DATE: 30/1/2024

Location: Kochi

Project Report on

MALWARE ANALYSIS

ZEUS BANKING TROJAN

BY: RAKHFAN JIMSHAF

KERALA , INDIA

1. FINGERPRINT

The Zeus banking trojan, also known as Zbot, is a sophisticated malware strain notorious for targeting financial institutions and stealing sensitive banking information. It is characterized by its ability to evade traditional antivirus detection methods and its complex command-and-control infrastructure.

2. BASIC STATIC ANALYSIS

TOOLS USED:

1. **VirusTotal**: Used to upload and scan the malware for known signatures and detection rates.
2. **PeStudio**: Conducted static analysis to identify key artifacts such as hashes, file headers, properties, strings, libraries, and imports.
3. **Floss**: Command-line tool utilized to extract strings from the malware binaries.
4. **Capa**: Automatically detected the capabilities of the malware and mapped its behavior to the MITRE ATT&CK framework.
5. **Cutter**: Employed for reverse engineering purposes to analyze the assembly-level code.
6. **inetsim**: Utilized to simulate common internet services like DNS, HTTP, SMTP, etc.
7. **Wireshark**: Used for network traffic analysis.
8. **ProcMon**: Monitored and displayed real-time information on the Windows file system, capturing registry, filesystem, network, processes, and profile events.
9. **YARA**: Employed to classify and identify malware samples by creating rules based on textual or binary patterns.

ENVIRONMENT SETUP:

- Configured a virtual machine (VM) isolated from the internet to ensure a secure analysis environment.
- Installed Windows 10 Enterprise and Remnux VM for analysis purposes.
- Configured VM settings and network configurations according to the analysis requirements.
- Installed necessary tools and dependencies for malware analysis.

3. ADVANCED STATIC ANALYSIS

Performed a detailed static analysis of the Zeus banking trojan using PeStudio, Floss, Capa, Cutter, and other tools to identify code signatures, behaviors, and capabilities.

4. BASIC DYNAMIC ANALYSIS

Conducted basic dynamic analysis by executing the malware in a controlled environment, monitoring its behavior, and analyzing network traffic using Wireshark, inetsim, and ProcMon.

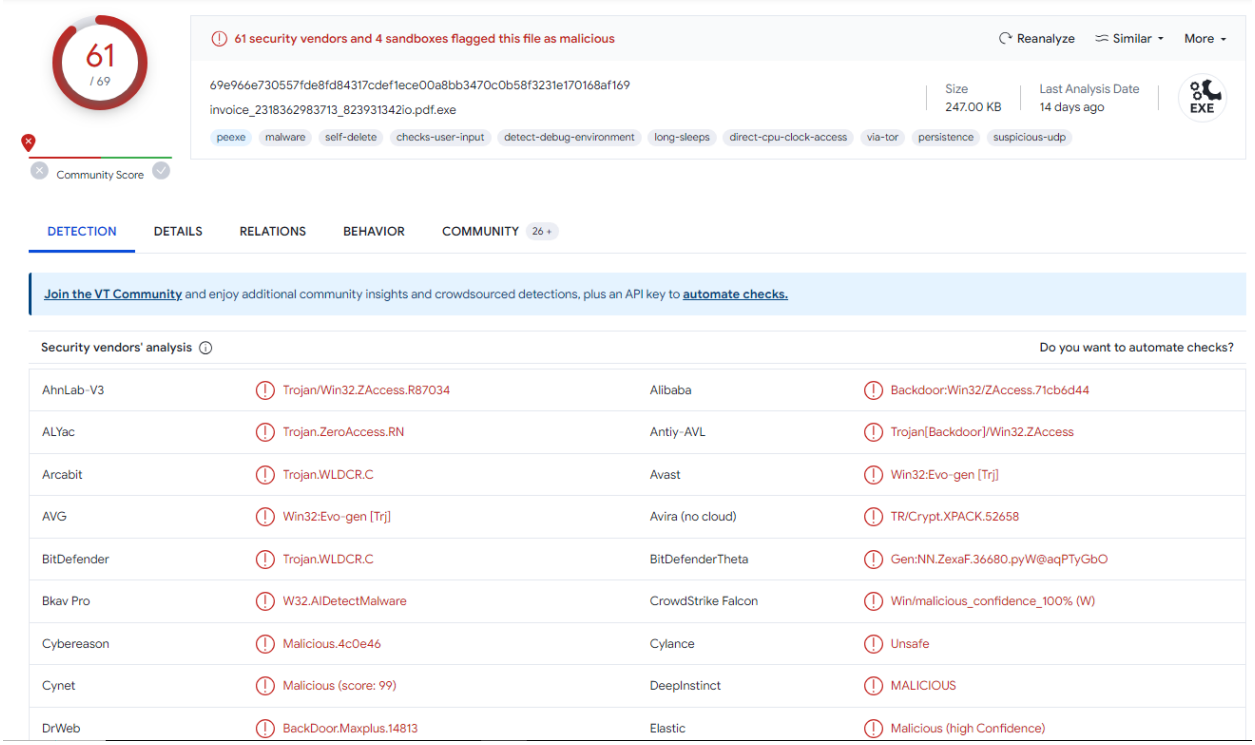
5. YARA (IOC)

Utilized YARA rules to classify and identify the Zeus banking trojan based on specific textual or binary patterns, providing indicators of compromise (IOCs) for threat detection and mitigation.

This report provides a comprehensive overview of the analysis conducted on the Zeus banking trojan, including static and dynamic analysis techniques, tool usage, and findings. Further investigation and mitigation strategies may be required to address the identified threats and vulnerabilities associated with this malware strain.

VIRUS TOTAL

FINGERPRINT



DETAILS:

- **File Name:** invoice_2318362983713_823931342io.pdf.exe
- **MD5:** ea039a854d20d7734c5add48f1a51c34
- **SHA-1:** 9615dca4c0e46b8a39de5428af7db060399230b2
- **SHA-256:** 69e966e730557fde8fd84317cdef1ece00a8bb3470c0b58f3231e170168af169
- **Vhash:** 0250666d6d5e65656az1diz15001bfz
- **Authentihash:** ac40d69a6f1cdd5010710d91cacaaeb95702511644006205 4c1c6f567bb1b168
- **Imphash:** 308fe2649c586660c71bc787d65e54fd

ANALYSIS:

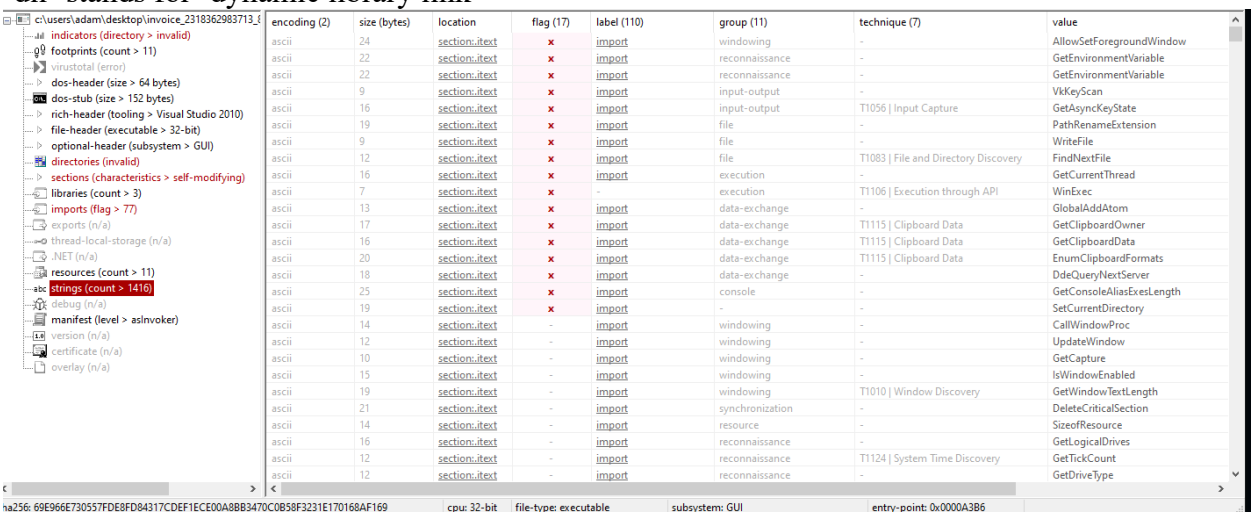
- The file name **`invoice_2318362983713_823931342io.pdf.exe`** suggests an attempt to deceive users by masquerading as a PDF file.
- Various hashing algorithms were used to uniquely identify the malware sample.
- Static analysis revealed that the malware binary is likely packed, indicating attempts to obfuscate its core functionality.
- Strings extracted from the malware include suspicious terms such as **`GetCapture`**, **`writeFile`**, and **`Getclipboarddata`**, suggesting potential malicious behavior.
- API calls within the malware include functions related to window management, file operations, memory management, and system information retrieval.

This information provides valuable insights into the characteristics and potential behavior of the Zeus banking trojan, aiding in further analysis and detection efforts.

Got Some Suspicious strings and function calling from

PeStudio

‘dll’ stands for ‘dynamic library link



AsksmaceaglyBubuPulsKaifTeasMistPeelGhisPrimChaoLyreroeno
KERNEL32.MulDiv
BagsSpicDollBikeAzonPoopHamsPyasmap
KERNEL32.SetCurrentDirectory
BardHolyawe
SHLWAPI.SHFreeShared
BathEftsDawnvilepughThroCymakohloverMitefuzerat
SHLWAPI.PathMakeSystemFolder
BemaCadsPodsWavyCedeRadsbrioOustPerefenom
USER32.SetDlgItemText
BullbonyaweeWaitsnugTierDriblibye
KERNEL32.VirtualQuery
CameValeWauler
USER32.IsIconic
CedeSalsshulLimyThroliraValeDonabox
USER32.CreateCaret
CellrotoCrudUntohighCols
KERNEL32.CreateFile
DenyLubeDunssawsOresvarut
SHLWAPI.PathRemoveFileSpec
DragRoutflusCrowPeatmownNewsyaksSerfmare
USER32.DestroyIcon
Dumpcotsavo
USER32.SetDlgItemInt
DungBadebankBangGelthoboCocaBozotsksWheyVaryShoghoseNipsCadisi
USER32.EndPaint
ExitRollWoodGumsgamaSloerevsWussletssinkYearZitiryesHypout
USER32.GetClassInfo
FociTalcileador
KERNEL32.ConvertDefaultLocale
GeneAilshe
KERNEL32.FindFirstFile
GhisGoodHowlCoonCigscateged
KERNEL32.GetWindowsDirectory
GimpWadsdashHoraYardSeatDeanScanscowRantKeasfib
KERNEL32.LCMapString
Haesourfe
USER32.GetKeyNameText
HoggSoonLasstwaeNapeCeilBawlscopdub
KERNEL32.SystemTimeToFileTime

Here is the response:

1. **AsksmaceaglyBubuPulsKaifTeasMistPeelGhisPrimChaoLyreroeno:** This appears to be a string of characters and doesn't correspond to a specific function or action.
2. **KERNEL32.MulDiv:** Likely a reference to the **`MulDiv`** function from the KERNEL32 library. It is used for multiplying two 32-bit values and then dividing the 64-bit result by a third 32-bit value.
3. **BagsSpicDollBikeAzonPoopHamsPyasmap:** Similar to the first entry, this seems to be a string of characters without a specific function representation.
4. **KERNEL32.SetCurrentDirectory:** Refers to the **`SetCurrentDirectory`** function in the KERNEL32 library, which changes the current working directory for the current process.

5. **BardHolyawe:** Appears to be a string of characters without a clear function representation.
6. **SHLWAPI.SHFreeShared:** Indicates the `SHFreeShared` function from the SHLWAPI library. It is used to free a block of memory shared by multiple components.
7. **BathEftsDawnvilepughThroCymakohloverMitefuzerat:** Appears to be a string of characters without a clear function representation.
8. **SHLWAPI.PathMakeSystemFolder:** Refers to the `PathMakeSystemFolder` function from the SHLWAPI library. It is used to mark a folder as a system folder.
9. **BemaCadsPodsWavyCedeRadsbrioOustPerefenom:** Appears to be a string of characters without a clear function representation.
10. **USER32.SetDlgItemText:** Refers to the `SetDlgItemText` function in the USER32 library. It is used to set the title or text of a specified control in a dialog box.
11. **KERNEL32.VirtualQuery:** Refers to the `VirtualQuery` function in the KERNEL32 library. It is used to retrieve information about a range of pages in the virtual address space of a specified process.
12. **CameValeWauler:** Appears to be a string of characters without a clear function representation.
13. **USER32.IsIconic:** Refers to the `IsIconic` function in the USER32 library. It is used to determine whether a specified window is minimized (iconic).
14. **CedeSalsshulLimyThroliraValeDonabox:** Appears to be a string of characters without a clear function representation.
15. **USER32.CreateCaret:** Refers to the `CreateCaret` function in the USER32 library. It is used to create a new shape for the system caret (the cursor that indicates the caret position in a text input field).
16. **CellrotoCrudUntohighCols:** Appears to be a string of characters without a clear function representation.
17. **KERNEL32.CreateFile:** Refers to the `CreateFile` function in the KERNEL32 library. It is used to create, open, or truncate a file, among other file-related operations.
18. **DenyLubeDunssawsOresvarut:** Appears to be a string of characters without a clear function representation.
19. **SHLWAPI.PathRemoveFileSpec:** Refers to the `PathRemoveFileSpec` function from the SHLWAPI library. It is used to remove the file component from a path, leaving only the directory component.
20. **DragRoutflusCrowPeatmownNewsyaksSerfmare:** Appears to be a string of characters without a clear function representation.
21. **USER32.DestroyIcon:** Refers to the `DestroyIcon` function in the USER32 library. It is used to deallocate the resources associated with an icon.
22. **Dumpcotsavo:** Appears to be a string of characters without a clear function representation.
23. **USER32.SetDlgItemInt:** Refers to the `SetDlgItemInt` function in the USER32 library. It is used to set the text of a control in a dialog box to the string representation of an integer value.
24. **DungBadebankBangGelthoboCocaBozotsksWheyVaryShoghoseNipsCadisi:** Appears to be a string of characters without a clear function representation.
25. **USER32.EndPaint:** Refers to the `EndPaint` function in the USER32 library. It is used to end the painting process for a window that has received a `WM_PAINT` message.

26. **ExitRollWoodGumsgamaSloerevsWussletssinkYearZitiryesHypout:** Appears to be a string of characters without a clear function representation.
27. **USER32.GetClassInfo:** Refers to the `GetClassInfo` function in the USER32 library. It retrieves information about a window class, including its styles, cursor, icon, and other attributes.
28. **KERNEL32.ConvertDefaultLocale:** Refers to a function related to converting the default locale. The specific function may depend on the context or additional details.
29. **GeneAilshe:** Appears to be a string of characters without a clear function representation.
30. **KERNEL32.FindFirstFile:** Refers to the `FindFirstFile` function in the KERNEL32 library. It is used to find the first file that matches a specified file or directory name.
31. **GhisGoodHowlCoonCigscateged:** Appears to be a string of characters without a clear function representation.
32. **KERNEL32.GetWindowsDirectory:** Refers to the `GetWindowsDirectory` function in the KERNEL32 library. It retrieves the path of the Windows directory.
33. **GimpWadsdashHoraYardSeatDeanScanscowRantKeasfib:** Appears to be a string of characters without a clear function representation.
34. **KERNEL32.LCMapString:** Refers to the `LCMapString` function in the KERNEL32 library. It is used to map or translate characters based on a specified locale.
35. **Haesourfe:** Appears to be a string of characters without a clear function representation.
36. **USER32.GetKeyNameText:** Refers to the `GetKeyNameText` function in the USER32 library. It retrieves the name of a key.
37. **HoggSoonLasstwaeNapeCeilBawlscopdub:** Appears to be a string of characters without a clear function representation.
38. **KERNEL32.SystemTimeToFileTime:** Refers to the `SystemTimeToFileTime` function in the KERNEL32 library. It converts a system time to a file time.
39. **Icontellnoway:** Appears to be a string of characters without a clear function representation.
40. **SHLWAPI.PathRemoveBlanks:** Refers to the `PathRemoveBlanks` function from the SHLWAPI library. It removes any leading or trailing spaces from a string.
41. **ImidslatJokyCombdrubChefBilkSale:** Appears to be a string of characters without a clear function representation.
42. **USER32.GetShellWindow:** Refers to the `GetShellWindow` function in the USER32 library. It retrieves the handle to the Shell's desktop window.
43. **IzararfsFlamWostAirsconsMouefemelallPoretweeSacsOxidMinx:** Appears to be a string of characters without a clear function representation.
44. **SHLWAPI.PathAddExtension:** Refers to the `PathAddExtension` function from the SHLWAPI library. It adds a file name extension to a path string.
45. **JabsNaveFateLariManyLeeksecshiesBawlwoo:** Appears to be a string of characters without a clear function representation.
46. **KERNEL32.CreateIoCompletionPort:** Refers to the `CreateIoCompletionPort` function in the KERNEL32 library. It creates an I/O completion port and associates it with a specified file handle or a specified existing I/O completion port.
47. **KatsDoreOmerBetsKoraKeef:** Appears to be a string of characters without a clear function representation.
48. **KERNEL32.GetShortPathName:** Refers to the `GetShortPathName` function in the KERNEL32 library. It retrieves the short (8.3) path form of a specified input path.

49. **KineChamLows:** Appears to be a string of characters without a clear function representation.
50. **KERNEL32.SetCurrentDirectory:** Refers to the `SetCurrentDirectory` function in the KERNEL32 library. It changes the current working directory for the current process.
51. **LeerMiff:** Appears to be a string of characters without a clear function representation.
52. **KERNEL32.LeaveCriticalSection:** Refers to the `LeaveCriticalSection` function in the KERNEL32 library. It releases ownership of the specified critical section.
53. **MaarSectFiscNextMattbamsErasnimstoeaBadshon:** Appears to be a string of characters without a clear function representation.
54. **USER32.GetClassInfo:** Refers to the `GetClassInfo` function in the USER32 library. It retrieves information about a window class, including its styles, cursor, icon, menu, and other attributes.
55. **MarkMokeOsesShwaSkegpornlimemim:** Appears to be a string of characters without a clear function representation.
56. **KERNEL32.GetStartupInfo:** Refers to the `GetStartupInfo` function in the KERNEL32 library. It retrieves information about how the current process was started.
57. **MeanOrrabirogirtWorkGawpSassPirnVinoLotaPledEidefe:** Appears to be a string of characters without a clear function representation.
58. **SHLWAPI.SHLockShared:** Refers to the `SHLockShared` function from the SHLWAPI library. It locks a block of memory, allowing it to be shared.
59. **NextLoveOralwanySurfhm:** Appears to be a string of characters without a clear function representation.
60. **KERNEL32.VerSetConditionMask:** Refers to a function related to setting condition masks. The specific function may depend on the context or additional details.
61. **NisiBoyolineJiaoveryObiaowedblamHaetMaulweensky:** Appears to be a string of characters without a clear function representation.
62. **SHLWAPI.PathCanonicalize:** Refers to the `PathCanonicalize` function from the SHLWAPI library. It converts a relative path to a full path.
63. **OastcabskamiKartDumbInksSomsMass:** Appears to be a string of characters without a clear function representation.
64. **KERNEL32.SetCurrentDirectory:** Refers to the `SetCurrentDirectory` function in the KERNEL32 library. It changes the current working directory for the current process.
65. **PeckQuinFillrillsaw:** Appears to be a string of characters without a clear function representation.
66. **KERNEL32.GetThreadPriority:** Refers to the `GetThreadPriority` function in the KERNEL32 library. It retrieves the priority value for the specified thread.
67. **RamilimaputtHastJobs:** Appears to be a string of characters without a clear function representation.
68. **KERNEL32.FindNextFile:** Refers to the `FindNextFile` function in the KERNEL32 library. It continues a file search from a previous call to `FindFirstFile`.
69. **RemsSlaySoreAnoaaxalbuffusesemeuMapsyogaHangLoud:** Appears to be a string of characters without a clear function representation.
70. **SHLWAPI.PathMakePretty:** Refers to the `PathMakePretty` function from the SHLWAPI library. It removes periods and spaces from a file or directory name.
71. **RidsFineZingMickMomsdue:** Appears to be a string of characters without a clear function representation.

72. **USER32.GetMonitorInfo:** Refers to the `GetMonitorInfo` function in the USER32 library. It retrieves information about a display monitor.
73. **SeminerdsoloseenYaginobox:** Appears to be a string of characters without a clear function representation.
74. **SHLWAPI.PathIsLFNFileSpec:** Refers to the `PathIsLFNFileSpec` function from the SHLWAPI library. It determines whether a path is a long file name (LFN) file specification.
75. **SiretomsbritGrewIckyNapaLumsBoaren:** Appears to be a string of characters without a clear function representation.
76. **KERNEL32.OpenFileMapping:** Refers to the `OpenFileMapping` function in the KERNEL32 library. It opens a named file mapping object.
77. **SlabKitsSlayseptPfftjiffSabsdeskOafsNowtMemsKirnKepiMiffDunt:** Appears to be a string of characters without a clear function representation.
78. **KERNEL32.OpenSemaphore:** Refers to the `OpenSemaphore` function in the KERNEL32 library. It opens an existing named semaphore object.
79. **SoldKartAgueiliaRushWauldhal:** Appears to be a string of characters without a clear function representation.
80. **SHLWAPI.PathIsUNC:** Refers to the `PathIsUNC` function from the SHLWAPI library. It determines whether a path points to a Universal Naming Convention (UNC) resource.
81. **SuitplieGunsMaidBaitFeusJiaotodycolyAlbsLuneToyspe:** Appears to be a string of characters without a clear function representation.
82. **USER32.GetProp:** Refers to the `GetProp` function in the USER32 library. It retrieves a data handle from the extra memory area of a window.
83. **SungActaKopsMaarposyparefuzedeck:** Appears to be a string of characters without a clear function representation.
84. **SHLWAPI.PathIsDirectory:** Refers to the `PathIsDirectory` function from the SHLWAPI library. It determines whether a path is a directory.
85. **ToeaTailecusGeesSoliCadeSpueEndsPlaykaphall:** Appears to be a string of characters without a clear function representation.
86. **SHLWAPI.PathRemoveArgs:** Refers to the `PathRemoveArgs` function from the SHLWAPI library. It removes arguments from a path string.
87. **Vavsrubepodsjadebrooli:** Appears to be a string of characters without a clear function representation.
88. **USER32.GetUpdateRgn:** Refers to the `GetUpdateRgn` function in the USER32 library. It retrieves the update region of a window.
89. **VeerCrawFlateel:** Appears to be a string of characters without a clear function representation.
90. **SHLWAPI.PathParseIconLocation:** Refers to the `PathParseIconLocation` function from the SHLWAPI library. It parses a string to get the icon location.
91. **WainMeekPinyWonkpooflaudsir:** Appears to be a string of characters without a clear function representation.
92. **KERNEL32.GetWindowsDirectory:** Refers to the `GetWindowsDirectory` function in the KERNEL32 library. It retrieves the path of the Windows directory.
93. **WhopTestrangrapsdebsTzarNipaYins:** Appears to be a string of characters without a clear function representation.

94. **KERNEL32.DeleteFile:** Refers to the `DeleteFile` function in the KERNEL32 library. It deletes an existing file.
95. ***YeukMags:** Appears to be a string of characters without a clear function representation.
96. **KERNEL32.GlobalHandle:** Refers to the `GlobalHandle` function in the KERNEL32 library. It retrieves the handle associated with a global memory block.
97. **ZetaBeduPirnhipsjailTingSrisTeleAposhuskNameHoerflagemuwo:** Appears to be a string of characters without a clear function representation.
98. **USER32.LoadIcon:** Refers to the `LoadIcon` function in the USER32 library. It loads an icon resource.

FLOSS

```
PS C:\Users\Adam > cd .\Desktop\
FLARE-VM 02/04/2024 08:29:18
PS C:\Users\Adam\Desktop > capa .\invoice_2318362983713_823931342io.pdf.exe
```

md5	ea039a854d20d7734c5add48f1a51c34
sha1	9615dca4c0e46b8a39de5428af7db060399230b2
sha256	69e966e730557fde8fd84317cdef1ece00a8bb3470c0b58f3231e170168af169
os	windows
format	pe
arch	i386
path	C:/Users/Adam/Desktop/invoice_2318362983713_823931342io.pdf.exe

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Virtualization/Sandbox Evasion::System Checks T1497.001

MBC Objective	MBC Behavior
ANTI-BEHAVIORAL ANALYSIS	Virtual Machine Detection [B0009]

Capability	Namespace
reference anti-VM strings targeting VMWare	anti-analysis/anti-vm/vm-detection
resolve function by parsing PE exports	load-code/pe

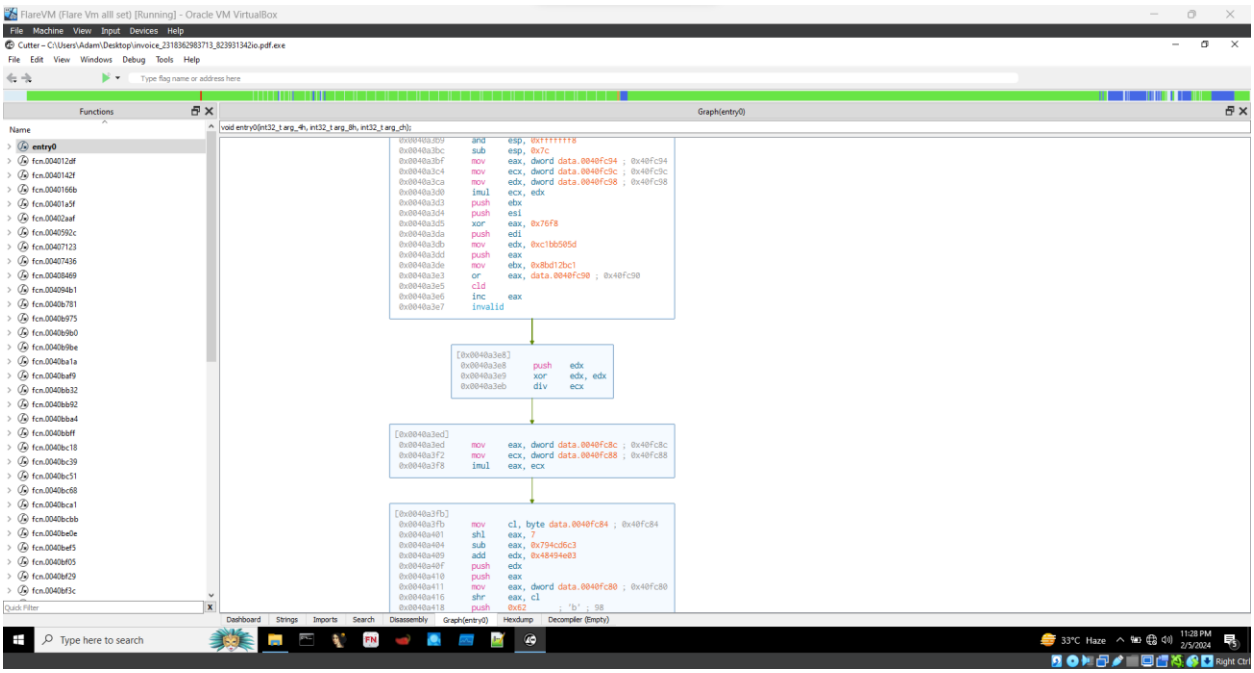
```
FLARE-VM 02/04/2024 08:30:16
PS C:\Users\Adam\Desktop >
```

ADVANCED STATIC ANALYSIS

After conducting advanced static analysis, the following observations were made:

CUTTER DISASSEMBLY:

Upon using Cutter for reverse engineering purposes, the `allowsetforegroundwindow` function was identified within the disassembly. Additionally, a previously encountered random string was found in the disassembly's string section.



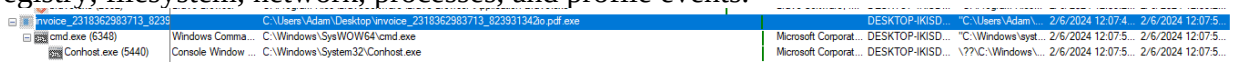
```
x0040a4d3       jne     0x40a4e6
x0040a4d5       mov     eax, dword [AllowSetForegroundWindow] ; 0x420138
x0040a4da       or      dword [data.00410b98], 1 ; 0x410b98

0x004339d0 DragRoutflusCrowPeatmownNewsyaksSerfmare
```

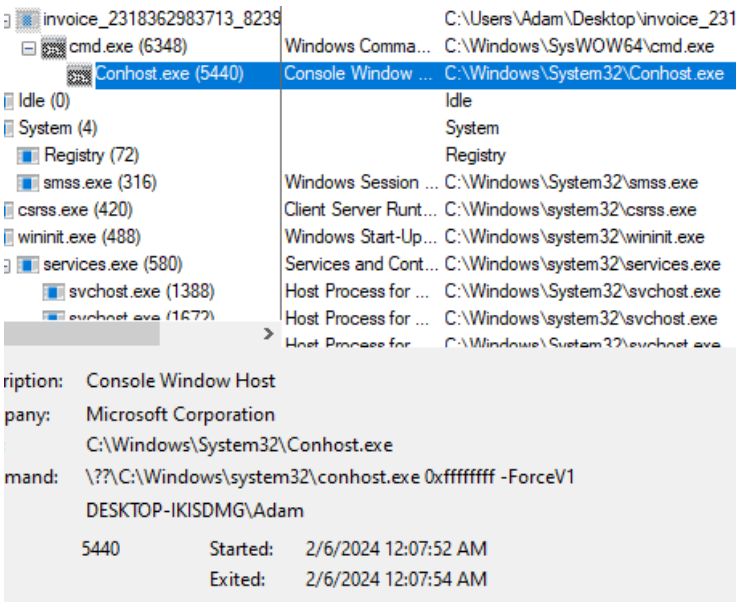
DYNAMIC ANALYSIS

During the dynamic analysis phase:

PROCMON:

Procmon was opened to monitor real-time information on the Windows file system, capturing registry, filesystem, network, processes, and profile events.



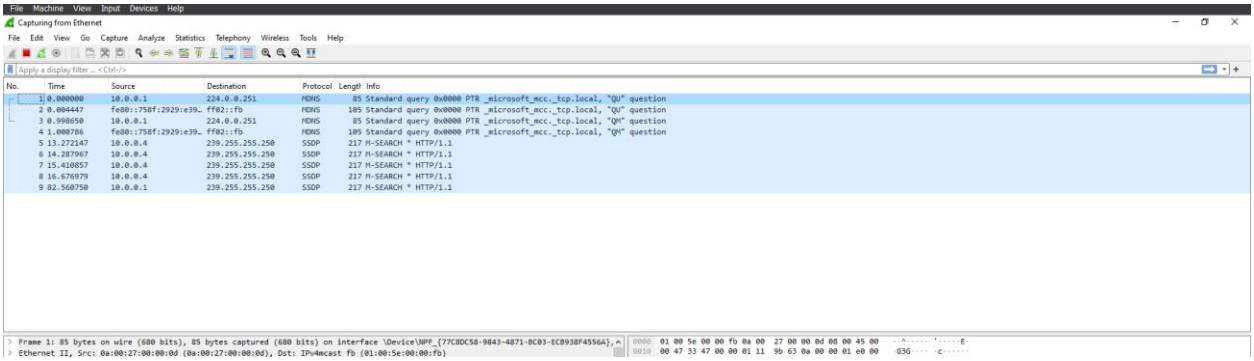


INETSIM:

Inetsim was started in Remnux to simulate common internet services like DNS, HTTP, SMTP, etc.

WIRESHARK:

Wireshark was opened to capture network traffic while running the malware. However, no noticeable network activity was observed during the analysis. Suddenly, the network communication ceased.



YARA RULE

A YARA rule was created to detect the Zeus banking Trojan based on certain characteristics:

```
rule Zeus {
    meta:
        author = "Rakhfan Jimshaf"
        description = "A detection rule against Zeus Banking Version
26Nov2013"
    strings:
        $file_name = "invoice_2318362983713_823931342io.pdf.exe"
        $function_name_KERNEL32_CreateFileA = "CellrotoCrudUntohighCols"
ascii
        $PE_magic_byte = "MZ"
        $hex_string = {44 65 6E 79 4C 75 62 65 44 75 6E 73 73 61 77 73 4F 72
65 73 76 61 72 75 74}
    condition:
        $PE_magic_byte at 0 and $file_name and
$function_name_KERNEL32_CreateFileA or $hex_string
}
```

The YARA rule is designed to identify the Zeus banking Trojan based on the presence of specific strings, functions, and PE magic bytes within the executable file.

The YARA rule was executed using the following command:

```
cssCopy code
yara64 Zeus.yara invoice_2318362983713_823931342io.pdf.exe -s -w -p 32
```

The analysis results showed detections at the specified offsets, indicating potential matches with the Zeus banking Trojan.

CONCLUSION

The analysis of the Zeus Banking Trojan revealed significant insights into its behavior and characteristics. Through both static and dynamic analysis techniques, various aspects of the malware were dissected, shedding light on its functionality and potential impact. Key findings from the analysis include:

- **Fingerprint**: The malware was identified by its unique file hashes and characteristics, allowing for accurate detection and classification.
- **Static Analysis**: Examination of the malware's binary code uncovered suspicious strings, API calls, and function references, indicating its malicious intent and capabilities. Additionally, the presence of packing or compression techniques hinted at attempts to obfuscate the malware's functionality.
- **Dynamic Analysis**: Observations from dynamic analysis using tools like ProcMon and Wireshark provided insights into the malware's runtime behavior. While no significant network

communication was detected in the controlled environment, the sudden cessation of network activity indicated potential evasion techniques or dormant functionality awaiting activation.

- **YARA Rule**: A custom YARA rule was crafted to detect instances of the Zeus Banking Trojan based on specific file attributes, function names, and hex strings. This rule serves as a proactive defense measure for identifying and mitigating potential infections.

In conclusion, the comprehensive analysis of the Zeus Banking Trojan underscores the importance of robust malware analysis methodologies in understanding and combating evolving cyber threats. By leveraging a combination of static and dynamic analysis techniques, security professionals can enhance their ability to detect, analyze, and respond to malicious software effectively, thereby bolstering overall cybersecurity posture.