D23 mov     dword ptr [ebp-380h], 1D0h
00408D2D sub     esp, 0Ch
00408D30 call    sub_402BF3
00408D35 mov     dword_433664, eax
00408D3A push    8Ah
00408D3F call    sub_40383C
00408D44 mov     [esp+8+arg_4], eax
00408D48 push    dword_433670
00408D4E call    sub_403258
00408D53 mov     dword_433648, eax
00408D58 push    1D4h
00408D5D call    sub_402BF3
00408D62 mov     [esp+8], eax
00408D66 push    3Ch
00408D68 call    sub_402D8E
00408D6D mov     dword_433658, eax
00408D72 push    96h
00408D77 call    sub_40383C
00408D7C mov     [esp+8+arg_58], eax
00408D80 push    0FFF58CCFh
00408D85 call    sub_402DFF
```

```
00403258 push    edi
00403259 mov     edx, [esp+4+arg_0]
0040325D sub     edx, 13Eh
00403263 lea     edi, unk_433564
00403269 mov     eax, off_42C554[edx]
0040326F mov     ecx, 28B3h
00403274 add     ecx, eax
00403276 sub     ecx, [edi]
00403278 mov     eax, ecx
0040327A pop     edi
0040327B retn    4
```

```
00408F7E mov     dword_433660, eax
00408F83 push    19Ah
00408F88 call    sub_403258
00408F8D mov     dword_433668, eax
00408F92 push    14Ah
00408F97 call    sub_403258
00408F9C jmp     loc_4090F6
```

```
00402BF3 push    edi
00402BF4 mov     edx, [esp+4+arg_0]
00402BF8 sub     edx, 1B4h
00402BFE lea     ecx, unk_433534
00402C04 mov     eax, dword_4359D8[edx]
00402C0A mov     edi, 19B7DBh
00402C0F sub     edi, eax
00402C11 add     edi, [ecx]
00402C13 mov     eax, edi
00402C15 pop     edi
00402C16 retn    4
```

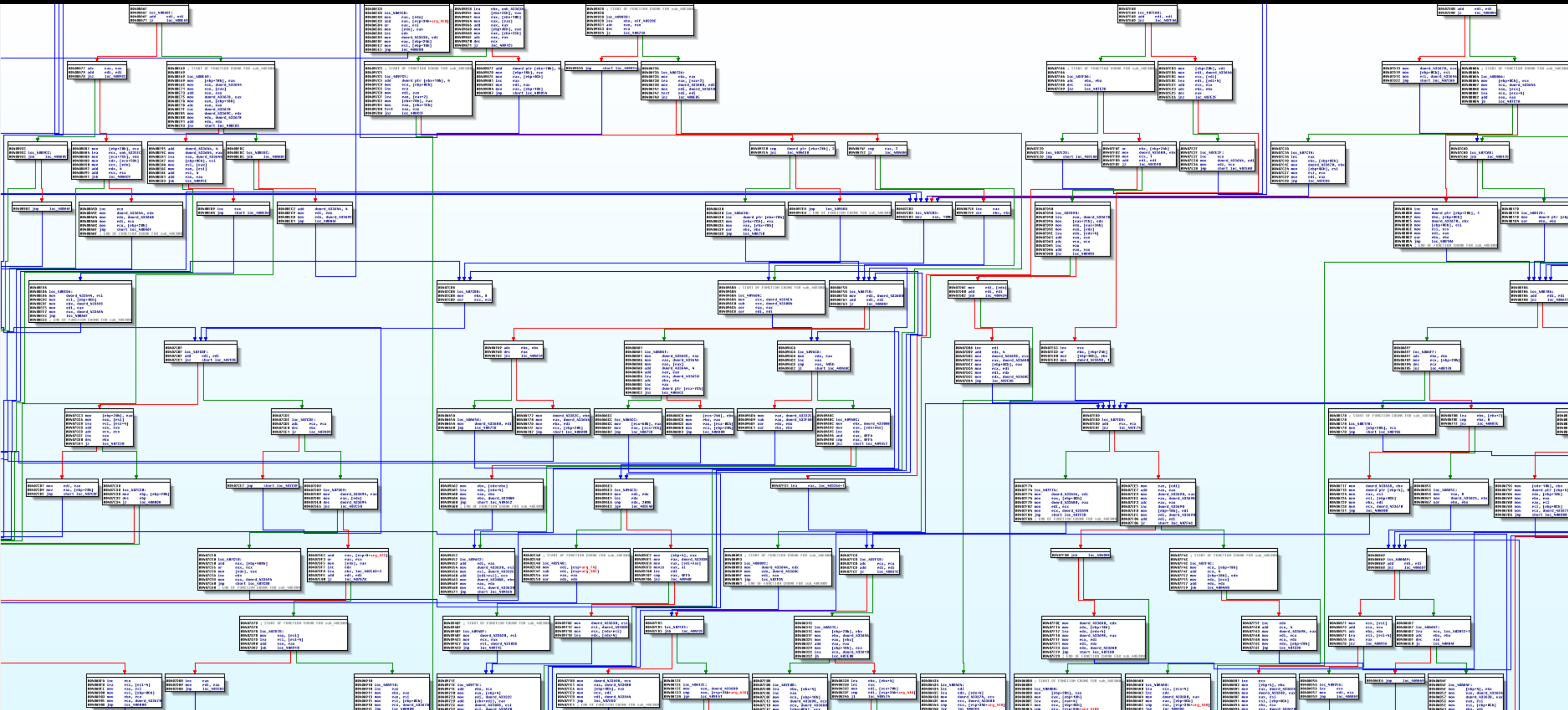
```
00402D8E push    edi
00402D8F push    ebx
00402D90 mov     ebx, [esp+8+arg_0]
00402D94 sub     ebx, 30h
00402D97 mov     eax, dword_4327EC[ebx]
00402D9D lea     edi, unk_43356C
00402DA3 mov     ecx, 2416B5h
00402DA8 sub     ecx, eax
00402DAA add     ecx, [edi]
00402DAC mov     eax, ecx
00402DAE pop     ebx
00402DAF pop     edi
00402DB0 retn    4
```

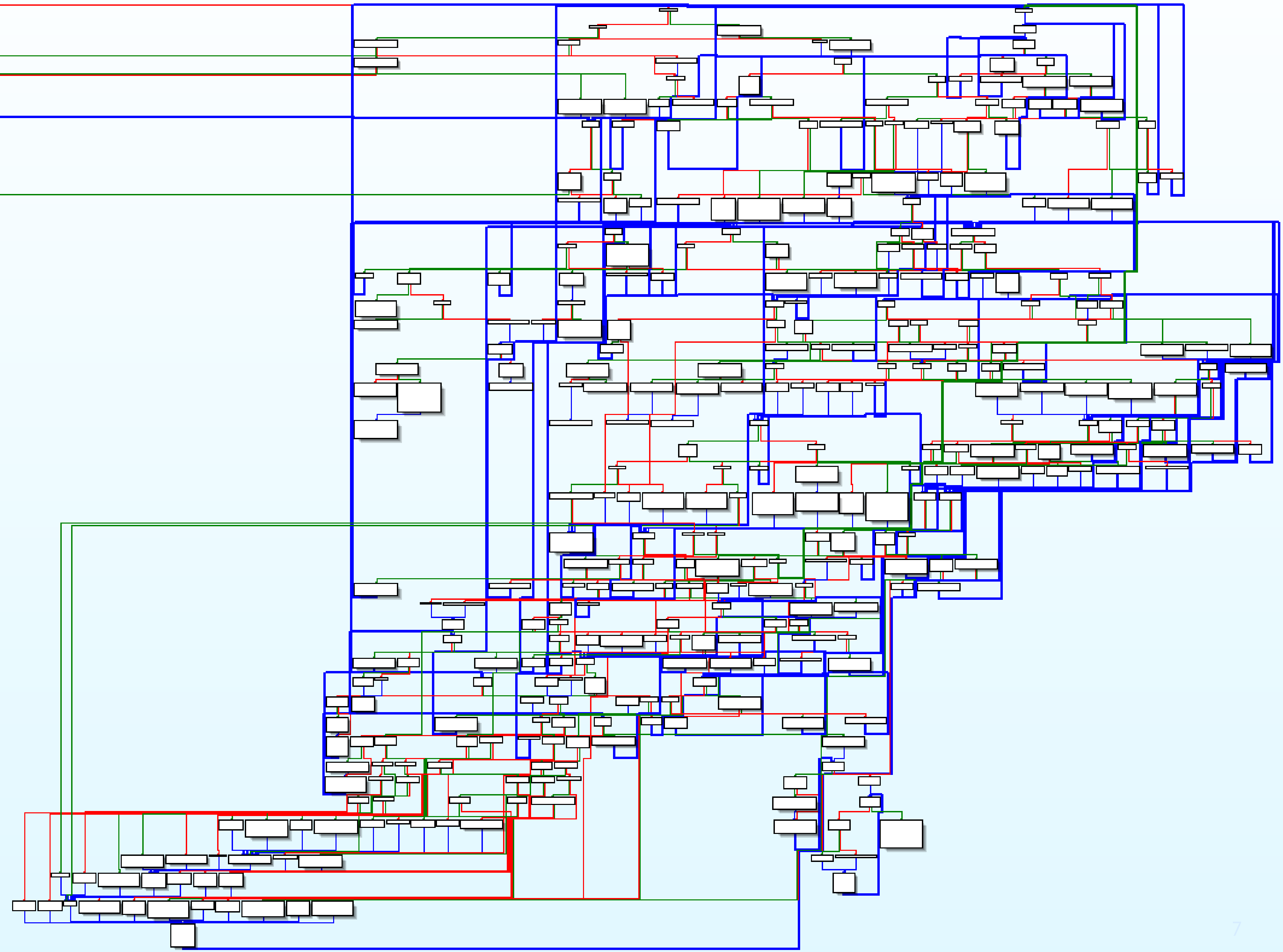
```
00402DB3 push    esi
00402DB4 push    ebx
00402DB5 mov     ebx, [esp+8+arg_0]
00402DB9 sub     ebx, 0FDh
00402DBF mov     eax, dword_437298[ebx]
00402DC5 lea     edx, dword_433530
00402DCB mov     esi, 2FBDh
00402DD0 xor     esi, eax
00402DD2 add     esi, [edx+18h]
00402DD5 sub     esi, [edx+44h]
00402DD8 mov     eax, esi
00402DDA pop     ebx
00402DDB pop     esi
00402DDC retn    4
```

```
0040390E push    edi
0040390F push    ebp
00403910 mov     ebp, esp
00403912 mov     edi, 55ED427Ah
00403917 sub     edi, [ebp+arg_0]
0040391A lea     edi, [edi+638A6h]
00403920 add     edi, (offset loc_403005+3)
00403926 mov     eax, edi
00403928 pop     ebp
00403929 pop     edi
0040392A retn    4
```

```
0040383C push    esi
0040383D mov     edx, [esp+4+arg_0]
00403841 sub     edx, 2Ah
00403844 lea     ecx, unk_433594
0040384A mov     eax, dword_435F38[edx]
00403850 mov     esi, 134E3Ch
00403855 sub     esi, eax
00403857 add     esi, [ecx]
00403859 add     esi, offset loc_403E9C
0040385F mov     eax, esi
00403861 pop     esi
00403862 retn    4
```

```
00402DFF push    esi
00402E00 push    ebp
00402E01 mov     ebp, esp
00402E03 mov     ecx, 141003h
00402E08 sub     ecx, [ebp+arg_0]
00402E0B lea     esi, dword_433530
00402E11 mov     edx, [esi+2Ch]
00402E14 xor     ecx, edx
00402E16 mov     eax, [esi+54h]
00402E19 xor     ecx, eax
00402E1B sub     ecx, [esi+6Ch]
00402E1E add     ecx, offset loc_403088
00402E24 mov     eax, ecx
00402E26 pop     ebp
00402E27 pop     esi
00402E28 retn    4
```





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/

- **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:

```
INT          D4
CHAR*        SA
PDWORD       I4
…
```

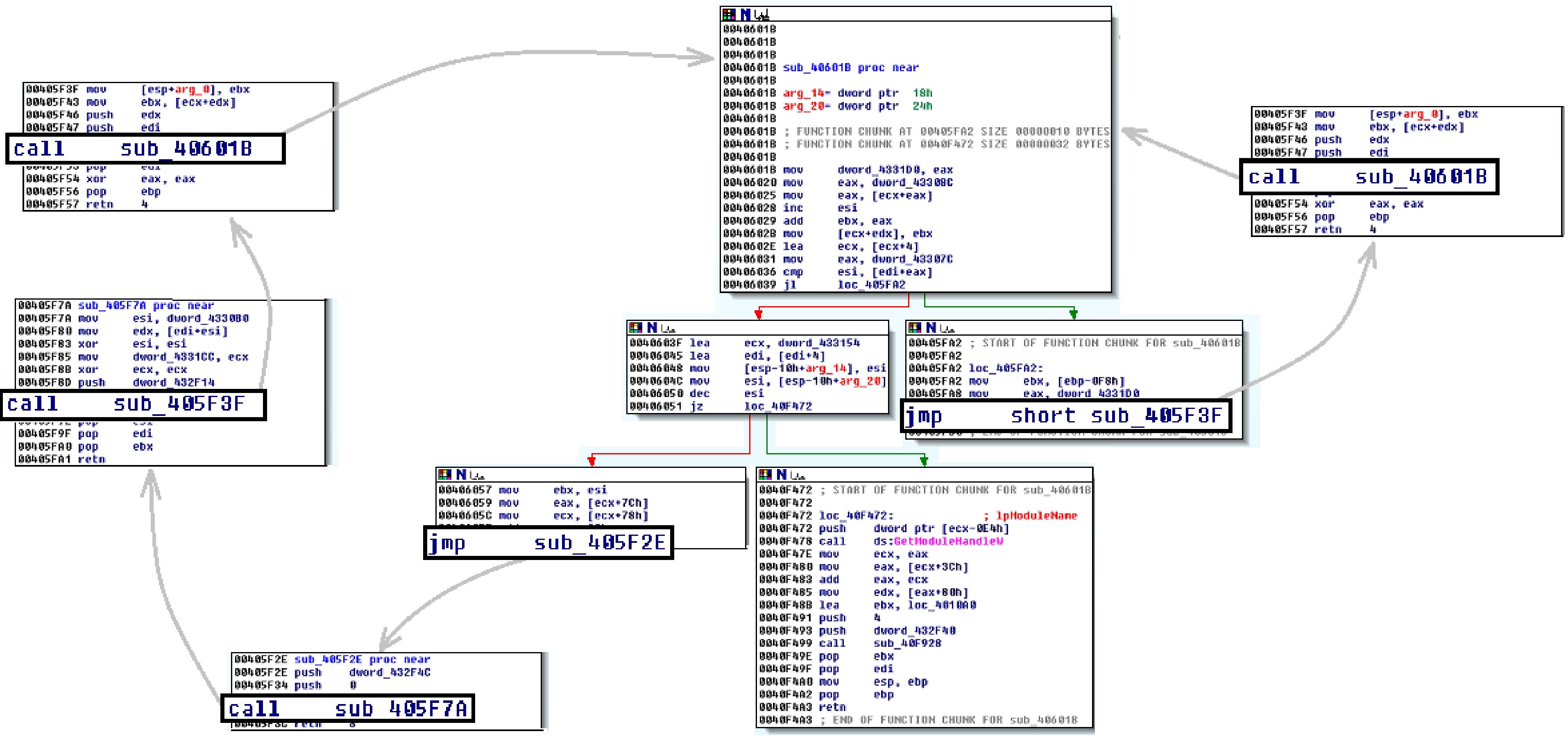
    - Complex types:

```
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.

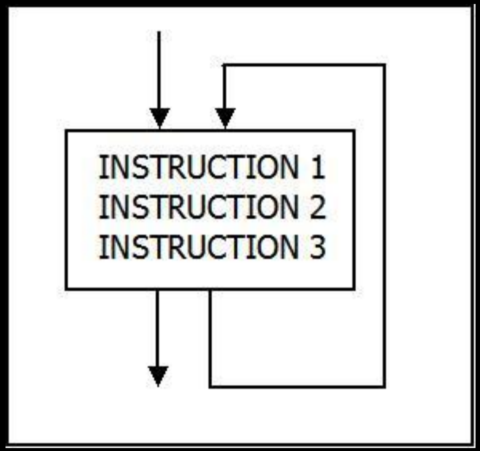
Loops in Swizzor!



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

Loops: How to detect them ? (1)

(SIMPLIFIED) STATIC POINT OF VIEW



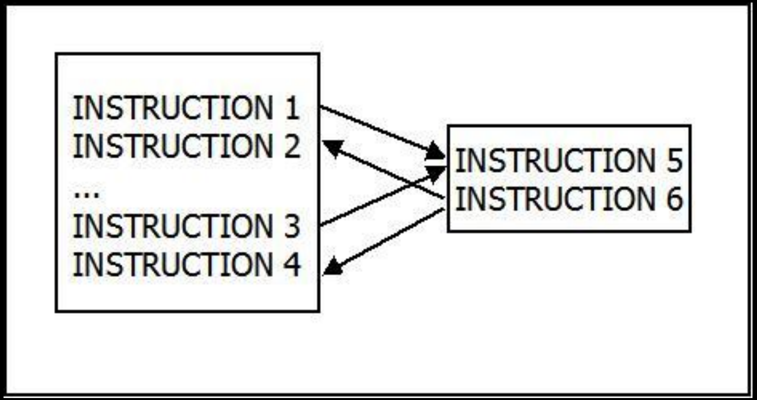
PIN TOOL POINT OF VIEW

EXECUTED	TIME
INSTRUCTION1	1
INSTRUCTION2	2
INSTRUCTION3	3
INSTRUCTION1	4
INSTRUCTION2	5
...	...

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



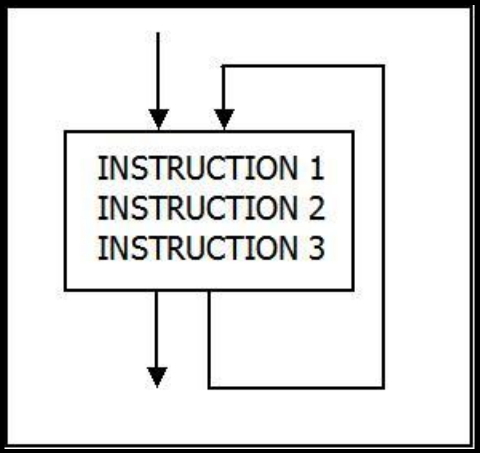
PIN TOOL POINT OF VIEW

EXECUTED	TIME
INSTRUCTION1	1
INSTRUCTION5	2
INSTRUCTION6	3
INSTRUCTION2	4
...	...
INSTRUCTION3	5
INSTRUCTION5	6
INSTRUCTION6	7

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
INSTRUCTION2	2
INSTRUCTION3	3
INSTRUCTION1	4
INSTRUCTION2	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 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)
```

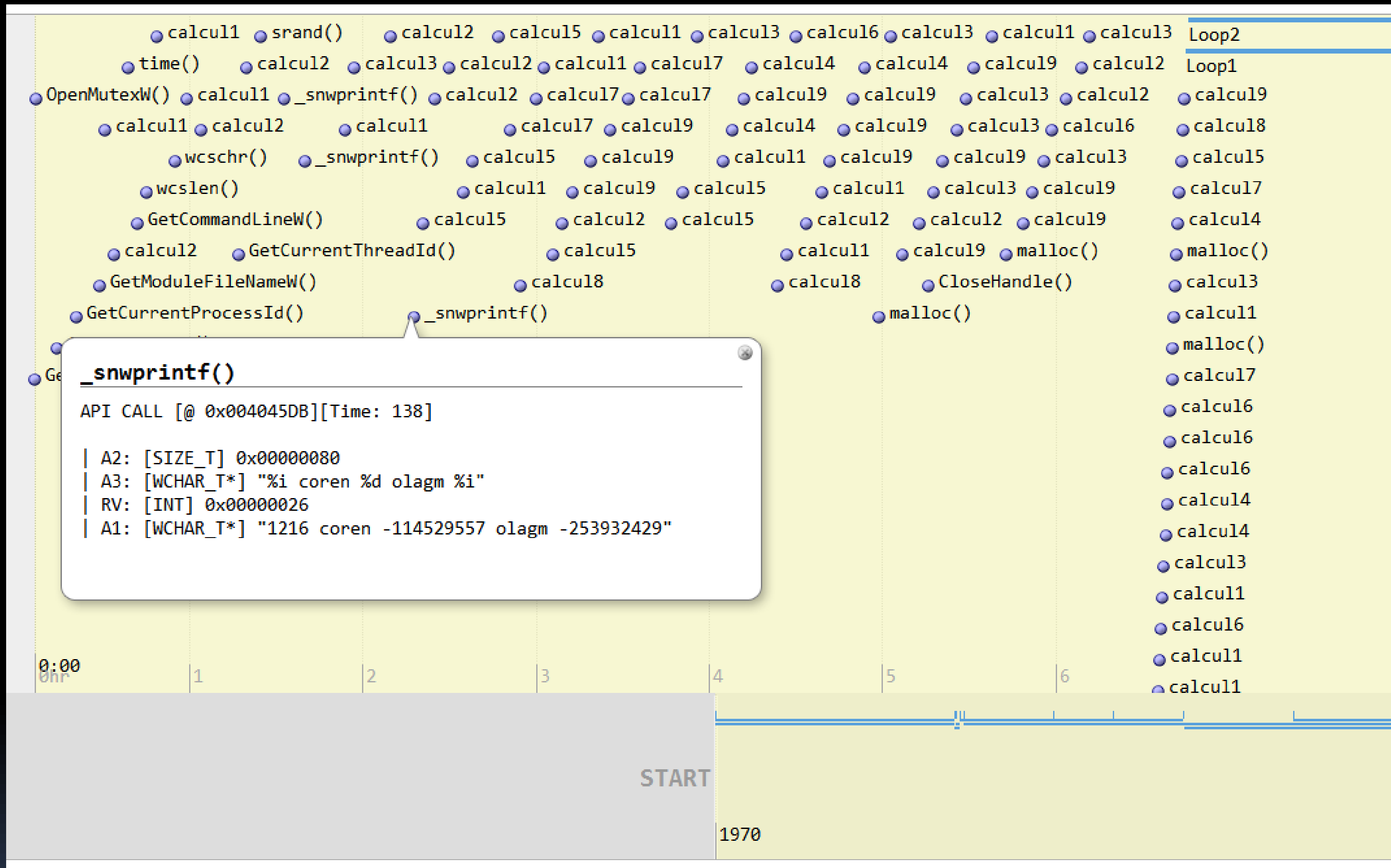
Output 2: timeline!

- **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 :

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

SMALL UNIT OF TIME

BIG UNIT OF TIME

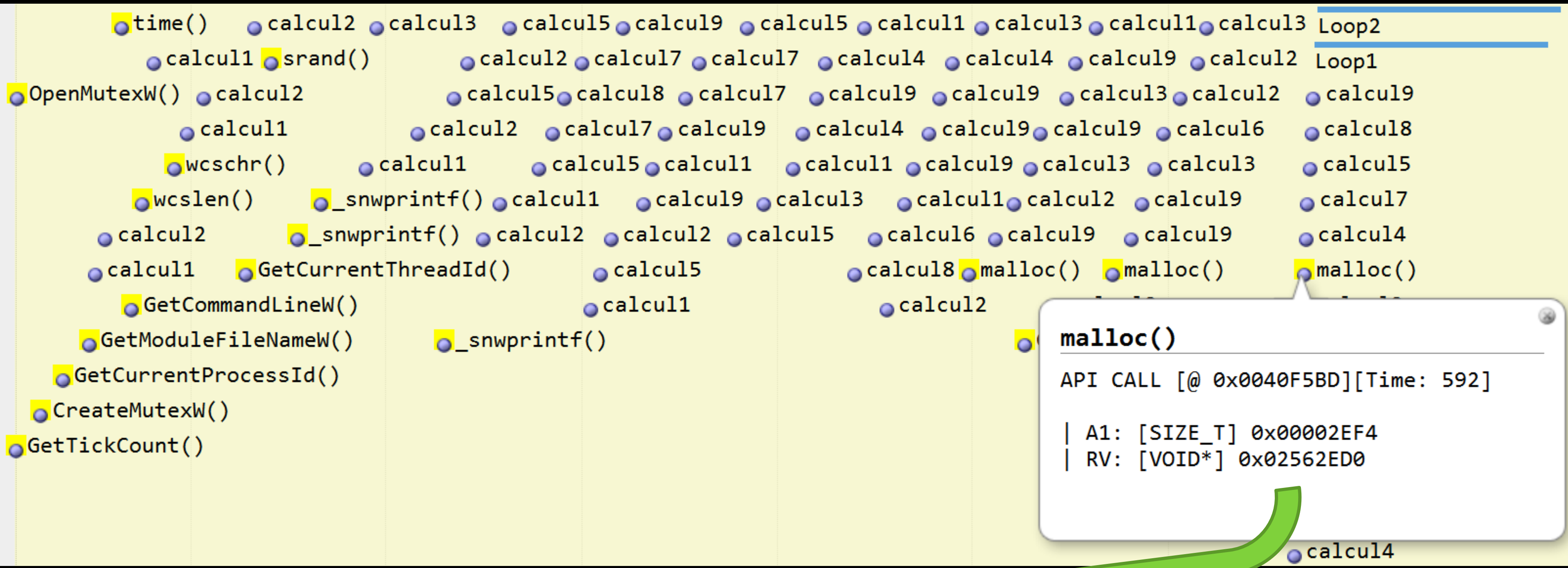


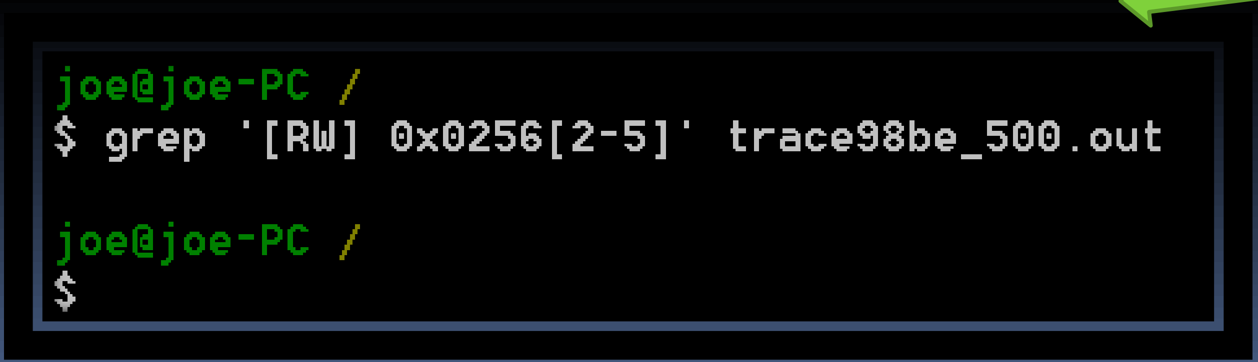
TIME


calcul3
calcul1
calcul3
Loop2
Loop4
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]
calcul1
calcul6
5
6
calcul1
calcul2
START
1970

Enough with the tools, what about the packer?

Era 0: FUD

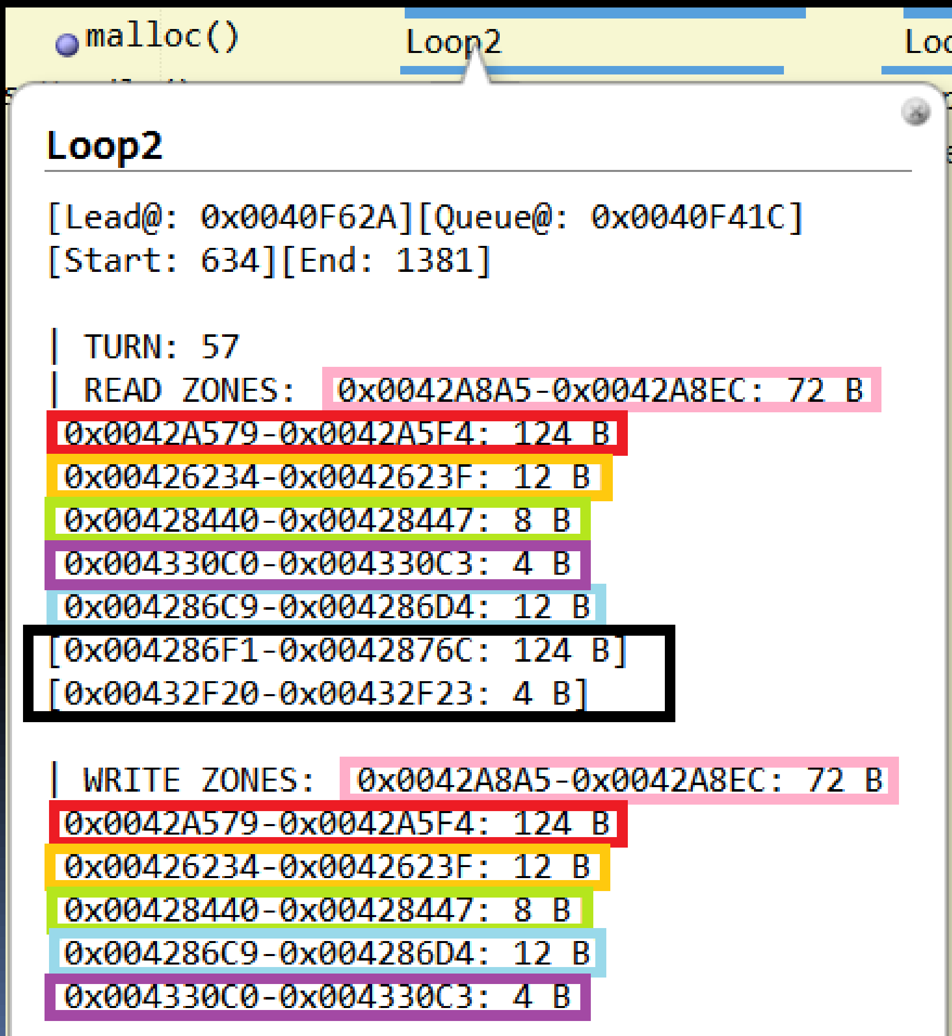


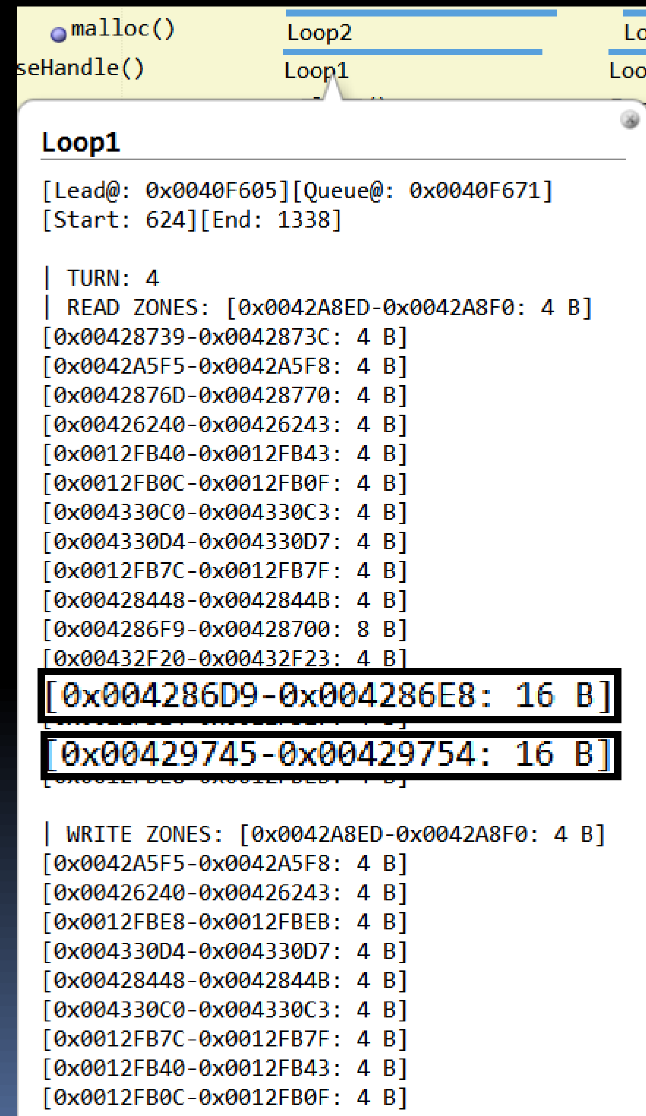


Useless malloc !

Era 1: Prepare the packer

Example of simple loop





Era 1: Example of simple loop (2)

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

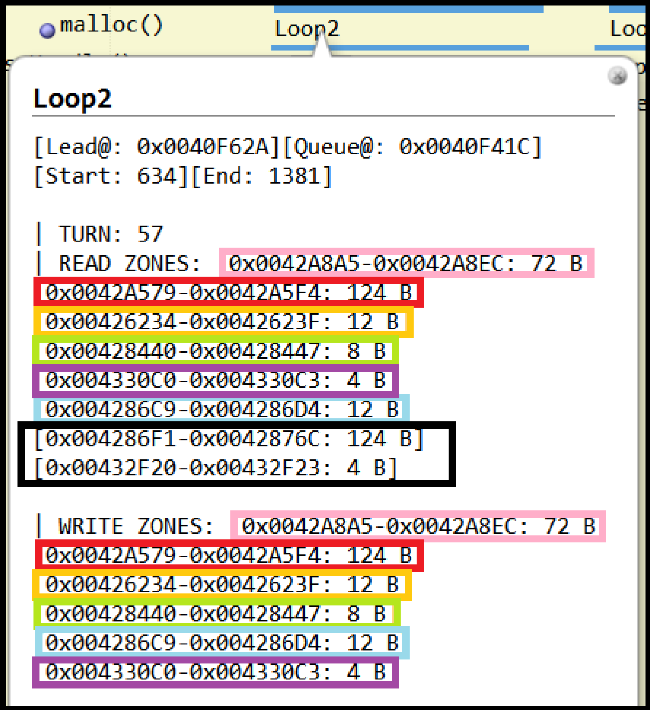
```
0x4284be - [5 0 0]
0x4284ca - [3 0 0]
0x4284ce - [2 0 0]
0x42850a - [1 0 0]
0x4284a8 - [1 0 0]
0x42951f - [1 0 0]
0x428212 - [1 1 0]
0x428499 - [1 1 0]
0x42a360 - [1 1 0]
0x42a678 - [1 1 0]
```

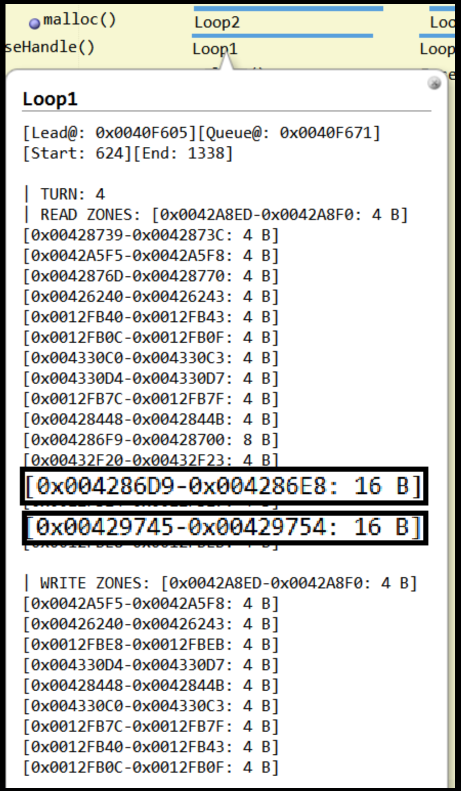
KEY

DECRYPTED AREAS

CONTROL STRUCTURES

Era 1: Example of simple loop (3)

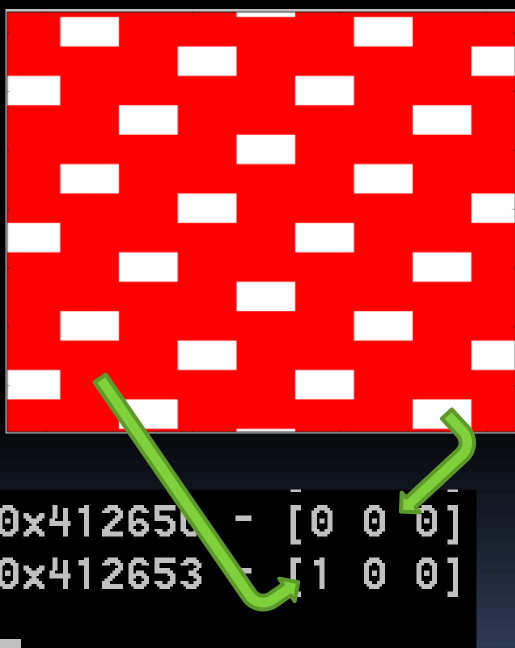





```
keyHeuristic... Done
0x426234-0x426244 [DECRYPTED BY KEY 0] on 16 bytes
0x428440-0x42844c [DECRYPTED BY KEY 0] on 12 bytes
0x4286c9-0x4286d5 [DECRYPTED BY KEY 0] on 12 bytes
0x4286d9-0x4286ed [CONTROL STRUCTURES FOR KEY 0x4286f1]
0x4286f1-0x428771 [KEY 0] 128 bytes
0x429745-0x429759 [CONTROL STRUCTURES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



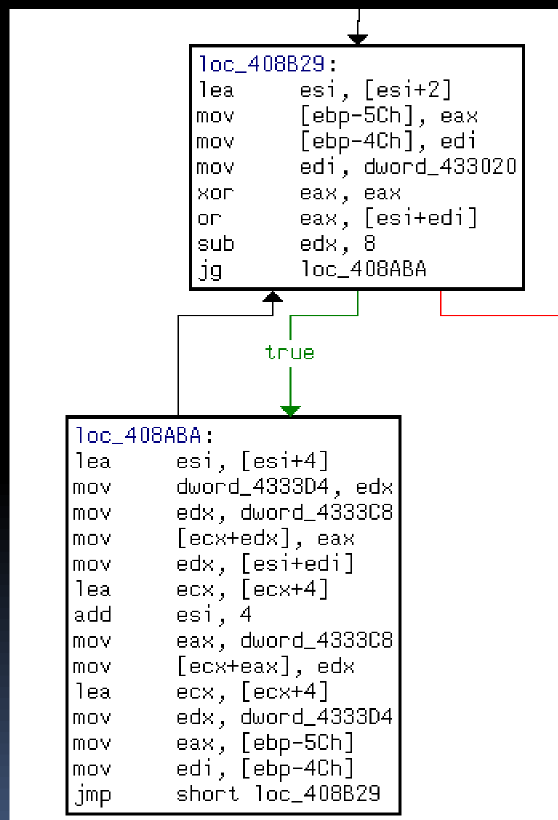
- Big write zone.

Loop28

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

| TURN: 7496
| READ ZONES: [0x0041193E-0x00411945: 8
B]
[0x00411948-0x0041194F: 8 B]   +3
[0x00411952-0x00411959: 8 B]   +3
[0x0041195C-0x00411963: 8 B]
[0x00423DC8-0x00423DCF: 8 B]
[0x00423DD2-0x00423DD9: 8 B]
[0x00423DDC-0x00423DE3: 8 B]
[0x00423DE6-0x00423DED: 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]
[0x0012F750-0x0012F753: 4 B]
```

Era 1: More original loops (2)

- Check the code:

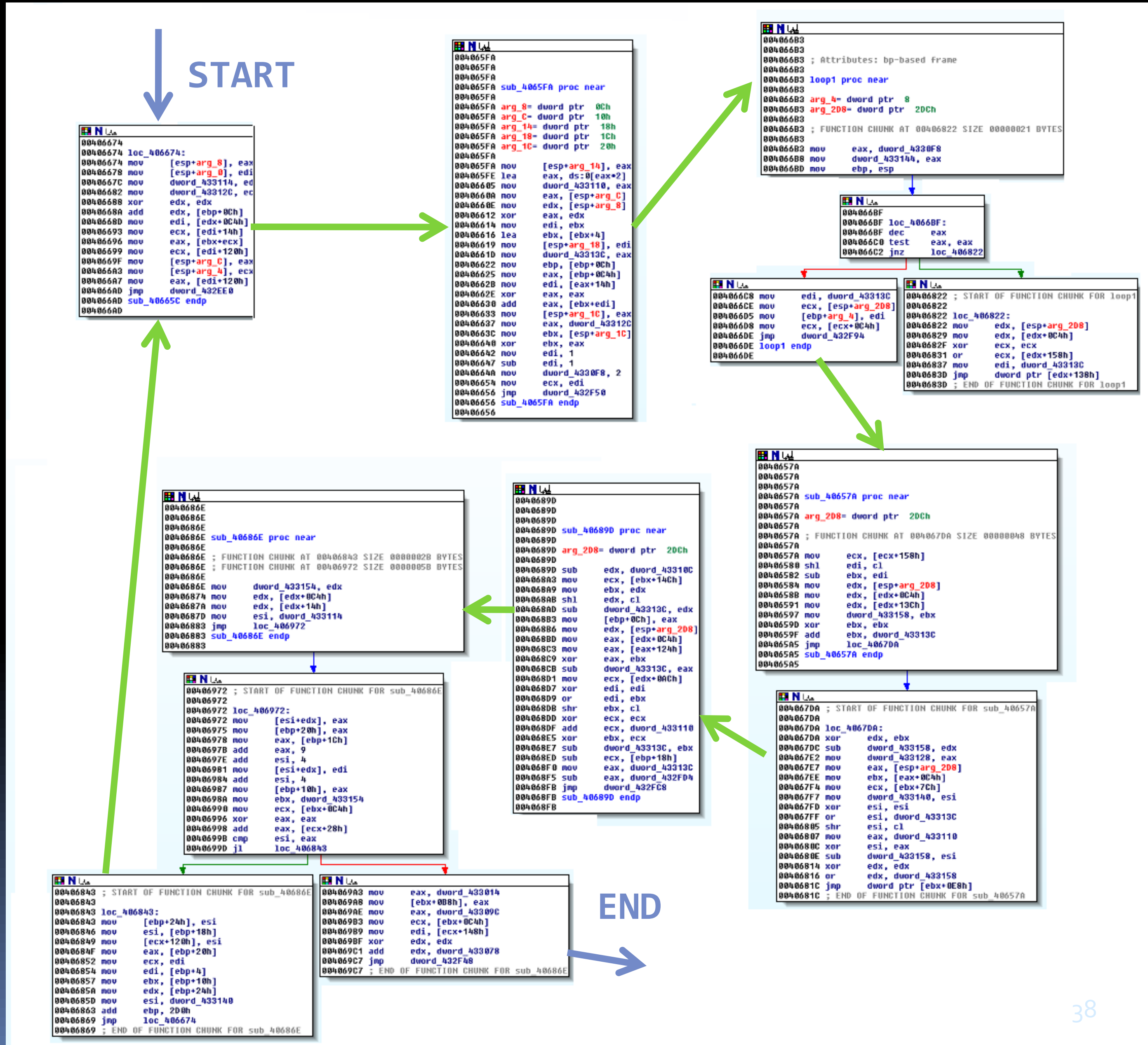


Simple, no ?

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.

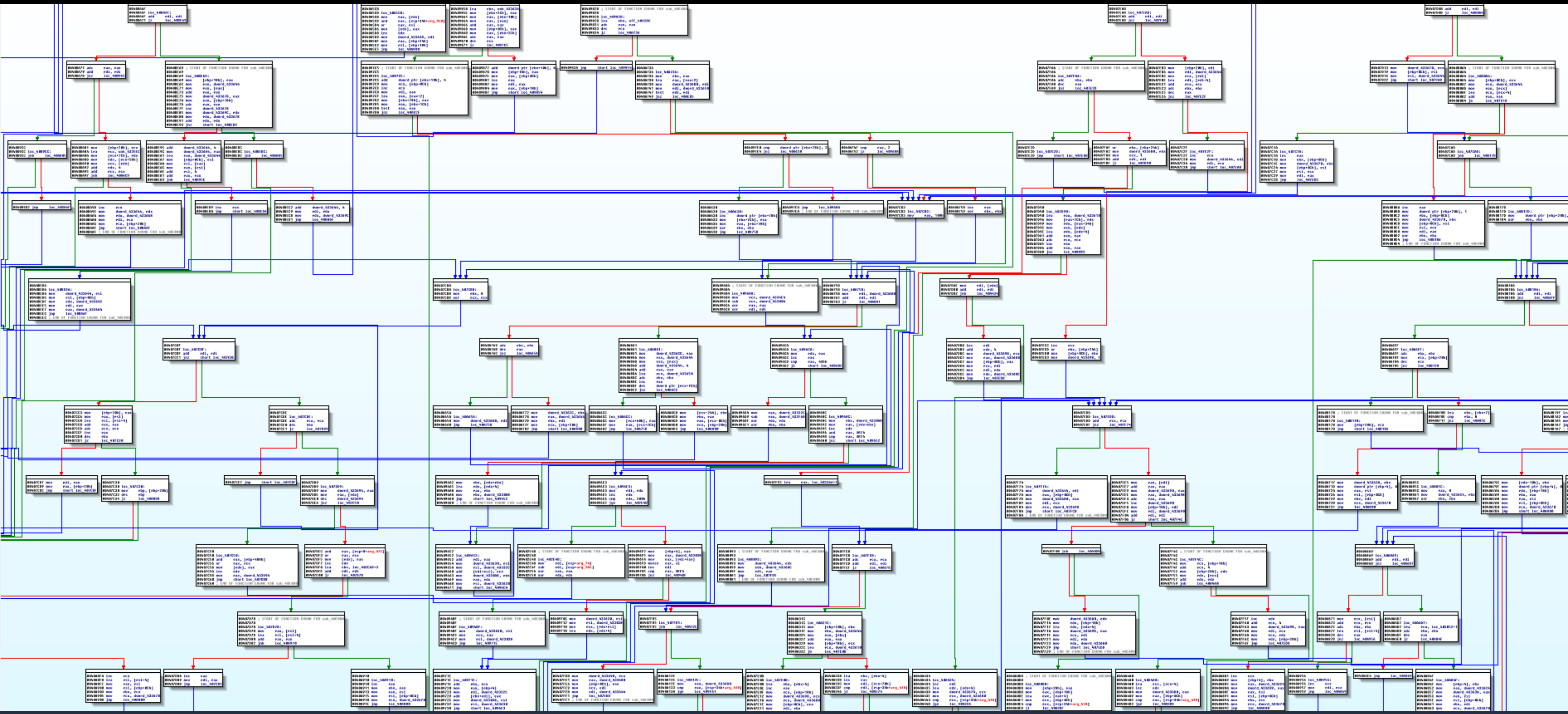
Exact same type of algorithm!

We only care about the write zone.

Loop3

```
[Lead@: 0x00406674][Queue@: 0x00406869]
[Start: 1521][End: 495021]

| TURN: 2625
| READ ZONES: [0x0042B2B9-0x0042B2C0: 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
B]
[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]
```

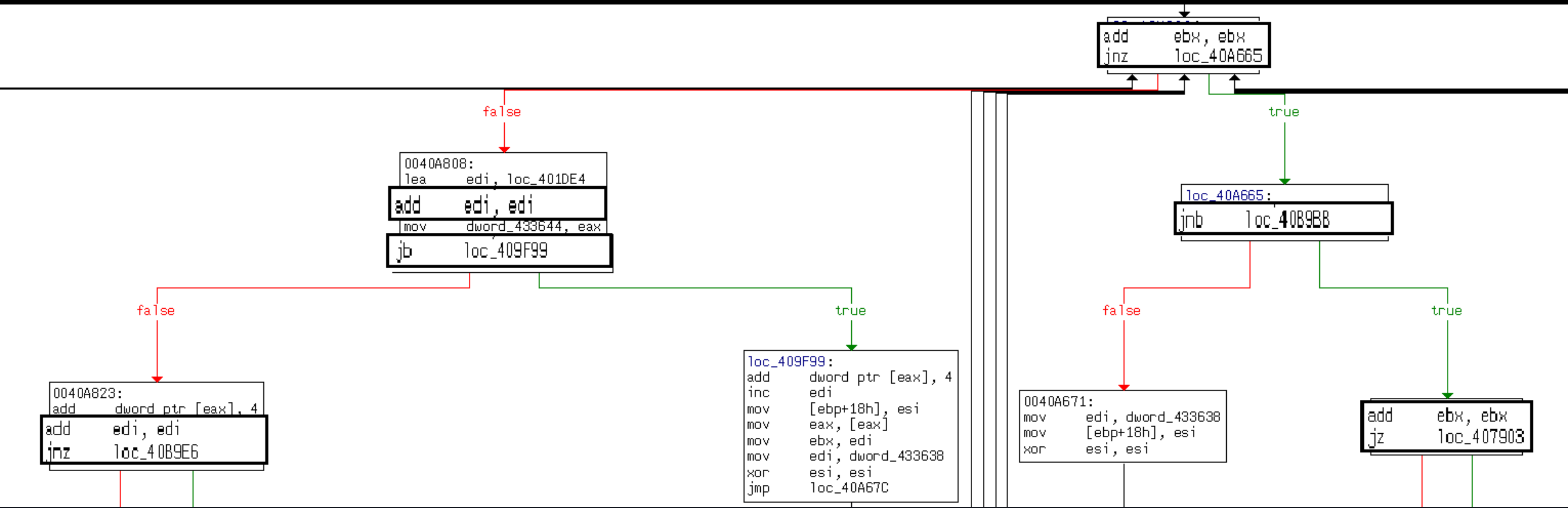
Era 2: Set up the unpacked code



Remember that ?

Era 2: Set up the unpacked code (2)

Let's take a closer look:

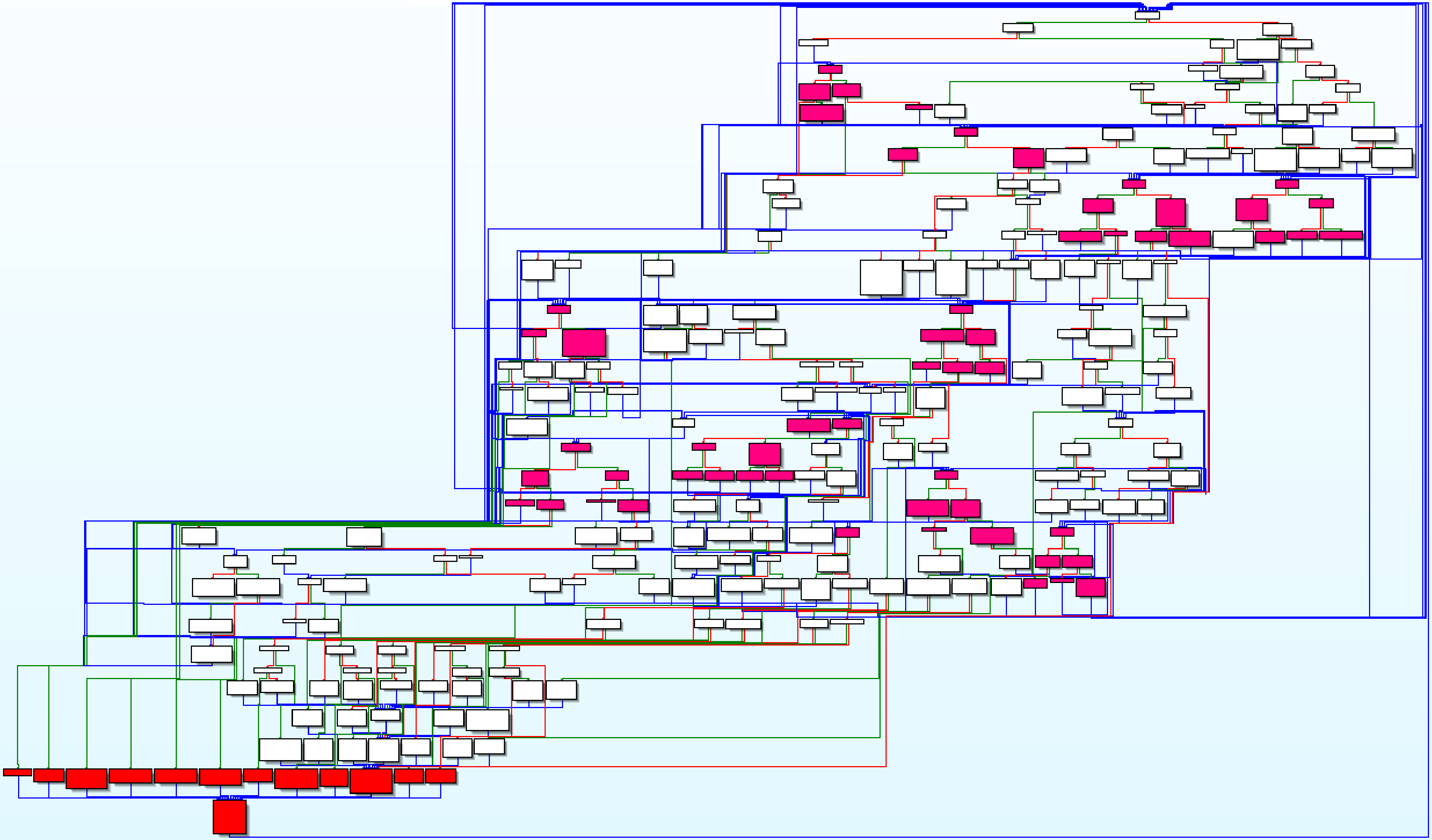


A binary tree where the path is built with successive addition plus JZ/JB.

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	Hex dump
00430AA0	FF 48 09 00 00 08 FF A0 06 00 00 FF 60 01 00 00
00430AB0	10 10 10 10 24 18 04 08 04 EC 08 04 04 04 04 04
00430AC0	04 04 04 04 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

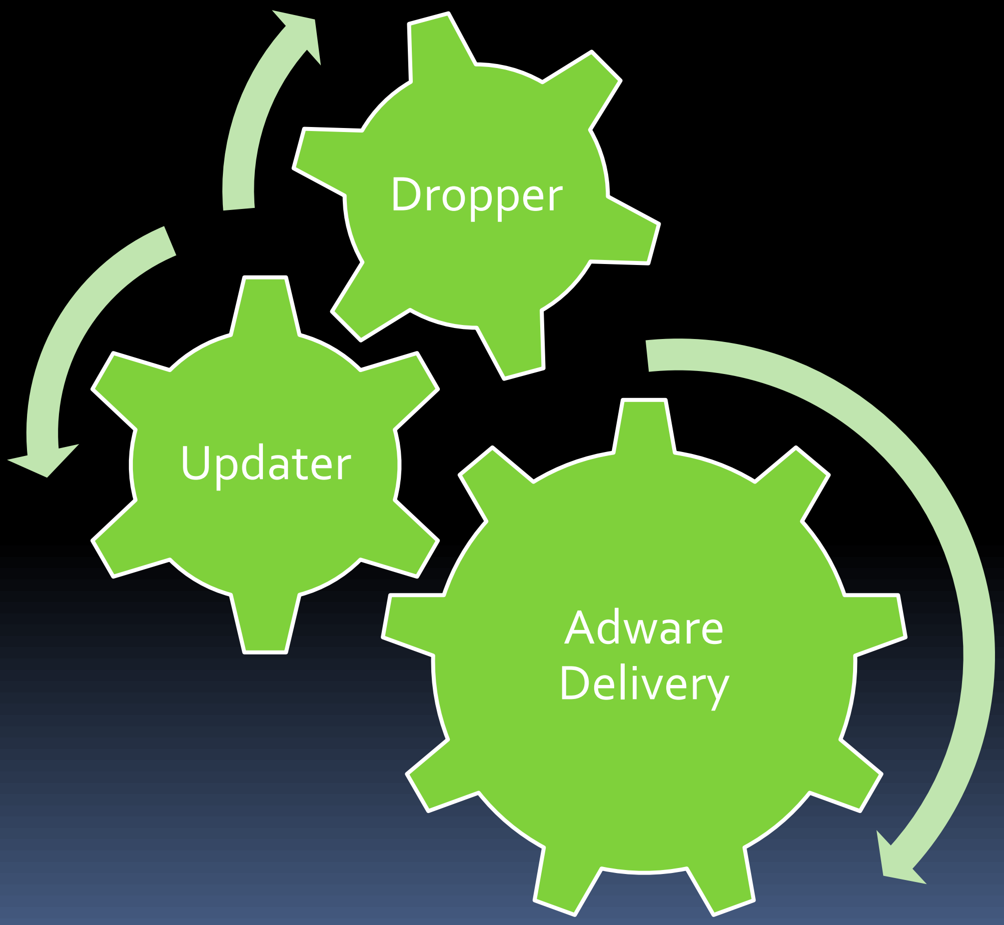
Hidden Code

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

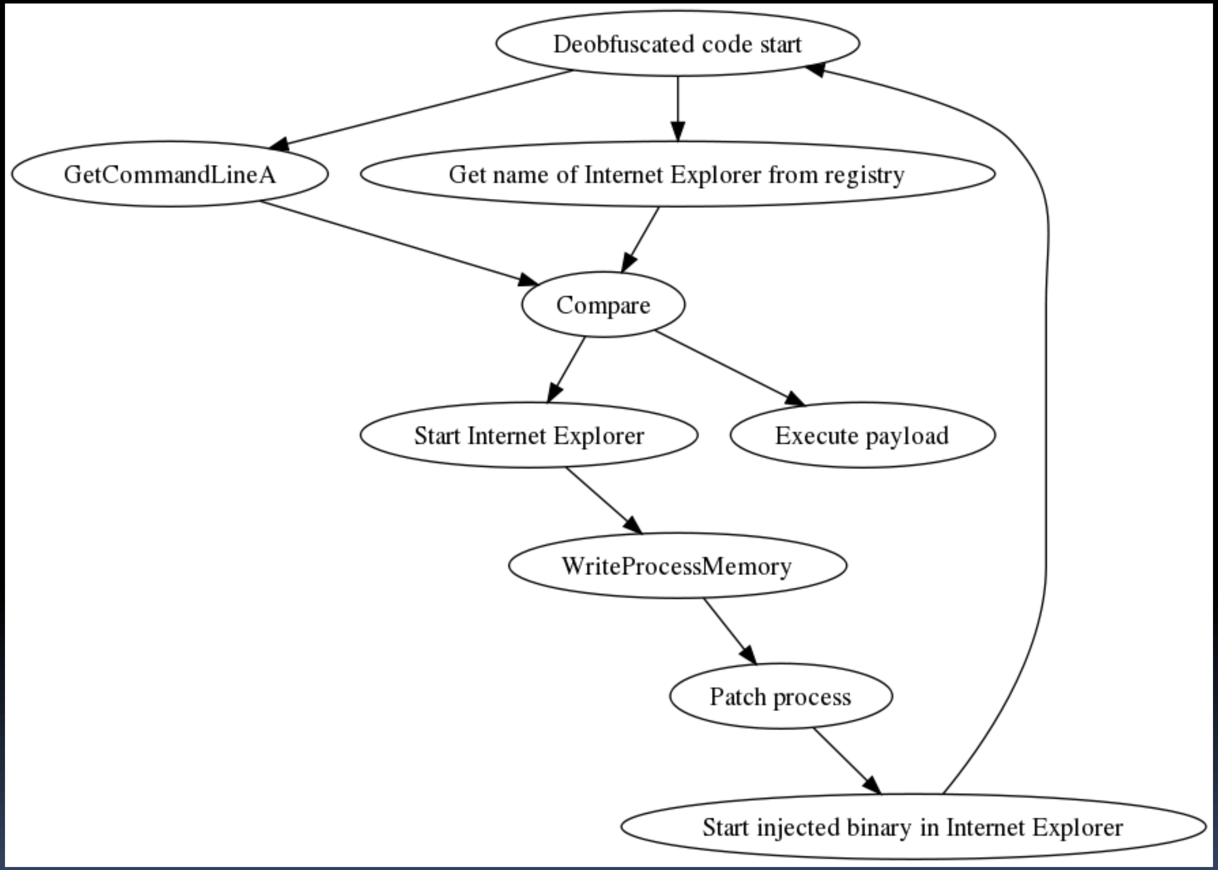
Typical Installation

1. 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



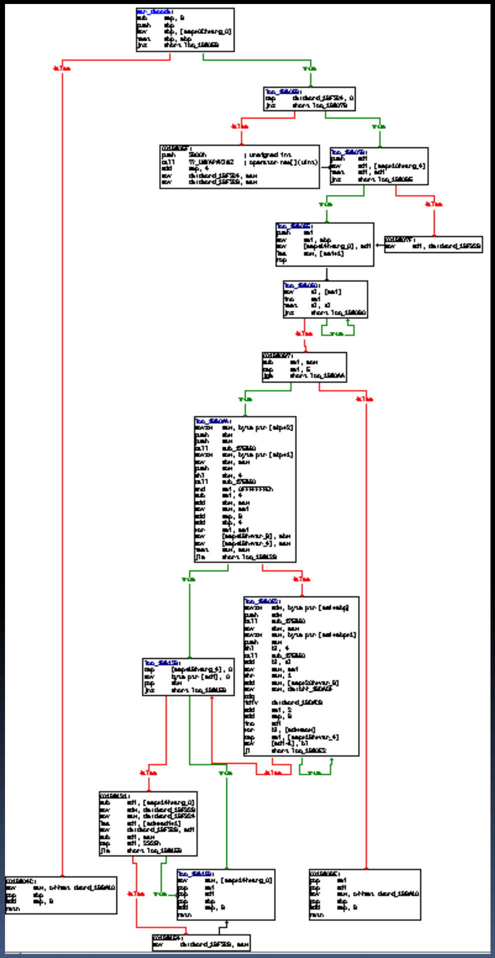
Code Injection



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
```

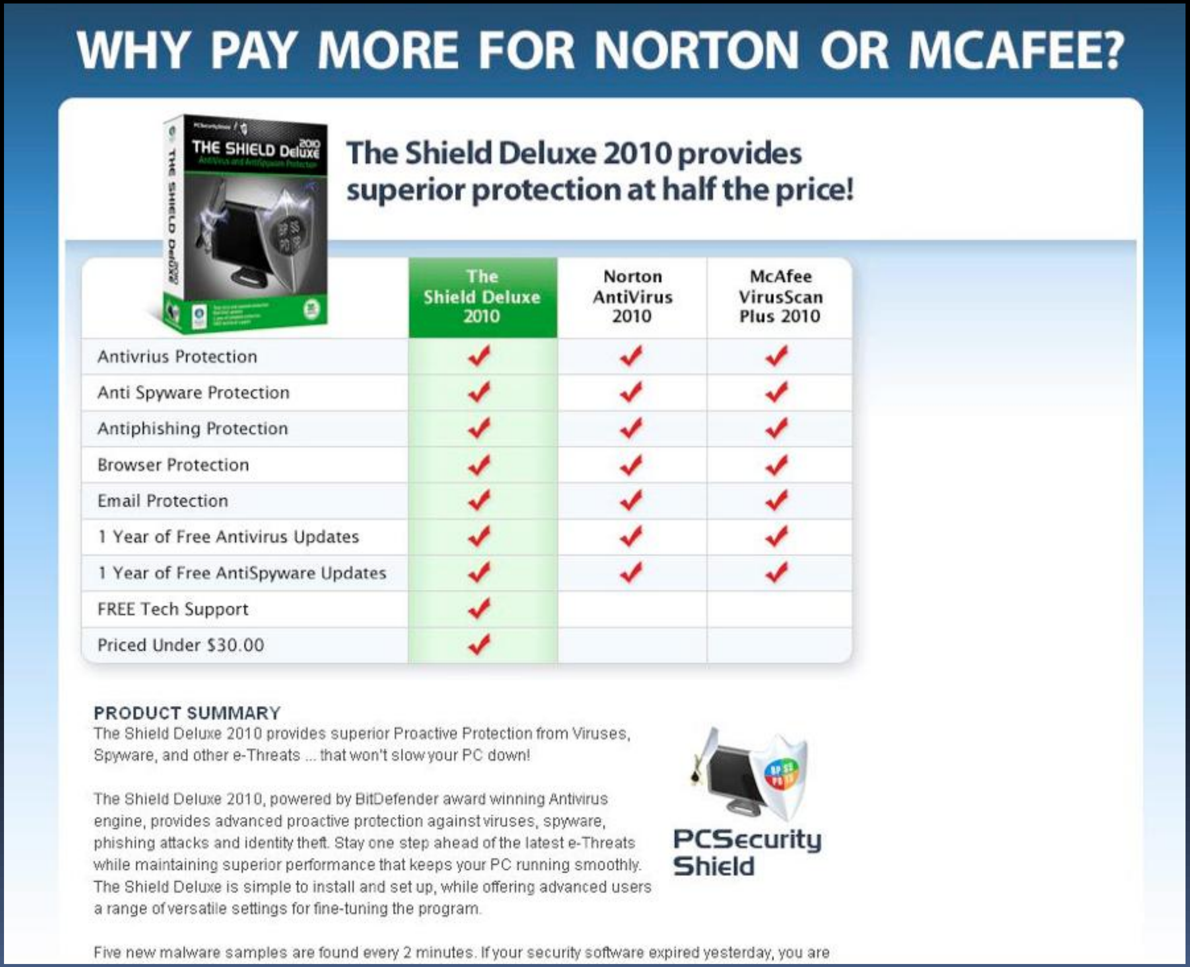
```
70000019:5.61msn:United States
```

- 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=292354428409850E819F8A4B73AA8FC3334E&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
```

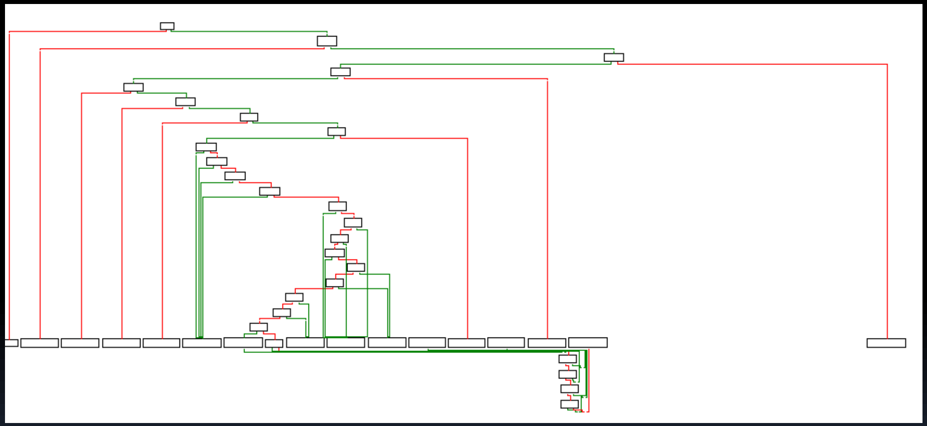
Advertisement



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

THANK YOU!