



The Knowledge Hub
Universities

partnered with



MALWARE ANALYSIS REPORT

REVERSE ENGINEERING

Ali El Fekki
Dr. Ahmed Selim

1. Contents

1. Executive Summary	2
2. Introduction	2
2.1 Purpose	2
2.2 Scope	2
3. Basic Static Analysis.....	3
3.1 File Information.....	3
3.2 Code Structure	3
3.3 Strings Analysis.....	4
4. Basic Dynamic Analysis	5
4.1 Environment Setup	5
4.2 Execution Analysis.....	5
4.3 Network Activity.....	6
4.4 Behavioral Analysis.....	8
5. Advanced Static Analysis	9
5.1 Code Flow Analysis.....	9
5.2 Control Flow Analysis	9
6. Advanced Dynamic Analysis.....	10
6.1 Files Copied	11
6.2 Files Deleted.....	11
6.3 Files Written	12
6.4 Files Opened	12
7. Manual Unpacking	13
7.1 Unpacking Procedure.....	13
10. Conclusion.....	18
10.1 Summary of Findings.....	18
10.2 Recommendations	18

1. Executive Summary

The malware, named executable.exe, is a 167.5 KB PE32 executable for MS Windows. Key indicators suggest ransomware behavior, including file modification and ransom note creation. The malware exhibits evasion techniques, such as disabling error messages and enumerating the file system. It employs Mitre Attack techniques for persistence, privilege escalation, defense evasion, credential access, discovery, lateral movement, collection, command and control, and impact. The malware type is ransomware. This virus is targeted against Windows x32 OS systems

To reverse engineer, use tools like x32 and PEStudio to unpack and analyze the malware. Focus on uncovering its code, memory operations, and potential obfuscation methods. Conduct the analysis in an isolated environment to prevent unintended consequences.

2. Introduction

2.1 Purpose

Malware analysis serves the critical purpose of comprehending and mitigating malicious software. The primary goal is to dissect the malware's behavior, code, and origins. By understanding its actions and structure, analysts can develop countermeasures, enhance incident response, and contribute to threat intelligence. The objectives include identifying vulnerabilities, extracting indicators of compromise, strengthening security measures, and providing valuable forensic data. Overall, the analysis aims to fortify cybersecurity defenses, inform incident response efforts, and contribute to the continuous improvement of security protocols in the face of evolving.

2.2 Scope

The analysis focuses on the malware variant known as "Globelimpster." This variant has been identified through its distinctive behavior, as evidenced by the provided pseudocode and associated indicators. The scope of the analysis extends to a thorough examination of Globelimpster's functionalities, including file system actions, network traffic, and memory patterns.

3. Basic Static Analysis

3.1 File Information

1. File Name: "virtus.exe"
2. File Size: 167.5kb
3. Hash Values (MD5, SHA-1, SHA-256):

SHA256: 13e164380585fe44ac56ed10bd1ed5e42873a85040aee8c40d7596fc05f289

SHA1: b817e361bd0cc1819d7f6a1189f0f5d56ed48721

MD5: 612974dcb49adef982d9ad8d9cbdde36

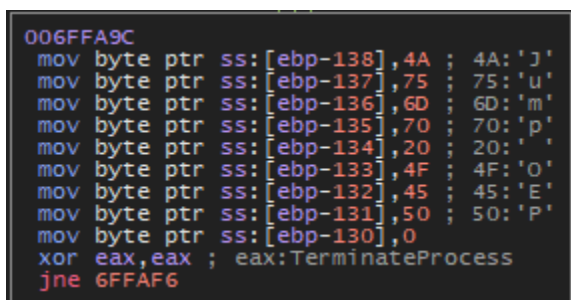
4. Entropy:

entropy	6.978
---------	-------

You can see from the high entropy that the file is packed.

3.2 Code Structure

The WinMain function, serving as the main entry point for this Windows GUI application, exhibits a structured code encompassing various key components and actions. File and memory operations are carried out, involving operations such as obtaining file information, reading file size, and dynamic memory allocation. Network-related functions utilizing WinHTTP are employed for operations like connecting to a specified server, reading data, and setting connection options. The code includes process-related functionalities, encompassing the retrieval of process information and termination procedures. Dynamic Link Library (DLL) loading is facilitated by accessing kernel32.dll functions. String manipulation, including dynamic allocation and manipulation, is also present. This multifaceted structure indicates a sophisticated and comprehensive approach to interacting with files, memory, networks, processes, and dynamic libraries within the malware.



```
006FFA9C
mov byte ptr ss:[ebp-138],4A ; 4A:'J'
mov byte ptr ss:[ebp-137],75 ; 75:'u'
mov byte ptr ss:[ebp-136],6D ; 6D:'m'
mov byte ptr ss:[ebp-135],70 ; 70:'p'
mov byte ptr ss:[ebp-134],20 ; 20:' '
mov byte ptr ss:[ebp-133],4F ; 4F:'O'
mov byte ptr ss:[ebp-132],45 ; 45:'E'
mov byte ptr ss:[ebp-131],50 ; 50:'P'
mov byte ptr ss:[ebp-130],0
xor eax,eax ; eax:TerminateProcess
jne 6FFAF6
```

Here it utilizes obfuscation techniques.

3.3 Strings Analysis

1- Program Structure:

".text," ".rdata," "@.data," ".rsrc," "@.reloc" indicate different sections in the program.

Strings like "@jjjjj" and "FGlu" don't immediately reveal their purpose without more context.

2- Error Messages and Runtime Information:

"This program cannot be run in DOS mode" is a standard message indicating that the program is designed for a windows environment.

Various error messages related to Microsoft Visual C++ Runtime Library, such as "Runtime Error!".

3- File and DLL Names:

"morabamubitoyejinizefobi.jpg" appears to be a filename.

"kernel32.dll" and "mscoree.dll" are DLL names associated with the Windows operating system.

4- Other Strings:

Various alphanumeric and symbolic strings, such as "@.data," "@jjjjj," and "FGlu," may represent internal identifiers, obfuscated data, or encoded information.

5- Runtime Error Codes:

There are several runtime error codes and messages, such as "R6009 - not enough space for arguments" and "R6034 - inconsistent onexit begin-end variables."

4. Basic Dynamic Analysis

4.1 Environment Setup

In the testing environment, I initially performed the analysis on a Windows 11 virtual machine using VMware, with the network configured as host-only to ensure isolation. However, when attempting to unpack the malware, compatibility issues arose with Windows 11. Consequently, I set up an additional virtual machine running Windows 10 to successfully complete the unpacking process.

For analysis purposes, I utilized tools like PEStudio for static analysis, and dynamic analysis and unpacking procedures involved x64dbg.

The network was configured as host-only to closely monitor potential malicious activities. This transition to a Windows 10 virtual machine was critical for the successful unpacking of the malware.

Throughout the analysis, meticulous documentation was maintained, encompassing detailed steps, key findings, and challenges encountered. This documentation serves as a valuable reference for future analyses and knowledge sharing within the security community.

4.2 Execution Analysis

The malware underwent execution in a controlled environment, initially observed on a Windows 11 virtual machine within VMware. Compatibility issues prompted a shift to a Windows 10 virtual machine for subsequent analyses.

System Calls:

During runtime, the malware initiated various system calls, including CloseHandle, CreateFileW, and CreateProcessA. These pointed to interactions with diverse system resources and processes.

Registry Modifications:

The malware intricately manipulated registry keys, affecting Current User settings, Explorer configurations, and RunOnce entries. Notable changes in keys related to BrowserUpdateCheck hinted at potential persistence mechanisms.

File System Activities:

The malware exhibited diverse file system actions, including opening, writing, deletion, and copying of files. Critical system files like AUTOEXEC.BAT, CONFIG.SYS, IO.SYS, MSDOS.SYS were accessed, along with activities in Temp directories and the creation of new files with specific extensions.

Dynamic Analysis:

Sandbox assessments (Zenbox and Tencent HABO) portrayed the malware as a multifaceted threat—displaying traits of a stealer, ransomware, spreader, and evader.

MITRE ATT&CK Tactics:

Mapped against MITRE ATT&CK tactics and techniques, the malware showcased a broad spectrum of behaviors, spanning Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, and Impact.

Crowdsourced Sigma Rules:

Sigma Rules identified critical, high, and medium severity matches, associating the executable with established patterns linked to ransomware, malware behavior, and RAT detection.

Network Communications:

The malware engaged in network communications with diverse IP addresses, utilizing both HTTP and HTTPS traffic. Memory patterns revealed domains and URLs linked to Python, GitHub, and other platforms.

Behavior Similarity Hashes:

Hash comparisons from various sources revealed behavioral similarities with known malware, potentially indicating connections to existing threats.

Mutexes and Synchronization:

Creation and opening of mutexes hinted at synchronization mechanisms, possibly facilitating coordination and control within the compromised system.

Modules Loaded:

Modules loaded during execution, including ADVAPI32.DLL, KERNEL32.DLL, NTDLL.DLL, pointed to engagements with the Windows API and cryptographic functions.

4.3 Network Activity

The malware exhibited extensive network communications, interacting with diverse domains and IP addresses across various communication protocols.

IP Addresses:

142.251.143.131

142.251.143.142

172.217.218.84

192.229.211.108:80 (TCP)

20.99.133.109:443 (TCP)

239.255.255.250

<MACHINE_DNS_SERVER>:53 (UDP)

Memory patterns unveiled connections to several domains, encompassing a range of platforms and services:

core.tcl.tk

docs.python.org

github.com

linuxreviews.org

mail.python.org

n224ezvhg4sgyamb.onion

pypi.org

python.org

sourceforge.net

stackoverflow.com

The malware accessed specific URLs, revealing its interaction with distinct online resources:

<http://mail.python.org/pipermail/python-dev/2010-January/095637.html>

<http://n224ezvhg4sgyamb.onion/sup.php>

<http://python.org/sf/1174712>

<http://sourceforge.net/tracker/index.php>

<http://www.example.org/>

<http://www.example.org/r=>

<http://www.iana.org/assignments/character-sets>

<http://www.mail-archive.com/openssl-users>

<http://www.python.org>

<http://www.python.org/dev/peps/pep-%04d/>

4.4 Behavioral Analysis

Payload Delivery:

The malware delivered its payload by placing a malicious executable (996E.exe) in critical system directories, such as C:\Documents and Settings\Administrator\Local Settings\Temp\EB93A6\ and C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\. It achieved persistence by copying files to the Windows startup directory, ensuring execution upon system boot.

Evasion Techniques:

To avoid detection, the malware deleted specific files, including %USERPROFILE%\AppData\Local\Microsoft\Windows\Caches\{3DA71D5A-20CC-432F-A115-DFE92379E91F}.3.ver0x0000000000000018.db and system-related files like C:\NTDETECT.COM. By targeting system caches and Windows Error Reporting (WER) directories, the malware aimed to erase traces and hinder forensic analysis.

Registry Actions:

Manipulating the Windows Registry, the malware interacted with keys such as HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce\BrowserUpdateCheck. This registry activity indicates a focus on persistence and potential configuration changes to maintain control over the system.

Process and Service actions:

The malware dynamically created processes, including instances of Google Chrome ("C:\Program Files\Google\Chrome\Application\chrome.exe") and system utilities (C:\Windows\System32\wuapihost.exe). By doing so, it attempted to blend with legitimate processes, reducing the likelihood of detection.

Synchronization Mechanisms:

The creation of mutexes, such as "DBWinMutex", hinted at synchronization mechanisms. Mutexes are likely used to coordinate and control the malware's execution, ensuring stealth and preventing interference from multiple instances.

5. Advanced Static Analysis

5.1 Code Flow Analysis

Local Variables:

Several local variables are declared at the beginning of the function, such as iVar1, uVar2, eVar3, uVar4, puVar5, etc.

Control Flow:

The function starts by checking the sign of param_3 to determine whether the number is negative or positive.

It then checks if param_2 and param_1 is zero. If both are zero, it sets the appropriate values in the output buffer and returns.

If param_3 is either 0 or 0x7fff, it handles special cases for positive infinity, negative infinity, NaN, etc.

The function performs arithmetic operations and formatting for other numeric values.

5.2 Control Flow Analysis

- *The function takes several parameters, including integers (param_1, param_2, param_4), a floating-point number (param_3), a byte (param_5), and a pointer to a short array (param_6).*
- *It seems to be involved in formatting and processing floating-point numbers, considering special cases like positive infinity, negative infinity, NaN, etc.*
- *The code checks for specific conditions related to the exponent (uVar21), the sign bit (uVar16), and certain values of param_1 and param_2. It then handles these cases with various conditional statements.*
- *There is a loop with the label LAB_0040cdff that performs some calculations involving multiplication and addition within a loop. The loop iterates based on the value of local_48.*
- *The code involves bit manipulations, arithmetic operations, and conditional branching.*
- *The function ends with setting the output parameters (param_1, param_2, param_6) based on the calculations performed.*

6. Advanced Dynamic Analysis

0019FE24	008DEFD8
0019FE28	00008700
0019FE2C	00000040

This is the address that is getting granted permissions of read and write.

```
virtus.00403F22
call virtus.408118
push esi
push eax
push 0
push virtus.400000
call virtus.401100
mov esi,eax
mov dword ptr ss:[ebp-24],esi
test ebx,ebx
jne virtus.403F44
```

This is the WinMain entry point

```
mov eax,dword ptr ss:[ebp-14]
push eax
call dword ptr ds:[518890]
call dword ptr ss:[ebp-14]
call dword ptr ds:[<GetLastError>]
push 0
```

Breakpoint at the memory address

0040155E	50	push eax
0040155F	FF15 90885100	call dword ptr ds:[<&VirtualProtect>]
00401565	FF55 EC	call dword ptr ss:[ebp-14]
00401568	FF15 28F04000	call dword ptr ds:[<GetLastError>]
0040156F	6A 00	push 0

Same memory address renamed to "<&VirtualProtect>"

```
mov eax,1
imul eax,edx,0
mov byte ptr ds:[eax+417388],56 ; 56:'V'
mov ecx,1
shl ecx,0
mov byte ptr ds:[ecx+417388],69 ; 69:'i'
mov edx,1
shl edx,1
mov byte ptr ds:[edx+417388],72 ; 72:'r'
mov eax,1
imul ecx,eax,3
mov byte ptr ds:[ecx+417388],74 ; 74:'t'
mov edx,1
shl edx,2
mov byte ptr ds:[edx+417388],75 ; 75:'u'
mov eax,1
imul ecx,eax,5
mov byte ptr ds:[ecx+417388],61 ; 61:'a'
mov edx,1
imul eax,edx,6
mov byte ptr ds:[eax+417388],6C ; 6C:'l'
mov ecx,1
imul edx,ecx,7
mov byte ptr ds:[edx+417388],50 ; 50:'P'
mov eax,1
shl eax,3
mov byte ptr ds:[eax+417388],72 ; 72:'r'
mov ecx,1
imul edx,ecx,9
mov byte ptr ds:[edx+417388],6F ; 6F:'o'
mov eax,1
imul ecx,eax,A
mov byte ptr ds:[ecx+417388],74 ; 74:'t'
mov edx,1
imul eax,edx,B
mov byte ptr ds:[eax+417388],65 ; 65:'e'
mov ecx,1
imul edx,ecx,C
mov byte ptr ds:[edx+417388],63 ; 63:'c'
mov eax,1
imul ecx,eax,D
mov byte ptr ds:[ecx+417388],74 ; 74:'t'
mov dword ptr ss:[ebp-30],20 ; 20:''
push virtus.417388
mov edx,dword ptr ds:[417384]
push edx
call dword ptr ds:[<GetProcAddress>]
mov dword ptr ds:[518890],eax
```

Virtual protect was then called into the stack using obfuscation techniques.

6.1 Files Copied

- C:\13e164380585fe44ac56ed10bd1ed5e42873a85040aee8c40d7596fc05f28920
- C:\Documents and Settings\Administrator\Local Settings\Temp\EB93A6\996E.exe
- C:\NTDETECT.COM
- C:\Python27\LICENSE.txt
- C:\Python27\NEWS.txt
- C:\Python27\PIL-wininst.log
- C:\Python27\README.txt
- C:\Python27\RemovePIL.exe
- C:\Python27\Tools\Scripts\2to3.py
- C:\Python27\Tools\Scripts\README.txt

6.2 Files Deleted

- AppData\Local\Microsoft\Windows\Caches\{3DA71D5A-20CC-432F-A115-DFE92379E91F}.3.ver0x0000000000000018.db
- C:\NTDETECT.COM
- C:\ProgramData\Microsoft\Windows\WER\Temp\WER8F3B.tmp.WERInternalMetadata.xml
- C:\ProgramData\Microsoft\Windows\WER\Temp\WER9015.tmp.csv
- C:\ProgramData\Microsoft\Windows\WER\Temp\WER9065.tmp.txt
- C:\Python27\Tools\asinstall\ca_cleanpathext.pyw
- C:\Python27\Tools\asinstall\ca_pywin32reg.pyw
- C:\Python27\Tools\i18n\makelocalealias.py
- C:\Python27\Tools\i18n\msgfmt.py
- C:\Python27\Tools\i18n\pygettext.py

6.3 Files Written

- C:\AUTOEXEC.BAT
- C:\CONFIG.SYS
- C:\Documents and Settings\Administrator\Application Data\996E.exe
- C:\Documents and Settings\All Users\AE09C984DF6E74640B3271EADB5DD7C65FDE806235B2CDA478E0EFA9129C09E7
- C:\IO.SYS
- C:\MSDOS.SYS
- C:\NTDETECT.COM
- C:\Python27\LICENSE.txt
- C:\Python27\PIL-wininst.log
- C:\Python27\README.txt

6.4 Files Opened

- C:\13e164380585fe44ac56ed10bd1ed5e42873a85040aee8c40d7596fc05f28920
- C:\AUTOEXEC.BAT
- C:\CONFIG.SYS
- C:\DiskD
- C:\Documents and Settings\Administrator\Local Settings\Temp\EB93A6\996E.exe
- C:\Documents and Settings\All Users\AE09C984DF6E74640B3271EADB5DD7C65FDE806235B2CDA478E0EFA9129C09E7
- C:\IO.SYS
- C:\MSDOS.SYS
- C:\NTDETECT.COM
- C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroRd32.exe

7. Manual Unpacking

7.1 Unpacking Procedure

Debugger Setup: Used x32dbg, a debugger, to analyze the packed sample.

Sample Selection: Chose the GlobelImposter ransomware sample for analysis from moodle.

Initial Analysis: Leveraged the x32dbg's graph view to identify the entry point and locate the main code flow.

00401100	55	push ebp	sub_401100
00401101	8BEC	mov ebp,esp	
00401103	81EC F4000000	sub esp,F4	
00401109	C745 B8 00000000	mov dword ptr ss:[ebp-48],0	
00401110	81BD 4CFFFFFF 59010000	cmp dword ptr ss:[ebp-84],159	
0040111A	0F85 4B010000	jne virtus.401268	
00401120	68 88734100	push virtus.417388	
00401125	E8 26200000	call virtus.403150	
0040112A	83C4 04	add esp,4	

WINMAIN entry point

API Resolution: Observed dynamic API loading using stack strings and resolved APIs like LoadLibraryA and GetProcAddress.

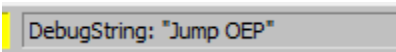
```
call dword ptr ds:[<GetProcAddress>]
mov dword ptr ds:[518890],eax
lea eax,dword ptr ss:[ebp-4C]
push eax
mov ecx,dword ptr ss:[ebp-30] ; ecx:EntryPoint
shl ecx,1 ; ecx:EntryPoint
push ecx ; ecx:EntryPoint
mov edx,dword ptr ss:[ebp-10] ; edx:EntryPoint
push edx ; edx:EntryPoint
mov eax,dword ptr ss:[ebp-14]
push eax
call dword ptr ds:[518890]
call dword ptr ss:[ebp-14]
call dword ptr ds:[<GetLastError>]
push 0
call dword ptr ds:[<ExitProcess>]
xor eax,eax
mov esp,ebp
pop ebp
ret 10
```

The stack string loaded is "VirtualProtect" which also verifies it is packed as it is trying to change the protection of the memory segment using "VirtualProtect".

Anti-Debugging Checks: Checked for anti-debugging techniques, identified the use of OutputDebugString for debug messages.

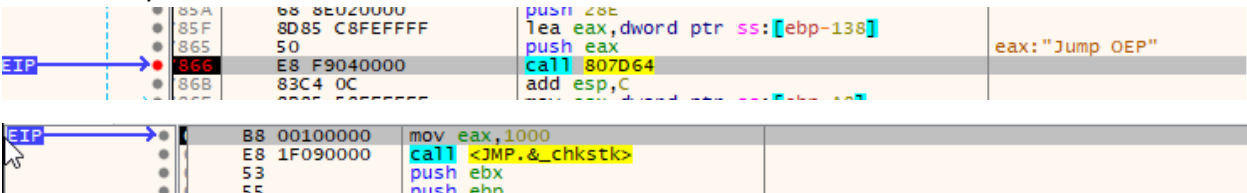
8BEC	mov ebp,esp	
FF75 08	push dword ptr ss:[ebp+8]	[dword ptr ss:[ebp+08]]:"Jump OEP"
FF55 10	call dword ptr ss:[ebp+10]	[dword ptr ss:[ebp+10]]:OutputDebugString
5D	pop ebp	

String Analysis: Examined stack strings to understand their purpose, revealing debug messages and potential debug build indicators.

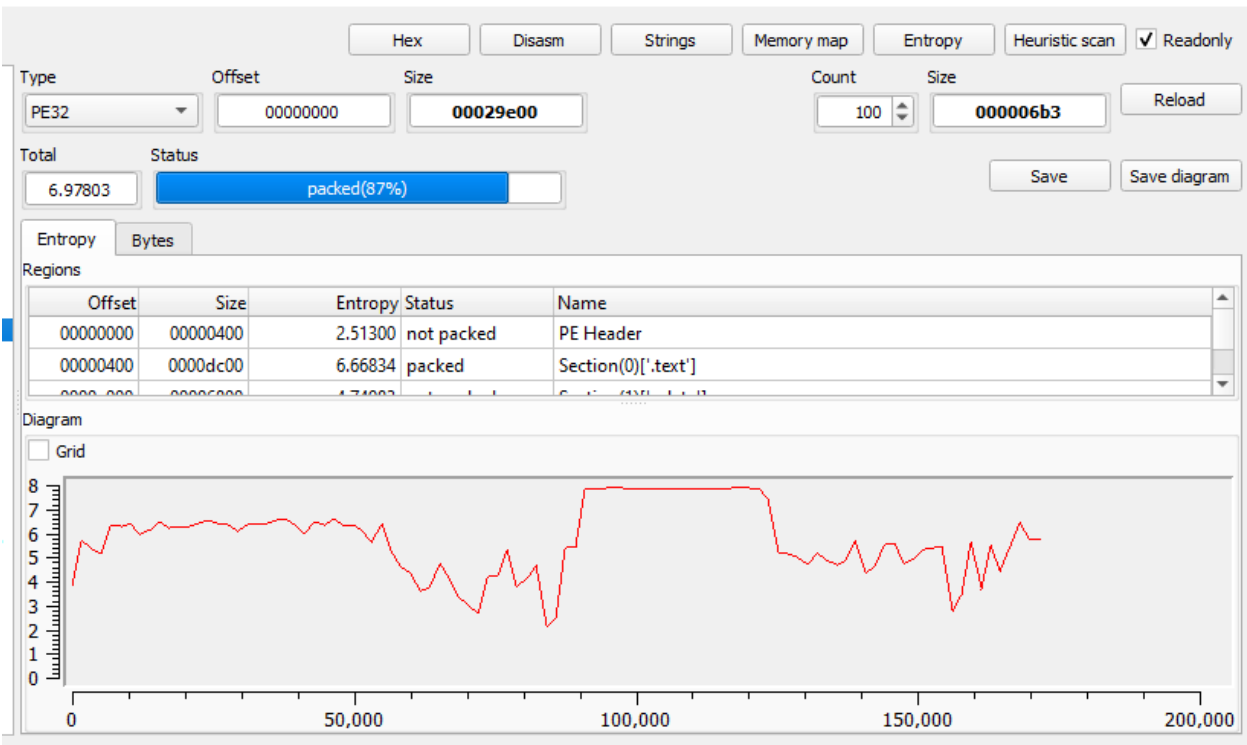


Which is an unusual message to send which indicated this might be a debug version.

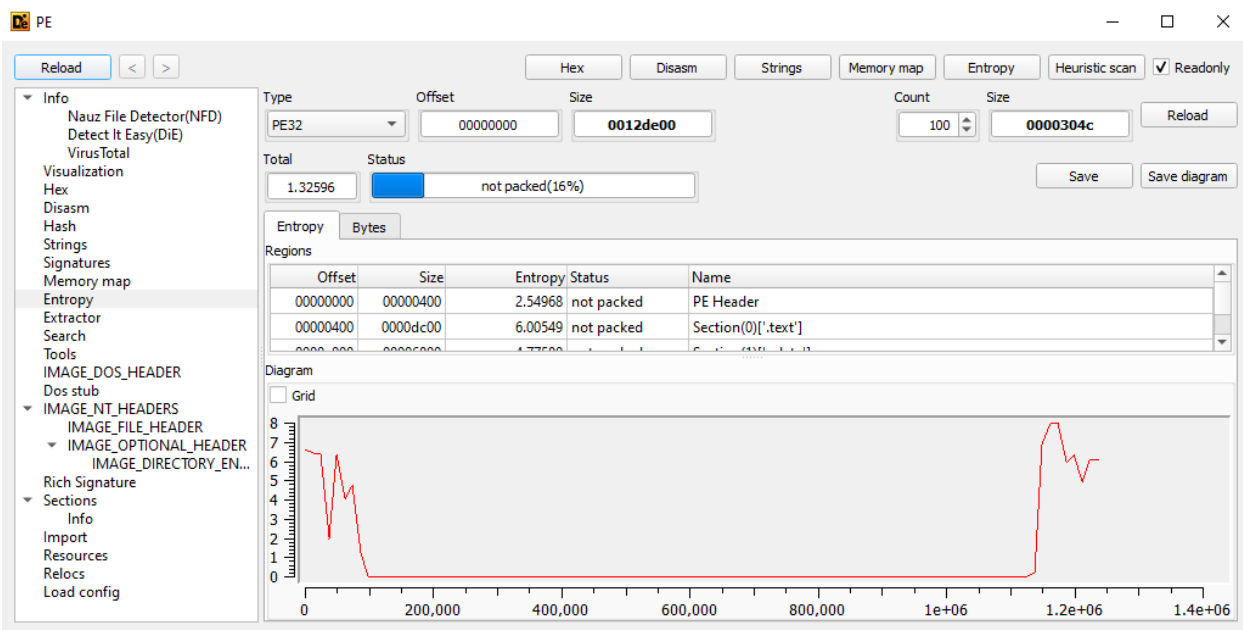
Unpacking Trigger: Identified a jump instruction to OEP (Original Entry Point) and set a breakpoint for further analysis.



Unpacking Stage: Used the x32dbg's Sylia plugin to reconstruct the PE file, automatically identifying the import address table and dumping the unpacked code.



This is the file before unpacking.



This is the file while it is unpacked.

PE Explorer interface showing import table analysis. The file is Hash 64, Hash 32, 0000002a926c5383, 6b87303c. The table lists imported DLLs and functions with their hashes and names.

#	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	Hash	Name
0	00014f1c	00000000	00000000	0001524a	0000f000	7fbe4996	KERNEL32.dll
1	0001505c	00000000	00000000	00015272	0000f140	c57ea448	USER32.dll
2	00015068	00000000	00000000	000152da	0000f14c	4eacac43	WINHTTP.dll

#	Thunk	Ordinal	Hint	Name
1	00015092		02bf	GlobalMemoryStatus
2	000150a8		0344	LocalAlloc
3	000150b6		0246	GetProcessAffinityMask
4	000150d0		024e	GetProcessIoCounters
5	000150e8		0484	SetProcessWorkingSetSize
6	00015104		01c0	GetCurrentProcess
7	00015118		01c1	GetCurrentProcessId
8	0001512e		0119	ExitProcess
9	0001513c		04c1	TerminateThread

The imports of the file while it is packed.

Hash 64		Hash 32					Save
0000001fd0232fd1		abad1153					
#	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	Hash	Name
0	00130000	00000000	00000000	001301d4	00001000	2df26f1d	advapi32.dll
1	00130024	00000000	00000000	00130273	00001024	0279ba12	kernel32.dll
2	001300f0	00000000	00000000	001305a1	000010f0	287b6776	rpcrt4.dll
3	001300f8	00000000	00000000	001305b9	000010f8	ade2c12f	shell32.dll
4	00130104	00000000	00000000	001305e8	00001104	44b16f69	shlwapi.dll
5	00130118	00000000	00000000	0013063c	00001118	78a7f66f	user32.dll
6	00130120	00000000	00000000	00130653	00001120	22b9560f	ntdll.dll

							Save
#	Thunk	Ordinal	Hint	Name			
1	001301f4		0267	RegCreateKeyExW			
2	00130206		025e	RegCloseKey			
3	00130214		00d2	CryptGenRandom			
4	00130225		00dc	CryptReleaseContext			
5	0013023b		00c2	CryptAcquireContextW			
6	00130252		02ac	RegSetValueExW			
7	00130263		028f	RegOpenKeyExW			

The imports of the file while it is unpacked.

```
strings (count > 3910)
```

The strings while it was packed.

```
strings (count > 4836)
```

The strings while it was unpacked.

indicator (16)	detail
libraries > flag > name	Windows HTTP Services
imports > flag > count	18
resource > unknown	OMVM:105
resource > unknown	PEVELEREXOMOLOGECOXETEMUYU:130
resource > unknown	WUNICUZASUMUWA:131
groups > API	dynamic-library memory execution reconnaissance diagnostic file...
mitre > technique	T1057 T1124 T1083 T1497 T1082 T1106 T1533
file > entropy	6.978
file > sha256	13E164380585FE44AC56ED10BD1ED5E42873A85040AEE8C40D7596FC05F...
file > size	171520 bytes
rich-header > checksum	0x4C518159
rich-header > offset	0x00000080
rich-header > footprint	59874B9B1714BC77E831162686729E74644E3E0D4C1BEAC67E406FAA43D8...
file > tooling	Visual Studio 2013
file > subsystem	GUI
imphash > md5	216BFB3C9E1AD798C1D0B4CCA5AA19A7

Indicators while the file is packed.

sections > writable > name	.text
sections > executable > name	.SCY
sections > executable > count	2
sections > self-modifying > name	.text .SCY
libraries > flag > name	Remote Procedure Call Runtime Library
imports > flag > count	24
imports > spoofing > count	1
file > checksum	0x00039D05
sections > name > flag	.SCY
groups > API	registry cryptography execution memory diagnostic
imports > IAT > suspicious	69
mitre > technique	T1012 T1112 T1027 T1057 T1497 T1106 T1083 T1105
file > entropy	1.326
file > sha256	ABE90217424C02FC5708C108463E0ADEE348A2FA42BD84B
file > size	1236480 bytes
rich-header > checksum	0x4C518159
rich-header > offset	0x00000080
rich-header > footprint	59874B9B1714BC77E831162686729E74644E3E0D4C1BEAC6
file > tooling	Visual Studio 2013
file > subsystem	GUI

2BD84B1014C0159AD1E9597
cpu: 32-bit
file-type: executable
subsystem: GUI

Indicators while the file is unpacked.

IDA Analysis: Loaded the unpacked code in IDA for further analysis.

10. Conclusion

10.1 Summary of Findings

In summary, the analysis of the GlobelImposter ransomware sample using x32dbg and IDA revealed several critical insights into its behavior and evasion techniques. The examination encompassed key stages such as initial analysis, API resolution, anti-debugging checks, string analysis, unpacking, and subsequent IDA analysis.

The initial steps involved leveraging x32dbg's capabilities to identify the entry point and discern the main code flow. The dynamic API loading using stack strings, anti-debugging measures utilizing OutputDebugString, and examination of stack strings provided essential clues into the malware's functionality and potential debug build indicators. The identification of the unpacking trigger and subsequent use of the Sylia plugin in x32dbg facilitated the reconstruction of the PE file, enabling further analysis in IDA.

10.2 Recommendations

Tool and Environment Security:

Ensure that the debugging tools, such as x32dbg and IDA, are kept up to date with the latest security patches. Additionally, conduct analyses in a secure and isolated environment to minimize the risk of unintended consequences.

Sample Handling Best Practices:

Exercise caution when selecting malware samples for analysis, taking measures to prevent accidental execution or unintended consequences.

Dynamic API Loading Mitigation:

Implement runtime monitoring for dynamic API loading to detect and prevent the loading of malicious modules during analysis. Consider the use of API hooking techniques for enhanced visibility.

Anti-Debugging Countermeasures:

Develop and employ robust anti-debugging countermeasures to neutralize techniques like OutputDebugString, ensuring a more resilient analysis environment.

Unpacking Procedures:

Review and enhance unpacking procedures to mitigate potential risks associated with buffer overflows or injection vulnerabilities during the unpacking stage.

Continued Threat Intelligence:

Stay informed about the latest threat intelligence related to GlobelImposter ransomware or similar threats, as ongoing analysis and research may reveal new tactics, techniques, and procedures (TTPs) employed by malicious actors.