

# Windows malware reversing

Alexandre-Xavier Labonté-Lamoureux & Thomas Dupuy

# Why?

- Protection of customers
- Deep analysis about malware families
- Follow evolution of these families
- Collaboration with law enforcement

# How?

- Static
- Dynamic

# Tools

- IDA Pro
- JEB
- Ghidra
- Radare/Cutter
- Hopper
- Binary Ninja
- x64dbg
- GDB

# Type of malware

- financial (ransomware, banker...)
- espionage (keylogger, spyware...)
- disruption (DDos, wiper...)

# Vectors of compromise

- email
- website
- document
- vulnerability
- ...

# Identifying potential malicious functionalities

- persistence
- suspicious connections
- obfuscation (packer, control flow flattening...)

# x86 assembly

Instruction	Meaning	Instruction	Meaning
mov dst, src	$dst = src$ ( <i>copy</i> )	imul dst, src	$dst *= src$
push src	push src to stack	idiv dst, src	$dst /= src$
pop dst	pop from stack to dst	and dst, src	$dst \&= src$
lea dst, src	load src mem addr to dst	or dst, src	$dst  = src$
add dst, src	$dst += src$	xor dst, src	$dst ^= src$
sub dst, src	$dst -= src$	shl dst, src	$dst <= src$
inc dst	dst++	shr dst, src	$dst >= src$
dec dst	dst--	not dst	$dst = -dst$

# x86 assembly code flow

Instruction	Meaning	Instruction	Meaning
jmp addr	jump to addr	cmp a, b	comparison
je/jz addr	jump if ==	call func	function call
jne/jnz addr	jump if !=	nop	does nothing
ja/jnbe/jg/jnle addr	jump if >		
jae/jnb/jge/jnl addr	jump if >=		
jb/jnae/jl/jnge addr	jump if <		
jbe/jna/jle/jng addr	jump if <=		

# Windows API functions

- Filesystem
  - CreateFile
  - ReadFile, WriteFile
  - CreateFileMapping, MapViewOfFile
- Windows Registry
  - RegOpenKey
  - RegSetValue
  - RegGetValue
- Networking
  - WSAStartup
  - bind, listen
  - accept, connect
  - recv, send

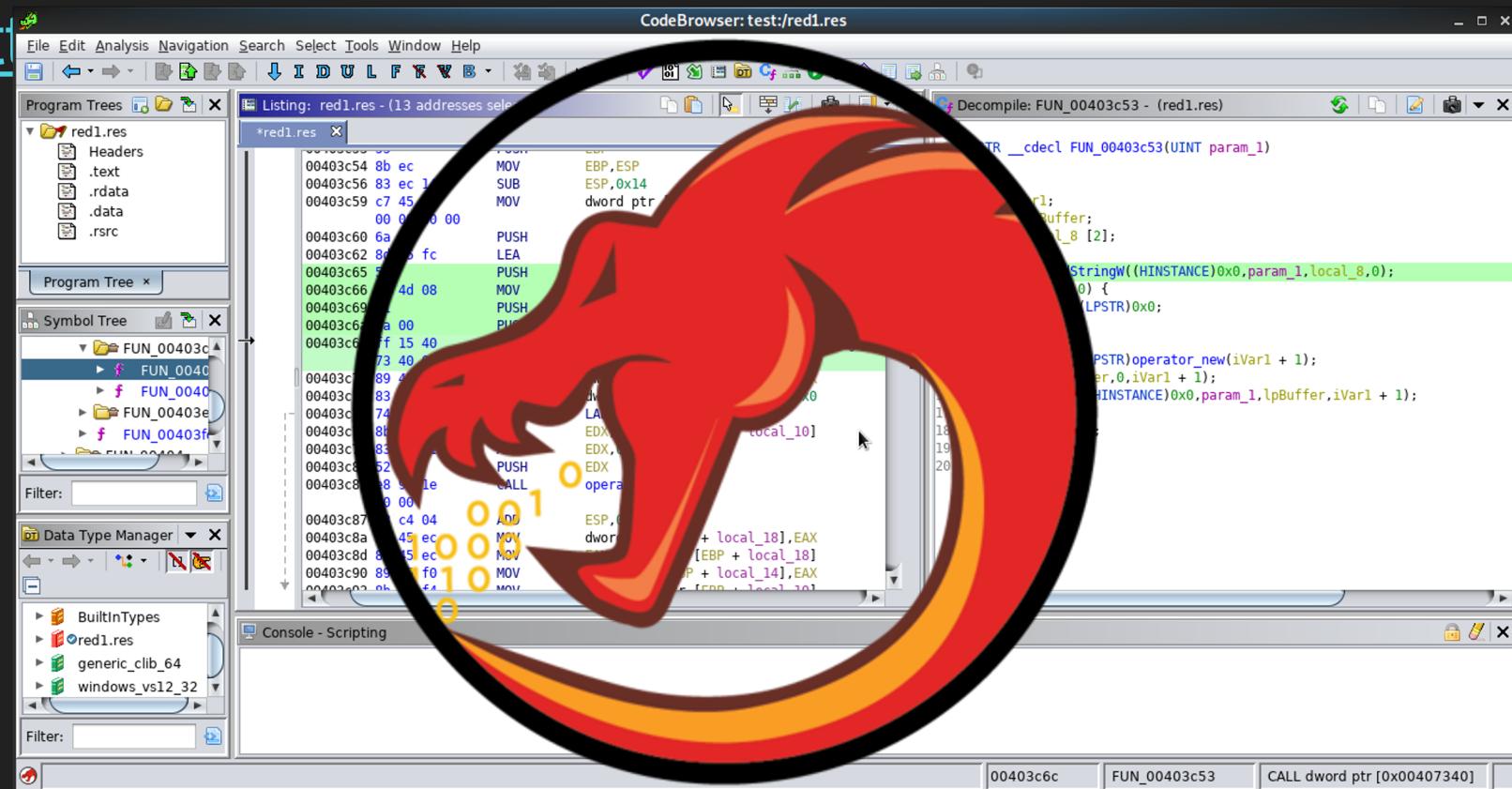
# Windows API functions

You can easily find what a Windows API function does by searching for it on MSDN

A screenshot of a Google search results page. The search query "msdn send" is entered in the search bar. Below the search bar, the "All" button is highlighted, followed by "Images", "News", "Videos", "Shopping", "More", "Settings", and "Tools". The search results indicate "About 7,820,000 results (0.46 seconds)". The top result is a link to the Microsoft Docs page for the "send" function. The page title is "send function (winsock2.h) - Win32 apps | Microsoft Docs". The URL is <https://docs.microsoft.com/en-us/windows/win32/api/nf-winsock2...>. The page content shows the function signature: "Dec 4, 2018 - The `send` function sends data on a connected socket. Syntax. int WSAAPI `send( SOCKET s, const char *buf, int len, int flags );` ...". Below the signature, there are links for "Parameters", "Return value", and "Remarks". A cursor arrow is visible on the left side of the page.

# Ghidra

# Download here:



# Working with Ghidra

How to start reverse engineering with Ghidra:

1. “File” menu → “New Project”
2. Select “Non-shared project”
3. Select a directory and project name
4. Your project was created. Now open the “File” menu and click “ImportFile”
5. Ghidra shows you info about the executable. Click “Ok” to continue.
6. Click on the executable’s name in the project folder. There will be a *cool* dragon animation.
7. Ghidra will ask if you want to analyze the file. Click “Yes”, then “Analyze”.

# Working with Ghidra

## Shortcuts

- “L”: Edit label, rename function or variable
- “;”: Add a comment
- “g”: Goto address
- “Ctrl+F”: Find
- “Ctrl+Z”: Undo
- “Ctrl+Shift+Z”: Redo
- “Alt+Left”: Go to previous location
- “Alt+Right”: Go to next location

Cheat sheet: <https://ghidra-sre.org/CheatSheet.html>

# Working with Ghidra

## Converting data

Note: You can only convert data via the disassembly listing

The screenshot shows the Ghidra interface with the Disassembly view open. A context menu is displayed over the assembly code at address 00101344. The menu includes options like 'Bookmark...', 'Clear Code Bytes' (with keyboard shortcut 'C'), 'Copy' (with 'Ctrl+C'), 'Comments', 'Instruction Info...', 'Patch Instruction' (with 'Ctrl+Shift+G'), 'Processor Manual...', 'Processor Options...', 'Create Function' (with 'F'), 'Create Thunk Function', 'Function', 'Compare Selected Functions...', 'Add Label...' (with 'L'), 'Show Label History...' (with 'H'), 'Clear Register Values...', 'Set Register Values...' (with 'Ctrl+R'), 'Colors', 'Convert' (selected), 'Set Equate...' (with 'E'), and 'Fallthrough'. The assembly code listed in the main window includes instructions like PUSH RBP, MOV RBP, RSP, SUB RSP, MOV dword ptr [RBP + local\_1], MOV byte ptr [RBP + local\_d], MOV RAX, MOV EDX, MOV qword ptr [RBP + local\_1], MOV qword ptr [RBP + local\_1], LEA RDX=>local\_68, MOV EAX, MOV ECX, MOV RDI, RDX, and MOV Char Sequence: "/tmp/.co".

# Working with Ghidra

Highlighting similar symbols

Click the middle mouse button

Note: You can change the button in: Edit -> ToolOptions -> ListingFields -> CursorTextHighlight



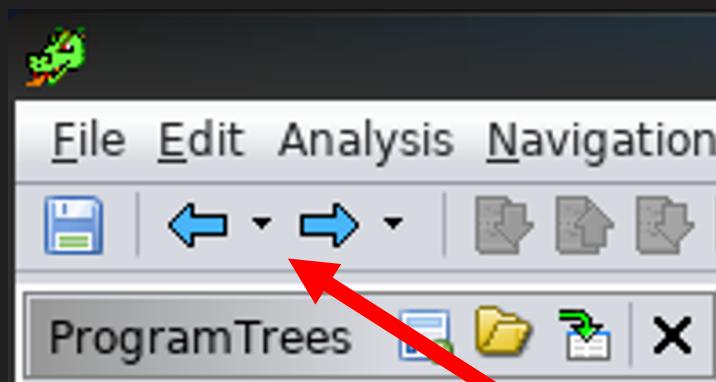
```
41 uVar3 = FUN_001013b4();
42 piVar4 = (int *) (ulong) uVar3;
43 piVar5 = local_28;
44 uVar7 = local_30;
45 if (uVar3 == 0) {
46     *(undefined *)piVar4 = *(undefined *)piVar4;
47     pcVar1 = (char *) ((long) piVar4 + -0x75);
48     *pcVar1 = *pcVar1 + (char) param_4;
49     ppuVar6 = &puStack64;
50     puStack64 = &stack0xfffffffffffff8;
51     if (param_4 != 1 && *pcVar1 != '\0') {
52         *(int *) ((long) piVar4 + -0x3f) = *(int *) ((long) piVar4 + -0x3f)
53         *piVar8 = (int) (in_ST0 * (float10) * (float *) ((long) piVar4 + -0x
54                                         /* WARNING: Bad instruction - Truncating control
55                                         halt_baddata();
56 }
57 piVar5 = piVar4;
58 uVar7 = (ulong) extraout_EDX;
59 }
```

# Working with Ghidra

## Navigation

Go to your previous location or  
your next location.

It's like undo and redo, but it's for  
the function that you were visiting.



# Working with Ghidra

Matching assembly code with decompiled code

Highlight the code in one panel so the matching code highlights in the other panel.

The screenshot shows the Ghidra interface with two main panels. The left panel, titled "Listing: redl.res - (13 addresses selected)", displays assembly code in three columns: address, opcodes, and comments. The right panel, titled "Decompile: FUN\_00403c53 - (redl.res)", shows the corresponding decompiled C-like code. A green highlight bar is positioned over the assembly code from address 00403c54 to 00403c6c, which corresponds to the highlighted code in the decompiled panel. This visual cue allows the user to quickly align the assembly and decompiled versions of the same code.

Address	Opcodes	Comments
00403c54	8b ec	MOV EBP,ESP
00403c56	83 ec 14	SUB ESP,0x14
00403c59	c7 45 f8	MOV dword ptr [EBP + local_c],0x0 00 00 00 00
00403c60	6a 00	PUSH 0x0
00403c62	8d 45 fc	LEA EAX=>local_8,[EBP + -0x4]
00403c65	50	PUSH EAX
00403c66	8b 4d 08	MOV ECX,dword ptr [EBP + param_1]
00403c69	51	PUSH ECX
00403c6a	6a 00	PUSH 0x0
00403c6c	ff 15 40 73 40 00	CALL dword ptr [->USER32.DLL::LoadStringW] LoadStringW((HINSTANCE)0x0,param_1,local_8,0)
00403c72	89 45 f4	MOV dword ptr [EBP + local_10],EAX
00403c75	83 7d f4 00	CMP dword ptr [EBP + local_10],0x0
00403c79	74 49	JZ LAB_00403cc4
00403c7b	8b 55 f4	MOV EDX,dword ptr [EBP + local_10]
00403c7e	83 c2 01	ADD EDX,0x1
00403c81	52	PUSH EDX

```
1 LPSTR __cdecl FUN_00403c53(UINT param_1)
2 {
3     int iVar1;
4     LPSTR lpBuffer;
5     WCHAR local_8 [2];
6
7     iVar1 = LoadStringW((HINSTANCE)0x0,param_1,local_8,0);
8     if (iVar1 == 0) {
9         lpBuffer = (LPSTR)0x0;
10    }
11    else {
12        lpBuffer = (LPSTR)operator_new(iVar1 + 1);
13        memset(lpBuffer,0,iVar1 + 1);
14        LoadStringA((HINSTANCE)0x0,param_1,lpBuffer,iVar1 + 1);
15    }
16    return lpBuffer;
17 }
18 }
```

# Scenario

A customer sends you an email saying that he received a weird email with an attachment. He already opened it, but nothing happened. He is wondering if it's malicious or not.

You can find the document compressed (password: montrehack) here:

Link: <http://fuckfuckgo.xyz/document.zip>

85b556bb944bdd0ae8de86f099ba71231cf734fb	document.zip
a57e52449d041121f502220cb0d3e403a9f7909f	facture.doc

# Questions

- What are the conditions for the malware to be executed?
- Can you see any mistakes from the operator?

Download the next stage... (please don't upload it on sharing platforms like VirusTotal, VirusShare, VirusBay...)

- What is the condition for the malware to be executed?
- What is the purpose of the function located at 0x402230?
- Starting at 0x401297, what does the malware do?
- How does the malware persists on the system?
- Explain what the malware does at 0x4015EB.

# Document

link: <http://fuckfuckgo.xyz/document.zip>

85b556bb944bdd0ae8de86f099ba71231cf734fb document.zip  
a57e52449d041121f502220cb0d3e403a9f7909f facture.doc

What are the conditions for the malware to be executed?

Can you see any mistakes from the operator?

Download the next stage... (please don't upload it on sharing platforms like VirusTotal, VirusShare, VirusBay...)

# v1

link: TODO

a3116dac8de52f7e614a6327402c3e88af0d09ba ph.zip  
ff9ba2fa2da1db170edad5223e98f21eecff9e07 v1

What is the condition for the malware to be executed?

What is the purpose of the function located at 0x402230?

Starting at 0x401297 what does the malware do?

How does the malware persists on the system?

Explain what the malware does at 0x4015EB.

# Bonus (v2)

link: TODO

Unpack the binary.

Write a script to automate the task.

# Bonus

TODO <link>

- Unpack the binary.
- Write a script to automatize the task.

# Links

- <https://www.welivesecurity.com/2019/08/08/varenyky-spambot-campaigns-france>
- <http://www.leparisien.fr/faits-divers/escroquerie-au-crypto-porno-un-pirate-doue-mais-instable-psychologiquement-14-12-2019-8217504.php>
- <https://krebsonsecurity.com/2019/12/nuclear-bot-author-arrested-in-sextortion-case/>