

Challenge: MalaCrypt Lab

Platform: CyberDefenders

Category: Malware Analysis

Difficulty: Medium

Tools Used: PEStudio, Floss, Strings, CyberChef, ProcMon, Cutter, Capa, VirusTotal

Summary: This lab involved investigating a malware sample and focused on practising basic to intermediate static and dynamic analysis of a PE file. It required collecting basic information about the sample, including its architecture, original filename, suspicious libraries, and strings. The dynamic analysis component involved executing the sample and using tools such as ProcMon to observe malware behaviour, including registry activity and image load events. Basic reverse engineering was also performed on the sample through the use of a disassembler; in this case, a combination of Capa and Cutter was used to effectively analyse suspicious code within the file. I recommend giving this lab a shot if you are relatively new to learning malware analysis and wish to practice.

Scenario: During routine monitoring at MalaCrypt company, a suspicious binary named "malware.exe" was found on a device. Initial checks of its hash values against threat intelligence platforms yielded no results, suggesting the attacker may have altered the malware to evade detection. As a security analyst, the next action is to investigate further, using alternative methods beyond hash-based detection, considering that attackers often modify hashes to bypass initial security checks.

Understanding the type of binary architecture allows us to determine the types of registers being used. What architecture is this binary built for?

Each CPU has its own instruction set architecture (ISA) which is the language a CPU understands. To find the architecture of this binary, we can use a static-analysis tool called PEStudio:

property	value
file	wait...
file > sha256	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
file > first-bytes-hex	M Z .. .@ .. .
file > first-bytes-text	1079128 bytes
size	7.201
entropy	executable
file > type	cpu
cpu	64-bit
subsystem	GUI
version	1.0.0.144
description	ZEON Gaiilo Doc Application
entry-point > first-bytes-hex	48 83 EC 28 E8 CB 05 00 00 48 83 C4 28 E9 72 FE FF CC CC 48 83 EC 28 4D 8B 41 38 48 8B CA 49 8B
entry-point > location	0x000069E0 (section[,text])
signature tooling	wait...

We can see that this binary is for 64-bit CPUs, i.e., CPUs which support a x64 architecture.

Answer: x64

Executables are sometimes renamed or altered to evade detection or disguise their true purpose. What is the original name of the executable?

Within Portable Executable (PE) files is a specific field in the file's version information resource called the original filename. As the name implies, this stores the file's name as it was at compile time. Using PEStudio, we can see that the original filename was "DefaultViewer.exe":

version > original-file-name DefaultViewer.exe

Answer: DefaultViewer.exe

Some DLL files are responsible for accessing Windows registries. Which DLL file is utilized to manipulate the Windows Registry?

One incredible feature of PEStudio is its ability to flag suspicious imports and strings. If you navigate to the imports section and look at the flagged imports, we can see references to RegSetValueExA and RegDeleteValueA:

The screenshot shows the PEStudio interface with the following details:

- File Menu:** file settings about
- Toolbar:** File, Settings, About, Open, Save, Exit, Help
- Left Panel (File Explorer):** Shows the file structure of the executable, including sections like indicators (imports > flag), footprints, virustotal, dos-header, dos-stub, rich-header, file-header, optional-header, directories, sections, libraries, imports (flag > 115), and exports (n/a).
- Right Panel (Imports View):** A table showing flagged imports:

imports (115)	flag (25)
680 (IsUserAnAdmin)	x
RegSetValueExA	x
RegDeleteValueA	x
GetCurrentProcessId	x
GetEnvironmentVariableA	x
VirtualAlloc	x
WriteFile	x
DeleteFileA	x
FindNextFileA	x
FindFirstFileA	x
GetCurrentProcess	x
GetCurrentThreadId	x

Upon viewing the Microsoft documentation, we can determine that RegSetValueExA is used to create registry keys, and RegDeleteValueA is used to remove a value from registry. If you look at the library column, we can see that both functions are imported from the ADVAPI32.dll library:

imports (115)	flag (25)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (0)	technique (12)	type (5)	ordinal (1)	library (2)
680 (IsUserAnAdmin)	x	0x000000000000002A8	0x000000000000002A8	0 (0x0000)	security	-	implicit	x	SHFL32.dll
RegSetValueExA	x	0x0000000000003C64A	0x0000000000003C64A	680 (0x02A8)	registry	T1112 Modify Registry	implicit	-	ADVAPI32.dll
RegDeleteValueA	x	0x0000000000003C624	0x0000000000003C624	626 (0x0272)	registry	T1485 Data Destruction	implicit	-	ADVAPI32.dll

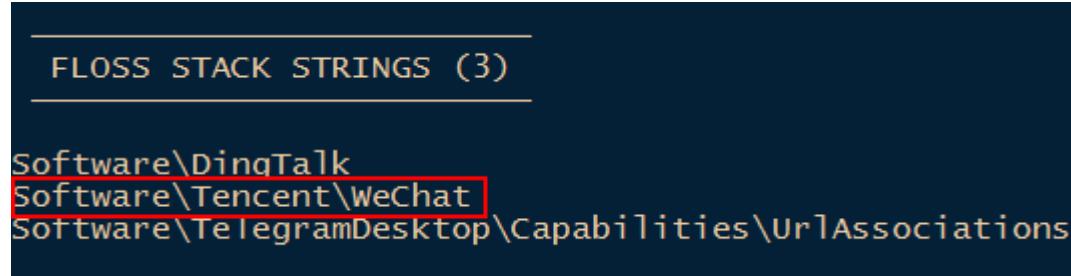
Answer: ADVAPI32.dll

Certain strings may reveal specific information. What is the name of the Chinese messaging app discovered in the basic static analysis?

Whilst there are many tools you can use to extract strings, a popular one is called FLOSS, which is a tool developed by Mandiant that extracts and deobfuscates all strings from binaries. Basic usage is simple:

- `floss <binary_to_analyse>`

After Floss has completed analysing the binary, we can find an interesting stack string:



FLOSS STACK STRINGS (3)

Software\DiqqTalk
Software\Tencent\WeChat
Software\TelegramDesktop\Capabilities\UrlAssociations

Stack strings are created during the program's run-time and are stored temporarily on the stack. In this case, we can see a stack string from "WeChat", which is a popular Chinese multi-purpose app, with features like messaging, social media, and much more.

Answer: WeChat

The Windows API can be used for malicious purposes. Which Windows API is used to destroy previously generated encryption keys?

Navigating back to flagged functions in PEStudio, we can find a series of cryptographic related functions all imported from ADVAPI32.dll:

<code>CryptDestroyKey</code>	x	0x000000000003C6B8	0x000000000003C6B8	200 (0x00C8)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll
<code>CryptDecrypt</code>	x	0x000000000003C690	0x000000000003C690	197 (0x00C5)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll
<code>CryptCreateHash</code>	x	0x000000000003C67E	0x000000000003C67E	196 (0x00C4)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll
<code>CryptDeriveKey</code>	x	0x000000000003C66C	0x000000000003C66C	198 (0x00C6)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll
<code>CryptHashData</code>	x	0x000000000003C65C	0x000000000003C65C	217 (0x00D9)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll
<code>CryptDestroyHash</code>	x	0x000000000003C636	0x000000000003C636	199 (0x00C7)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll
<code>CryptReleaseContext</code>	x	0x000000000003C60E	0x000000000003C60E	220 (0x00DC)	crypto	T1027 Obfuscated Files or Information	implicit	-	ADVAPI32.dll

The one that stands out is `CryptDestroyKey`, which frees a cryptographic key handle, making it invalid for further use.

Answer: `CryptDestroyKey`

Knowing the attacker's IP can help trace the source of the attack and gather information about their location and network. What IP address is found in the executable that belongs to Hong Kong?

To extract IP addresses from this binary, I am going to use a combination of the strings command and CyberChef:

- `strings .\malware.exe > strings.txt`

Using the Extract IP addresses recipe in CyberChef, we can find one IP address:

The screenshot shows the CyberChef interface with the 'Extract IP addresses' recipe selected. In the 'Input' pane, there is raw binary data: `y / H%`, `JY?`, `*KK`, `-83 ' =`, and `\xg`. Below this, the byte count is `rec 126594` and the character count is `21448`. In the 'Output' pane, two IP addresses are listed: `154.82.85.12` and `1.0.0.144`. The first IP address is highlighted with a red border.

Upon submitting this IP on VirusTotal, we can see that it geolocates to Hong Kong:

The screenshot shows the VirusTotal analysis page for the IP address 154.82.85.12. The page displays a community score of 5/95, indicating 5 out of 95 security vendors flagged the IP as malicious. The IP is also associated with AS 399077 (TERAEXCH). The geolocation is listed as HONG KONG (HK), and the last analysis date is 6 days ago.

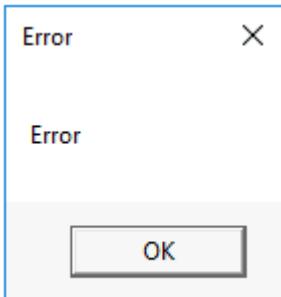
Answer: 154.82.85.12

In dynamic analysis, we examine the behavior of the malware and identify any suspicious activities. What message is displayed on the screen when the binary is executed?

To start dynamical analysis on this sample, I am going to use a Sysinternals tool called Process Monitor (ProcMon). Make sure that the capture is enabled before executing the binary:

The screenshot shows the Process Monitor interface. The title bar reads "Process Monitor - Sysinternals". The menu bar includes "File", "Edit", "Event", "Filter", and "Help". Below the menu is a toolbar with icons for File, Edit, Event, Filter, and Capture (represented by a blue circle icon). At the bottom, there are buttons for "Time ...", "Proc", and "Capture (Ctrl+E)".

When executing this binary, we can immediately see a popup that displays "Error":



After the binary has finished execution, make sure to stop the capture in ProcMon.

Answer: Error

Identifying the executed DLLs gives us insight into the attacker's strategies and goals. What is the name of the first DLL file that is loaded after the binary is executed?

To reduce the noise from other processes running on the system, we can filter for events related to this malware sample by clicking the Process Tree button, Selecting the binary, and clicking Include Subtree:

Process	Description	Image Path	Life Time	Company	Owner	Comm
msdtc.exe (1892)	Microsoft Distribut...	C:\Windows\Syst...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	C:\Wi
svchost.exe (3284)	Host Process for ...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	EC2AMAZ-B1MT...	C:\Wi
amazon-smi-agent.exe (183)		C:\Program Files\...	00:00:00.000000000	NT AUTHORITY\...	"C:\P	
sas.exe (612)	Local Security Aut...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	C:\Wi
winlogon.exe (528)	Windows Logon A...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	winlog
dwm.exe (472)	Desktop Window ...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	Window Manager...	"dwm
LogonUI.exe (3016)	Windows Logon ...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	"Log
MicrosoftEdgeUpdate.exe (1564)	Microsoft Edge U...	C:\Program Files\...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	"C:\P
crss.exe (2476)	Client Server Run...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	%Syst
winlogon.exe (2500)	Windows Logon A...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	NT AUTHORITY\...	winlog
dwm.exe (2640)	Desktop Window ...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	Window Manager...	"dwm
Explorer.EXE (3612)	Windows Explorer	C:\Windows\Expl...	00:00:00.000000000	Microsoft Corporat...	EC2AMAZ-B1MT...	C:\Wi
pestudo.exe (4788)	Malware Initial Ass...	C:\Users\Administr...	00:00:00.000000000	www.winrar.com	EC2AMAZ-B1MT...	"C:\U
cmd.exe (1372)	Windows Comma...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	EC2AMAZ-B1MT...	"C:\W
conhost.exe (3112)	Console Window ...	C:\Windows\syst...	00:00:00.000000000	Microsoft Corporat...	EC2AMAZ-B1MT...	\?C
Procmon.exe (2852)	Process Monitor	C:\Users\Administr...	00:00:00.000000000	Sysinternals - ww...	EC2AMAZ-B1MT...	"C:\U
malware.exe (3476)	ZEON Gaaiho Do...	C:\Users\Administr...	00:00:00.000000000	Zeon Corporation	EC2AMAZ-B1MT...	"C:\U

Process Details:

- Description: ZEON Gaaiho Doc Application
- Company: Zeon Corporation
- Path: C:\Users\Administrator\Desktop\Start Here\Artifacts\malware.exe
- Command: "C:\Users\Administrator\Desktop\Start Here\Artifacts\malware.exe"
- User: EC2AMAZ-B1MTPOB\Administrator
- PID: 3476 Started: 12/26/2025 3:15:48 AM
- Exited: 12/26/2025 3:16:18 AM

Buttons:

- Go To Event
- Include Process
- Include Subtree
- Close

Immediately you will find Load Image events, the first being ntdll.dll:

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
3:15:4...	malware.exe	3476	Process Start		SUCCESS	Parent PID: 3612, ...
3:15:4...	malware.exe	3476	Thread Create		SUCCESS	Thread ID: 4064
3:15:4...	malware.exe	3476	Load Image	C:\Users\Administrator\Desktop\Start Here\Artifacts\malware.exe	SUCCESS	Image Base: 0x7f7...
3:15:4...	malware.exe	3476	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7f9...
3:15:4...	Administrator	3476	Load Image	C:\Windows\system32\kernel32.dll	PREPARE	Desired Access: 0x1...

Answer: ntdll.dll

Registry enumeration involves listing all the keys and values in the Windows Registry that a process has accessed to understand its structure and contents. What is the full path of the registry key associated with fallback handling in language packs that was successfully enumerated?

Let's start by limiting the results in ProcMon to only show registry events, we can do so by unselecting these three boxes and making sure the only option enabled is for registry events:



To reduce the noise even further, select the filter button and filter for the RegQueryValue operation:

Column	Relation	Value	Action
<input checked="" type="checkbox"/> PID	is	3476	Include
<input checked="" type="checkbox"/> Operation	is	RegQueryValue	Include

Scrolling through the results, we can see that the malware queried HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback multiple times:

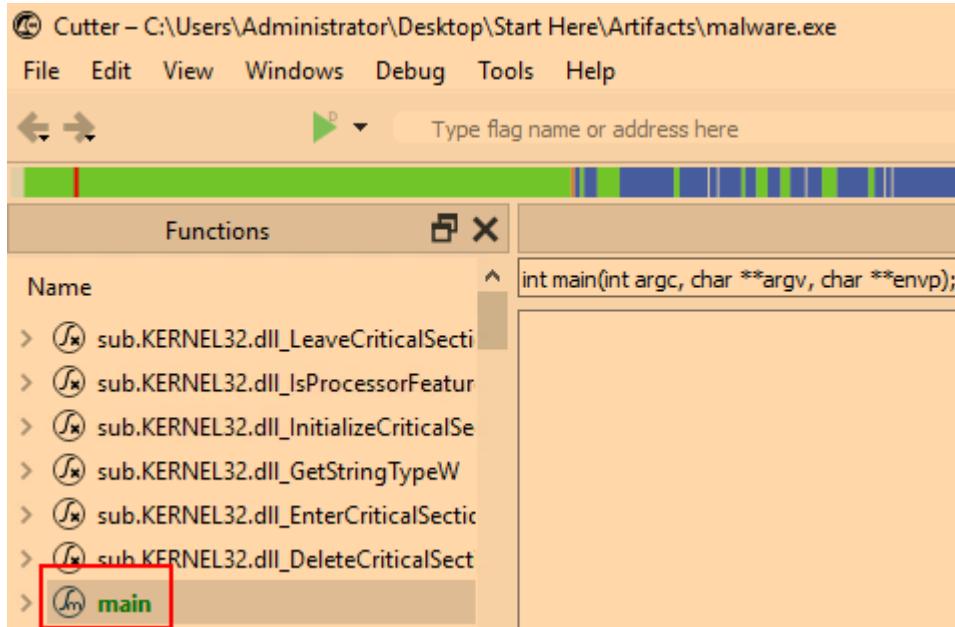
malware.exe	3476	RegQueryValue	HKLM\System\CurrentControlSet\Control\Nls\Language Groups\1	SUCCESS	Type: REG_SZ, Le...
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\DataStore_V1.0\Disable	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback	SUCCESS	Type: REG_SZ, Le...
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane1	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane2	SUCCESS	Type: REG_SZ, Le...
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane3	SUCCESS	Type: REG_SZ, Le...
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane4	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane5	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane6	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane7	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane8	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane9	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane10	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane11	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane12	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane13	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane14	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane15	NAME NOT FOUND	Length: 144
malware.exe	3476	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback\Plane16	NAME NOT FOUND	Length: 144

This registry key holds settings for how Windows handles surrogate characters when a specific font isn't available.

Answer: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack\SurrogateFallback

Tracing Windows API calls helps understand what the malware is intended to do by analyzing specific patterns or arguments used in these calls. What is the first Windows API call made from the function that is called from the main function?

To determine the first API call in the main function, we need to disassemble the malware. Cutter is a free and open-source reverse engineering tool that we can use for this purpose. Once cutter has loaded the binary, locate the main function on the left-hand side pane:



```
[0x140004690]
int main(int argc, char **argv, char **envp);
0x140004690    sub    rsp, 0x28
0x140004694    call   fcn.140003d9c ; fcn.140003d9c
0x140004699    int3
```

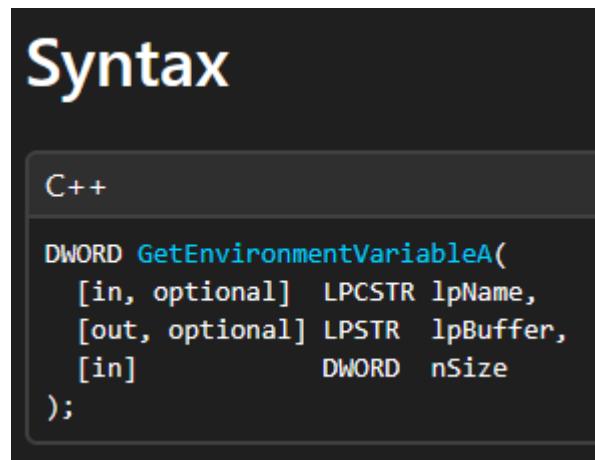
This is a very small function that calls another function, meaning it will execute the code within the address supplied to call. If you double click “0x140004694”, it will take us to the function that gets executed. Within this function, we can see that the first API call made is GetEnvironmentVariableA:

```
call    qword [GetEnvironmentVariableA]
```

Answer: GetEnvironmentVariableA

Understanding the number of arguments passed helps identify the data type being transmitted or processed. How many arguments does the API mentioned in the previous question accept?

Viewing the documentation for “GetEnvironmentVariableA”, we can see that it accepts three arguments:



Those being lpName, lpBuffer, and nSize:

Parameters

[in, optional] lpName

The name of the environment variable.

[out, optional] lpBuffer

A pointer to a buffer that receives the contents of the specified environment variable as a null-terminated string. An environment variable has a maximum size limit of 32,767 characters, including the null-terminating character.

[in] nSize

The size of the buffer pointed to by the *lpBuffer* parameter, including the null-terminating character, in characters.

We can see this within the disassembled code in Cutter:

```
mov    r8d, esi ; DWORD nSize
mov    rdx, rax ; LPSTR lpBuffer
lea    rcx, [str.TEMP] ; 0x140038394 ; LPCSTR lpName
call   qword [GetEnvironmentVariableA] ; 0x140027198 ;
```

Answer: 3

Identifying specific details within a binary can provide insights into the target's identity or the attacker's intent. What is the name of the company embedded in the binary?

Navigating back to PEStudio, we can find the company name in the version header:

c:\users\administrator\Desktop\start here\artifa		property	value
indicators (imports > flag)		footprint > sha256	1046F597CC0B6EE2F95097551F6E0B2208FF1D2EAC0AB6ADB28FBA7694BA7D99
footprints (type > sha256)		location	.rsrc:0x00104C70
virustotal (status > error)		language	neutral
dos-header (size > 64 bytes)		code-page	Unicode UTF-16, little endian
dos-stub (size > 200 bytes)		FileVersion	1.0.0.144
rich-header (tooling > Visual Studio 2017)		ProductVersion	1.0.0.144
file-header (executable > 64-bit)		FileDescription	ZEON Gaaiho Doc Application
optional-header (subsystem > GUI)		CompanyName	Zeon Corporation
directories (count > 8)		OriginalFilename	DefaultViewer.exe
sections (count > 6)		ProductName	ZEON Gaaiho Doc
libraries (count > 6)		LegalCopyright	Copyright (C) 1993-2010 ZEON Corporation. All rights reserved.
imports (flag > 115)			
exports (n/a)			
thread-local-storage (n/a)			
.NET (n/a)			
resources (count > 10)			
strings (flag > 39)			
debug (stamp > Jul.2024)			
manifest (n/a)			
version (FileDescription > ZEON Gaaiho Doc)			
certificate (issued-to > error)			
overlay (signature > unknown)			

Answer: Zeon Corporation

Injecting shellcode through a Windows callback function involves placing the shellcode into a process and using a callback mechanism. What is the memory location of VirtualAlloc as identified in the Capa rules related to this technique?

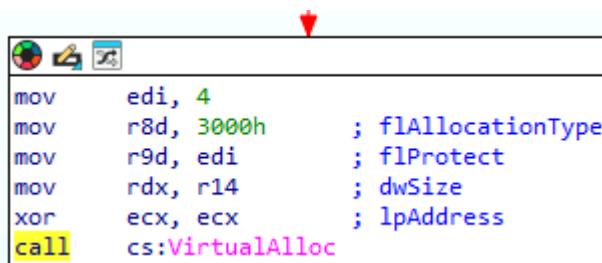
Capa is a tool developed by Mandiant that detects capabilities in executable files. To run Capa and display the virtual location of detected capabilities, execute the following command (make sure to use the -vv option):

- capa -vv malware.exe

The -vv flag (very verbose) reports exactly where Capa finds capabilities, which is useful to verify the output and analyse the sample further. Within the output, we can find the location of VirtualAlloc as identified by Capa:

```
execute shellcode via indirect call
namespace load-code/shellcode
author ronnie.salomonsen@mandiant.com
scope function
mbc Memory::Allocate Memory [C0007]
function @ 0x14000469C
and:
match: allocate RWX memory @ 0x140004A1E
and:
match: allocate memory @ 0x140004A1E
or:
api: kernel32.VirtualAlloc @ 0x140004A31
number: 0x40 = PAGE_EXECUTE_READWRITE @ 0x140004A92
or:
characteristic: indirect call @ 0x140004730, 0x140004732, 0x140004754, 0x140004764, and 4 more...
```

This is extremely helpful as we can take the base address from capa, rebase our image and navigate to this address to analyse suspicious behaviour further. For example, I loaded the binary into IDA, navigated to Edit > Segments > Rebase program, supplied the base address given by Capa and navigated to the address identified in the previous image:



Answer: 0x140004A31