Mystic Stealer

TECHNICAL ANALYSIS REPORT

ZAYOTEM

ZARARLI YAZILIM ÖNLEME VE TERSİNE MÜHENDİSLİK

Contents

CONTENTS 1
OVERVIEW 2
MYSTIC.EXE ANALYSIS 3
STATIC ANALYSIS 4
DYNAMIC ANALYSIS 5
DUMP.EXE ANALYSIS 12
NETWORK ANALYSIS 15
YARA RULES 17
MITRE ATTACK TABLE 19
RECOMMENDATIONS 19
PREPARED BY 20

Overview

Mystic Stealer is a new malware that emerged in April 2023 and quickly gained a significant position in the cybercrime world. This malicious software targets credentials from web browsers, browser extensions, cryptocurrency applications, multi-factor authentication (MFA) systems, and password management applications, as well as from popular social media platforms. Mystic Stealer employs advanced techniques to bypass and neutralize analysis and defense mechanisms, including avoiding static analysis methods. The use of these advanced techniques not only enables Mystic Stealer to access a wide range of data but also makes it a sophisticated threat that is difficult to detect.

Mystic.exe Analysis

Name	Mystic.exe
MD5	692a59e85b4c932049ab55cb372a9509
SHA256	dc094161dd0f8395e5363c61b3364191562edc470c785802d55d86f14bc40eaa
File Type	Portable Executable 32 (x86)

The malicious software's details such as MD5, SHA256 are provided in the table above. The original name of the malware, “dc094161dd0f8395e5363c61b3364191562edc470c785802d55d86f14bc40eaa.exe”, has been renamed to “Mystic.exe” for ease of analysis during the investigation.

Static Analysis

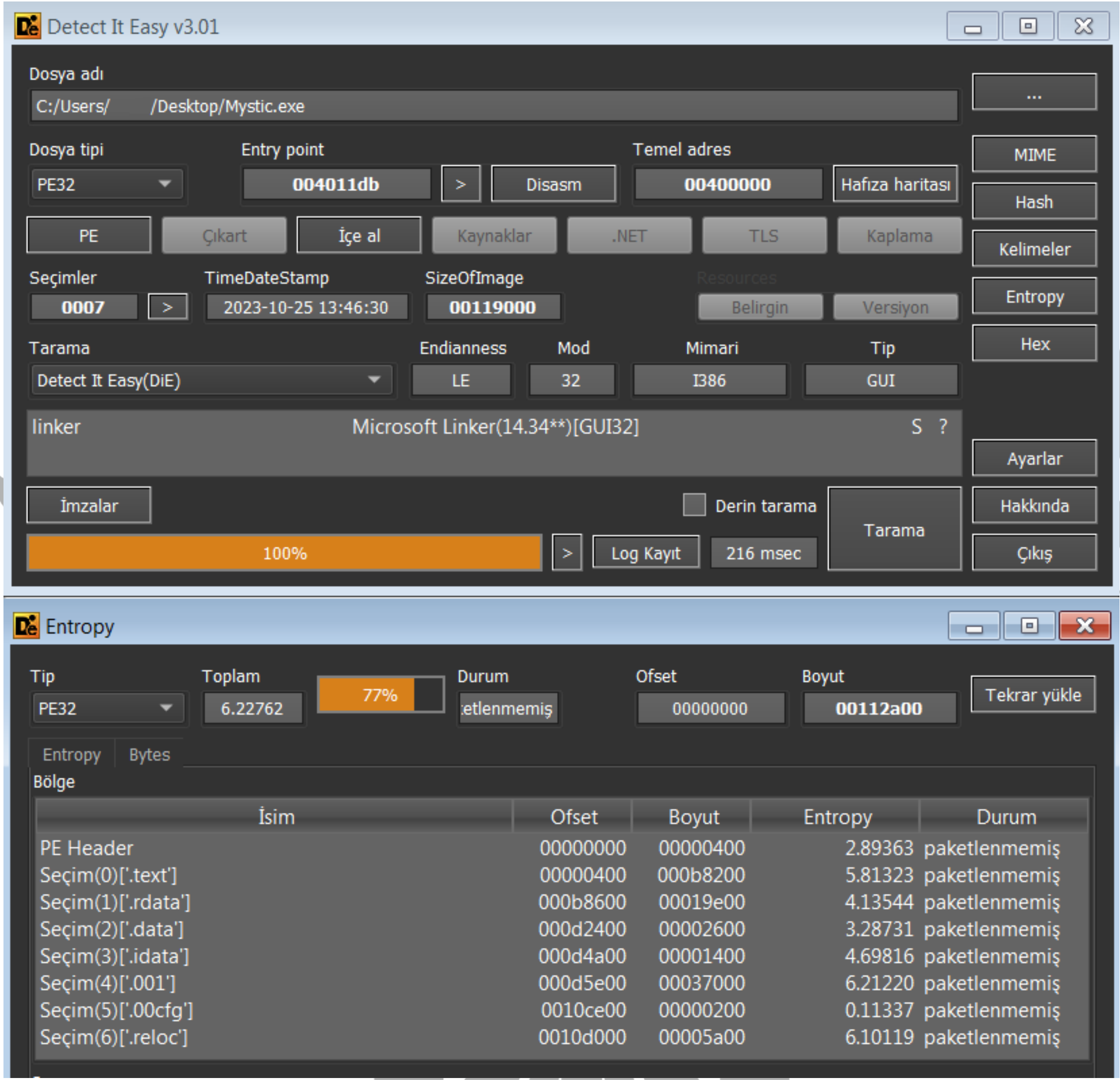


Figure 1 – Static Analysis of the Malware

When examined, it is observed that the Mystic.exe malware is not packaged.

Dynamic Analysis

```
sub_1297350
sub_1297350
sub_1297350        ; Attributes: bp-based frame
sub_1297350
sub_1297350        sub_1297350 proc near
sub_1297350        push    ebp
sub_1297350+1      mov     ebp, esp
sub_1297350+3      push    2DA00h
sub_1297350+8      push    135B0F0h
sub_1297350+D      call    sub_12813DE     ; Call Procedure
sub_1297350+12     add     esp, 8          ; Add
sub_1297350+15     pop     ebp
sub_1297350+16     retn                    ; Return Near from Procedure
sub_1297350+16     sub_1297350 endp
sub_1297350+16
```

Figure 2 – Decryption Function

During dynamic analysis, it was observed that the malware decrypts the data it keeps encrypted in memory. This is done through a two-parameter function, as also shown in Figure 2. The first parameter is the hexadecimal value 2DA00h, which determines the number of bytes to be decrypted and thus how many times the loop will repeat. The second parameter is the hexadecimal address 135B0F0h, indicating the starting point of the decryption process.

With the specified parameters, the function decrypts a memory block starting from the address 135B0F0h up to the length of 2DA00h, that is, up to the address 1388AF0. As a result, upon decrypting the encrypted data, it has been observed that this data is an executable file.

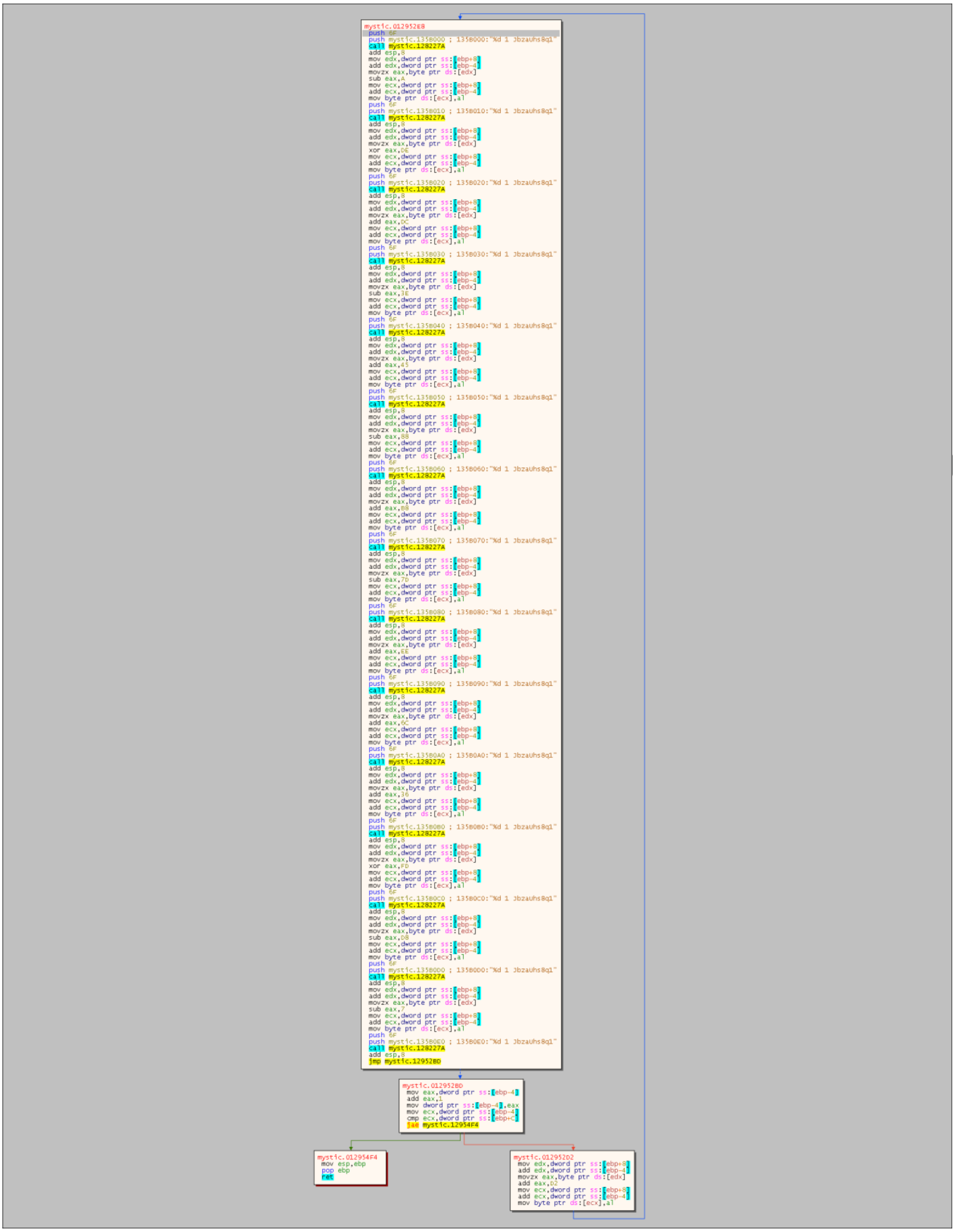


Figure 3 – Algorithm for Decrypting Encrypted Data

The algorithm used to decrypt the encrypted data is as illustrated in Figure 3.

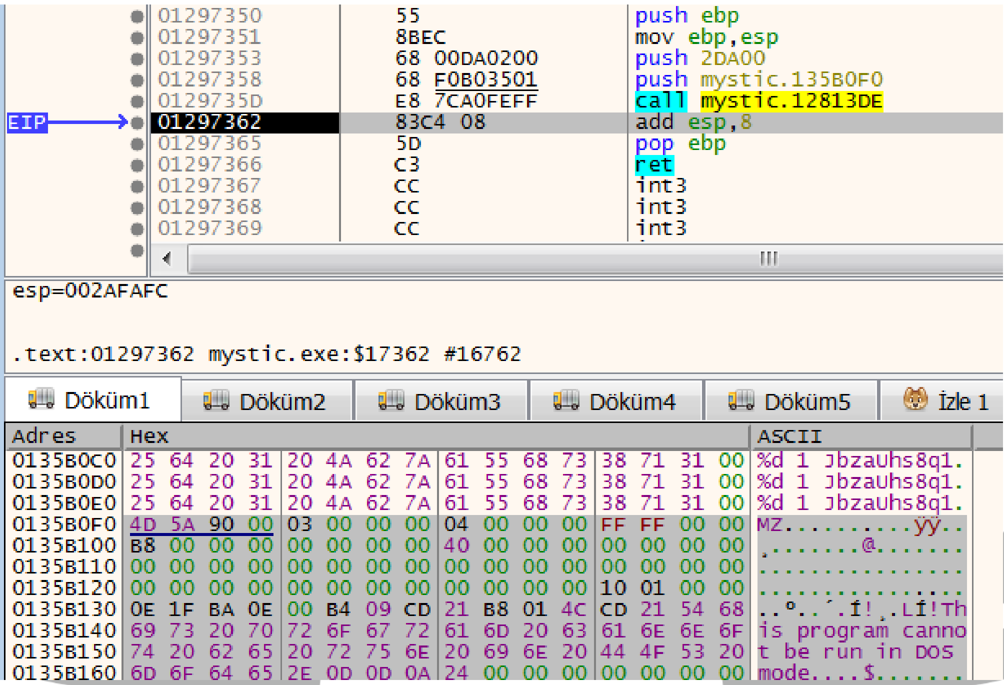


Figure 4 – Decrypted Executable File

The executable file discussed in the previous step of the report, which was decrypted byte by byte, is as shown in Figure 4.

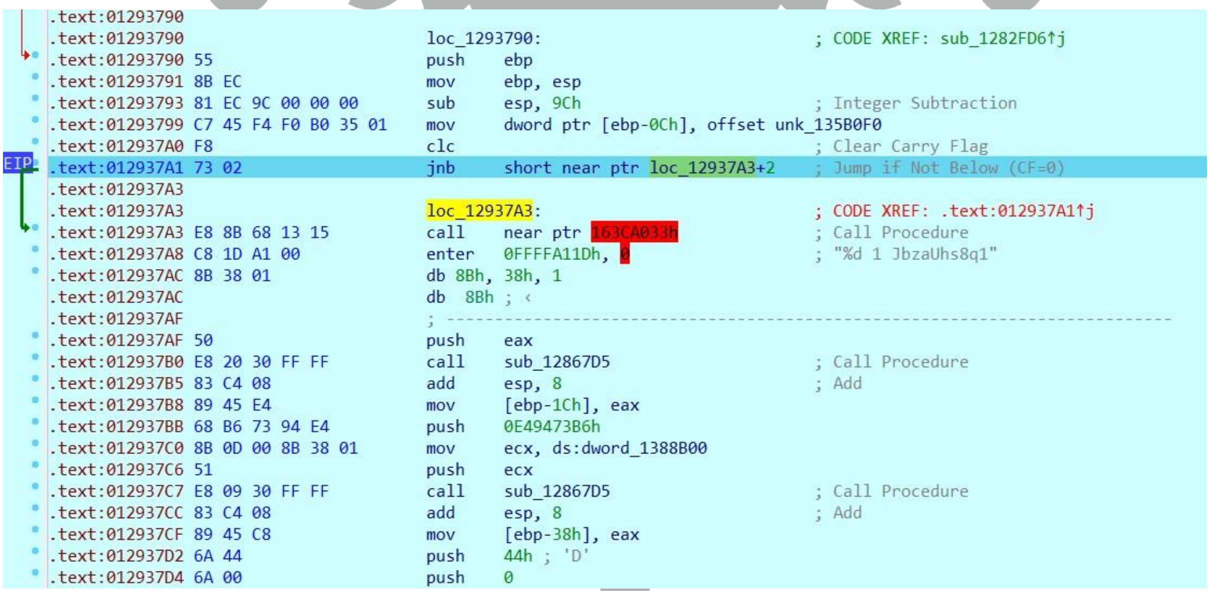


Figure 5 – Anti-disassembly Technique

When the dumped executable file is examined in detail, it was discovered that the malware utilizes an anti-disassembly technique called **Impossible Disassembly**. This technique obstructs the disassembler's ability to interpret the following actual command by inserting data bytes right after a conditional jump instruction. This deception causes the disassembler to misinterpret the real commands, leading to incorrect analysis and erroneous outputs.

```
.text:01293790
.text:01293790                      loc_1293790:                          ; CODE XREF: sub_1282FD6↑j
.text:01293790 55                   push    ebp
.text:01293791 8B EC                mov     ebp, esp
.text:01293793 81 EC 9C 00 00 00    sub     esp, 9Ch                      ; Integer Subtraction
.text:01293799 C7 45 F4 F0 B0 35 01 mov     dword ptr [ebp-0Ch], offset unk_135B0F0
.text:012937A0 F8                   clc                                   ; Clear Carry Flag
.text:012937A1 73 02                jnb     short loc_12937A5             ; Jump if Not Below (CF=0)
.text:012937A1                      ; ---------------------------------------------------------------
.text:012937A3 90                   db 90h
.text:012937A4 90                   db 90h
.text:012937A5                      ; ---------------------------------------------------------------
.text:012937A5
.text:012937A5                      loc_12937A5:                          ; CODE XREF: .text:012937A1↑j
.text:012937A5 68 13 15 C8 1D       push    1DC81513h
.text:012937AA A1 00 8B 38 01       mov     eax, ds:dword_1388B00
.text:012937AF 50                   push    eax
.text:012937B0 E8 20 30 FF FF       call    sub_12867D5                   ; Call Procedure
.text:012937B5 83 C4 08             add     esp, 8                        ; Add
.text:012937B8 89 45 E4             mov     [ebp-1Ch], eax
.text:012937BB 68 B6 73 94 E4       push    0E49473B6h
.text:012937C0 8B 0D 00 8B 38 01    mov     ecx, ds:dword_1388B00
.text:012937C6 51                   push    ecx
.text:012937C7 E8 09 30 FF FF       call    sub_12867D5                   ; Call Procedure
```

Figure 6 – Patched Code

The patching process was carried out to neutralize the technique used by the malicious software. In this patching process, specific opcodes such as E8 and 8B in the malware's code were replaced with the 90 (NOP) opcode. The NOP operation instructs the processor to pass to the next command without making any changes, thereby enabling the disassembler program to correctly read the remainder of the code. As a result of these corrections, the analysis of the malicious software was accurately conducted.

```
      01293790   55                push ebp
      01293791   8BEC              mov ebp,esp
      01293793   81EC 9C000000     sub esp,9C
      01293799   C745 F4 F0B03501  mov dword ptr ss:[ebp-C],mystic.135B0F0
      012937A0   F8                clc
      012937A1   73 02             jae mystic.12937A5
      012937A3   90                nop
      012937A4   90                nop
      012937A5   68 1315C81D       push 1DC81513
      012937AA   A1 008B3801       mov eax,dword ptr ds:[1388B00]
      012937AF   50                push eax
      012937B0   E8 2030FFFF       call mystic.12867D5
      012937B5   83C4 08           add esp,8
      012937B8   8945 E4           mov dword ptr ss:[ebp-1C],eax
      012937BB   68 B67394E4       push E49473B6
      012937C0   8B0D 008B3801     mov ecx,dword ptr ds:[1388B00]
      012937C6   51                push ecx
      012937C7   E8 0930FFFF       call mystic.12867D5
EIP → 012937CC   83C4 08           add esp,8
      012937CF   8945 C8           mov dword ptr ss:[ebp-38],eax

dword ptr [ebp-1C]=[002AFAF0 <&WriteProcessMemory>]=<kernel32.WriteProcessMemory>
eax=<kernel32.CreateProcessW> (7692103D)

.text:012937B8 mystic.exe:$137B8 #12BB8
```

Figure 7 – API Hashing

Throughout the analysis process, it has been determined that the function named "**mystic.12867D5**" uses the API hashing technique to resolve the addresses of critical APIs.

CreateProcessW	WriteProcessMemory	VirtualAllocEx
GetThreadContext	SetThreadContext	ReadProcessMemory
TerminateProcess	ResumeThread	VirtualAlloc

Tablo 1- Some APIs obtained through the API Hashing technique

The table above displays some of the key APIs that have been resolved using the API hashing technique.

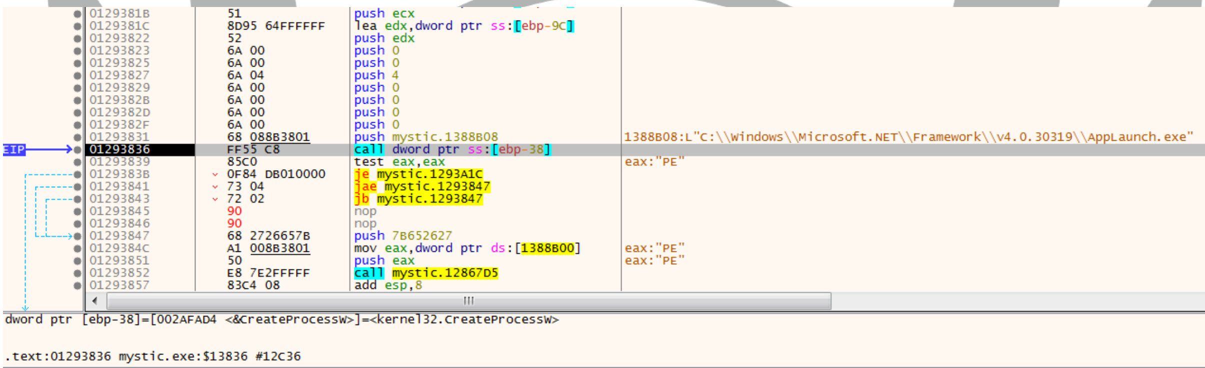


Figure 8 – Use of CreateProcessW and Anti-debug

The call to the **CreateProcessW** API function has been executed. Upon examining the parameters given to the function, it was understood that a legal process was initiated in suspend mode. After the function call, the anti-debug technique was used again.

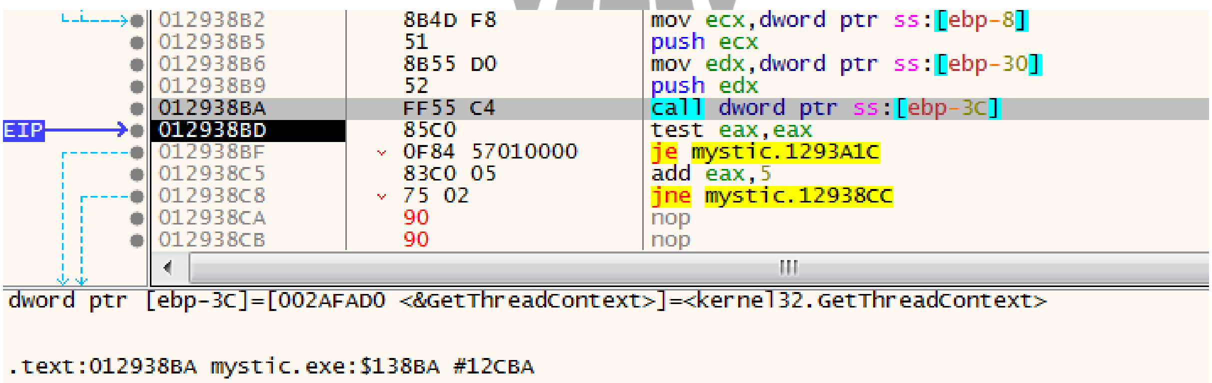


Figure 9 – Use of GetThreadContext and Anti-debug

Continuing the analysis, the malware called the **GetThreadContext** API and employed anti-debug techniques to complicate the analysis process.

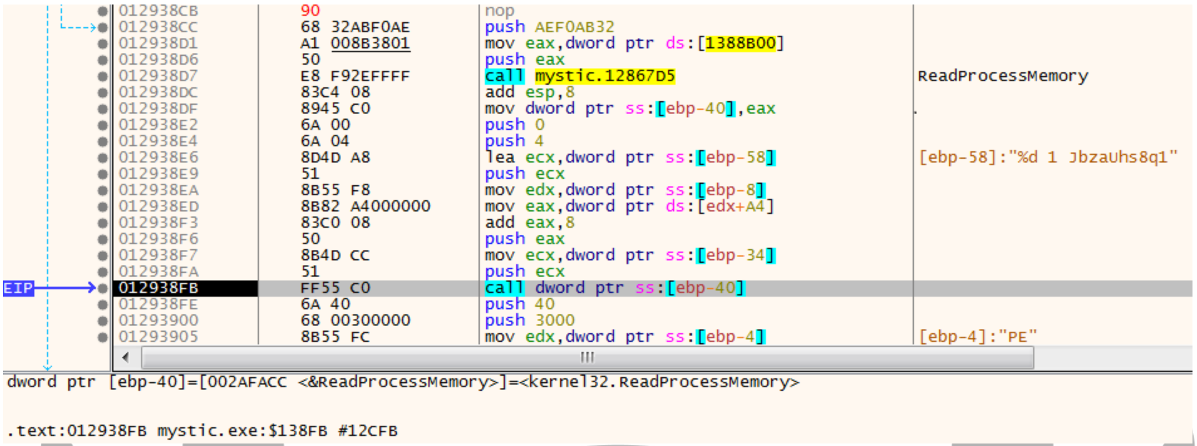


Figure 10 - Use of ReadProcessMemory

Subsequently, the malware called the **ReadProcessMemory** API.

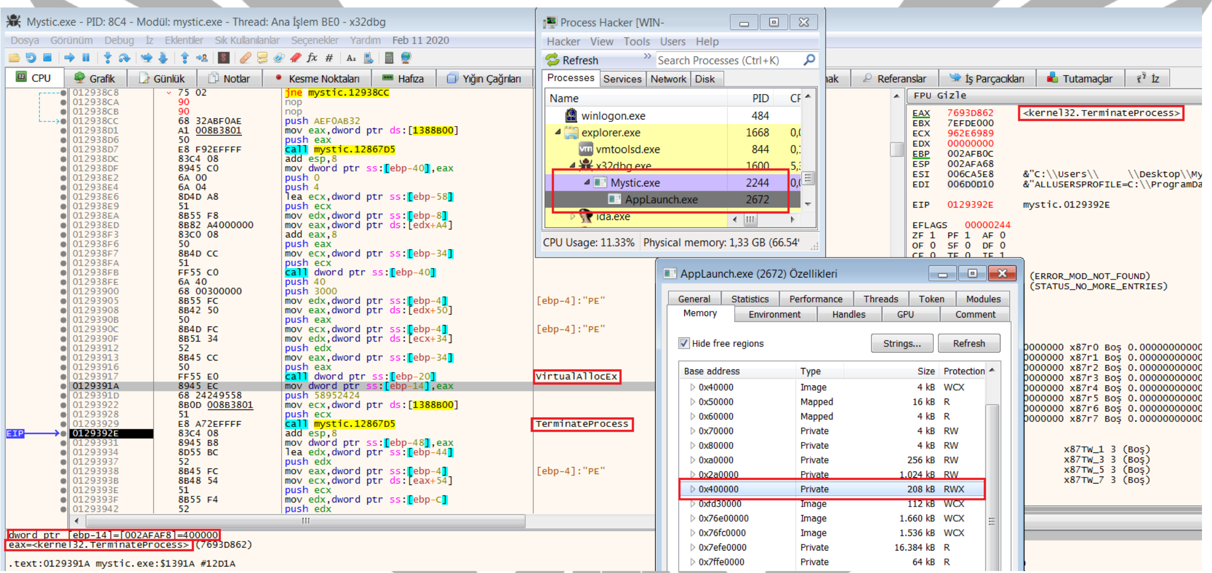


Figure 11 - Memory Allocation with VirtualAllocEx

The malware is allocating space using the **VirtualAllocEx** API in the memory space of an application named AppLaunch, which was previously initiated in suspend mode.

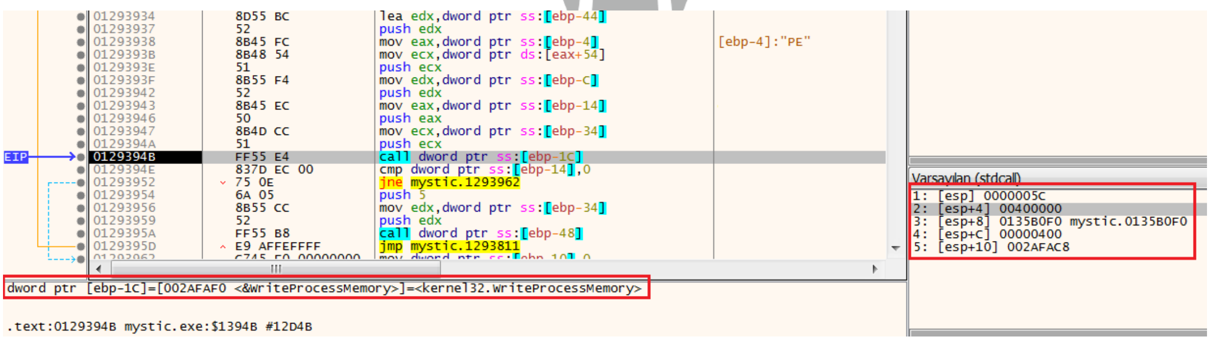


Figure 12 – Use of WriteProcessMemory

Upon the call to the **WriteProcessMemory** API, it was revealed that the malicious software interfered with a legitimate software named "**AppLaunch**" that was started in suspend mode. Using the **VirtualAllocEx** API, the malware wrote the previously decrypted executable file, with a starting address of 0x0135B0F0, to a specially allocated area at address 0x0400000 in the memory space of this application.

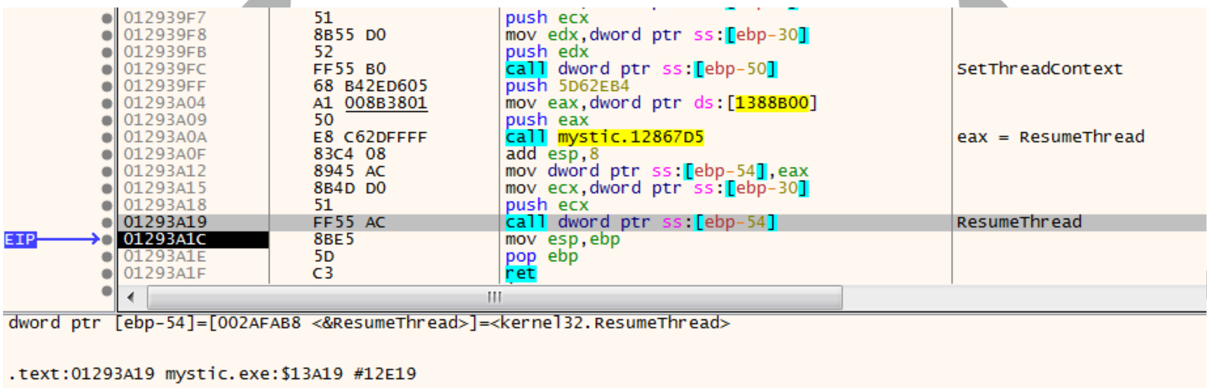


Figure 13 – Use of SetThreadContext and ResumeThread

The analysis has determined that the malware used the **SetThreadContext** API and subsequently used the **ResumeThread** API to successfully continue the suspended process. This indicates the completion of the final stage of the **Process Hollowing** technique by the malware, which involves continuing the process that appears legitimate but is filled with malicious code.

A **dump** of the injected code was taken for detailed analysis, and the examination was continued in the next phase.

Dump.exe Analysis

Name	Dump.exe
MD5	e561df80d8920ae9b152ddddefd13c7c
SHA256	5484ca53027230772ae149e3d7684b7e322432ceb013b6bc2440bd3c269192ea
File Type	Portable Executable 32 (x86)

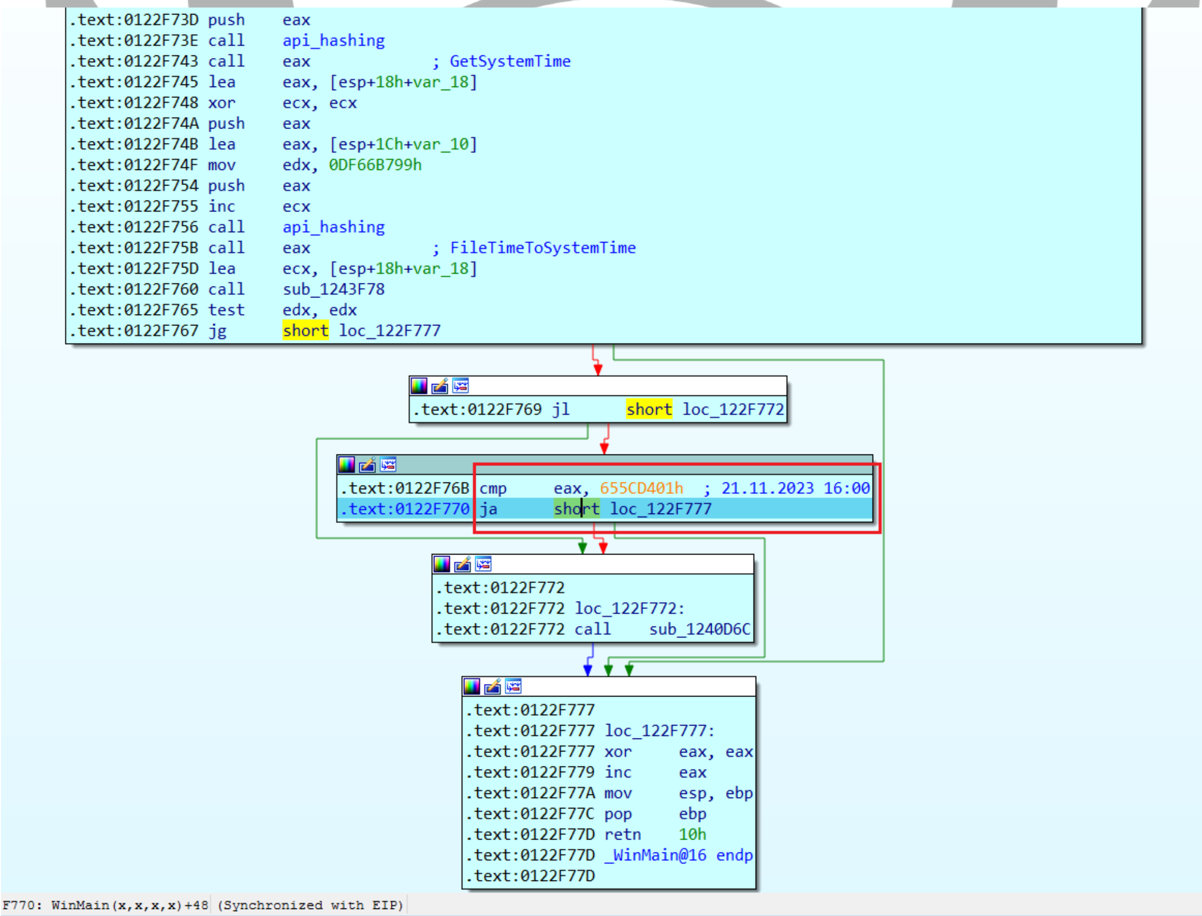


Figure 14 – Date Check

When the detailed examination of the malware's dump began, it was first identified that dynamic API resolving was conducted, and the **GetSystemTime** and **FileTimeToSystemTime** APIs were used in succession. Through these APIs, the malware obtains the current time information on the system. According to the analysis, the malware has a basic time control mechanism and checks if the system time is before November 21, 2023, 16:00. If the system time has passed this date, the malware shuts itself down.

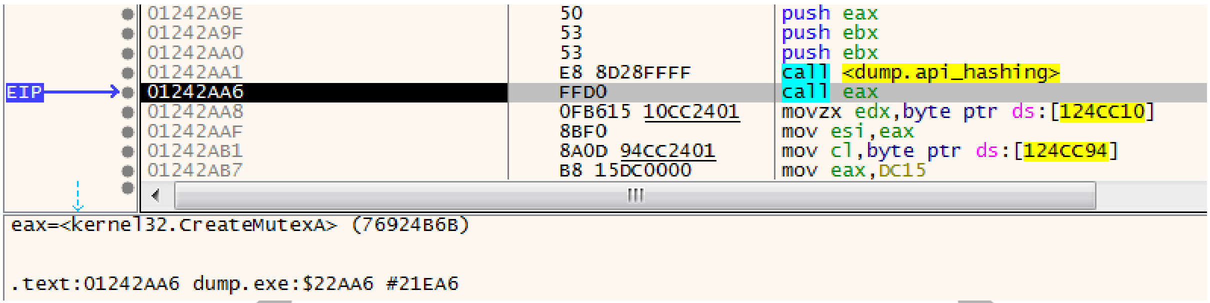


Figure 15 – Mutex Creation

The malware then attempts to create a mutex using the **CreateMutexA** API; if this fails or if the relevant mutex already exists, it shuts itself down.

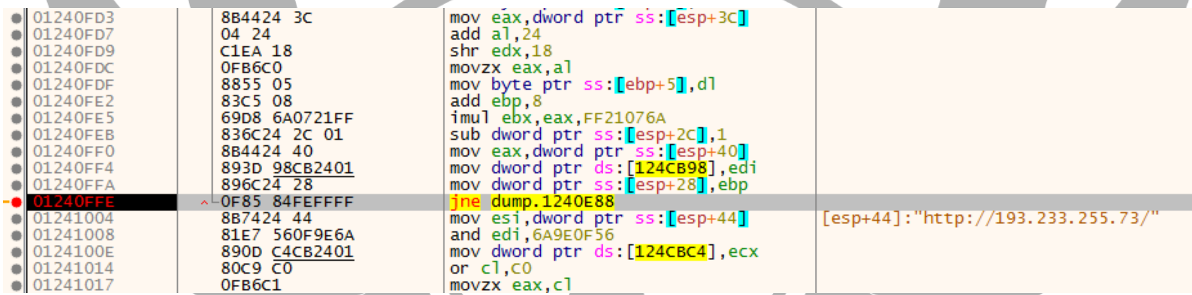


Figure 16 – IP Address Resolution

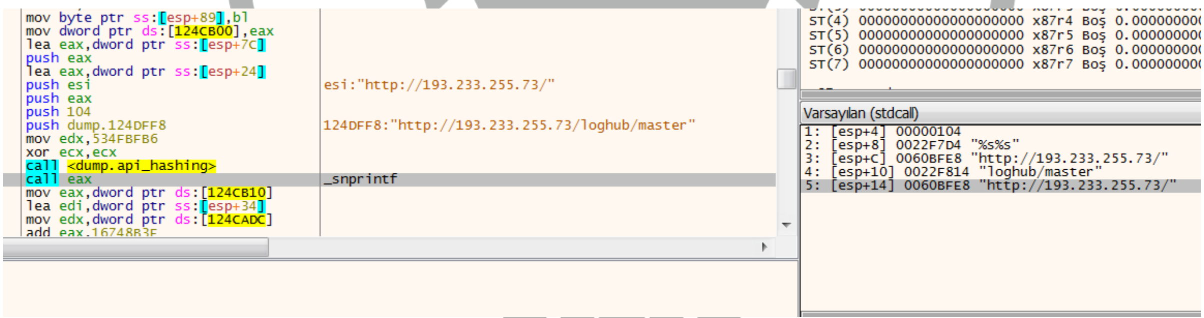


Figure 17 – Address Information

The malware decrypts the encrypted IP address and path information, creating the address **"193[.]233[.]255[.]73/master/login"** with the **_snprintf** function.

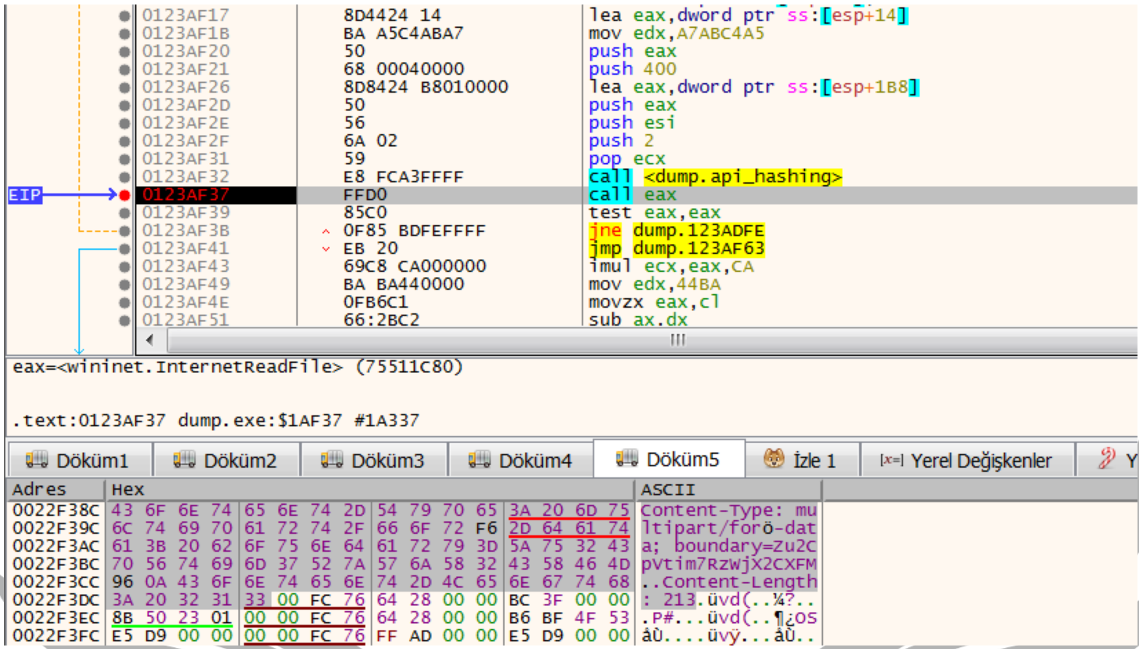
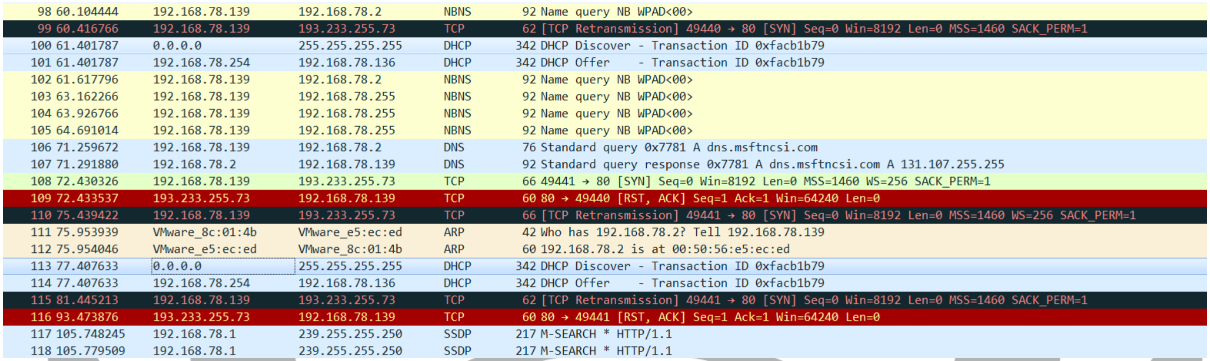


Figure 18 – InternetReadFile API

The malware uses a specific API chain in the process of communicating over the internet. Initially, it starts a general internet connection with **InternetOpenA**. Then, it utilizes **InternetCrackUrlA** for URL parsing and **InternetConnectA** to establish a connection to a specific server. Once the connection is made, an HTTP request is created with **HttpOpenRequestA**. The properties of the request are set by calling **InternetSetOptionA** three times consecutively. After these settings, the prepared HTTP request is sent with **HttpSendRequestA**. Following the request, the response from the server is queried using **HttpQueryInfoA**, and the content of the response is read with **InternetReadFile**. However, since the server is closed, the call to **HttpSendRequestA** fails, which prevents the malware from activating and continuing its operations, thus leading it to shut down. This behavior demonstrates the necessity for the malware to establish successful communication with external servers to activate and maintain its operations.

Network Analysis



98 60.104444	192.168.78.139	192.168.78.2	NBNS	92 Name query NB WPAD<00>
99 60.416766	192.168.78.139	193.233.255.73	TCP	62 [TCP Retransmission] 49440 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
100 61.401787	0.0.0.0	255.255.255.255	DHCP	342 DHCP Discover - Transaction ID 0xfacb1b79
101 61.401787	192.168.78.254	192.168.78.136	DHCP	342 DHCP Offer - Transaction ID 0xfacb1b79
102 61.617796	192.168.78.139	192.168.78.2	NBNS	92 Name query NB WPAD<00>
103 63.162266	192.168.78.139	192.168.78.255	NBNS	92 Name query NB WPAD<00>
104 63.926766	192.168.78.139	192.168.78.255	NBNS	92 Name query NB WPAD<00>
105 64.691014	192.168.78.139	192.168.78.255	NBNS	92 Name query NB WPAD<00>
106 71.259672	192.168.78.139	192.168.78.2	DNS	76 Standard query 0x7781 A dns.msftncsi.com
107 71.291880	192.168.78.2	192.168.78.139	DNS	92 Standard query response 0x7781 A dns.msftncsi.com A 131.107.255.255
108 72.430326	192.168.78.139	193.233.255.73	TCP	66 49441 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
109 72.433537	193.233.255.73	192.168.78.139	TCP	60 80 → 49440 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
110 75.439422	192.168.78.139	193.233.255.73	TCP	66 [TCP Retransmission] 49441 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
111 75.953939	VMware_8c:01:4b	VMware_e5:ec:ed	ARP	42 Who has 192.168.78.2? Tell 192.168.78.139
112 75.954046	VMware_e5:ec:ed	VMware_8c:01:4b	ARP	60 192.168.78.2 is at 00:50:56:e5:ec:ed
113 77.407633	0.0.0.0	255.255.255.255	DHCP	342 DHCP Discover - Transaction ID 0xfacb1b79
114 77.407633	192.168.78.254	192.168.78.136	DHCP	342 DHCP Offer - Transaction ID 0xfacb1b79
115 81.445213	192.168.78.139	193.233.255.73	TCP	62 [TCP Retransmission] 49441 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
116 93.473876	193.233.255.73	192.168.78.139	TCP	60 80 → 49441 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
117 105.748245	192.168.78.1	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
118 105.779509	192.168.78.1	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1

Figure 19 – Wireshark Image

The interaction of the malware with the internet is as shown in the screenshot taken from Wireshark in Figure 19.

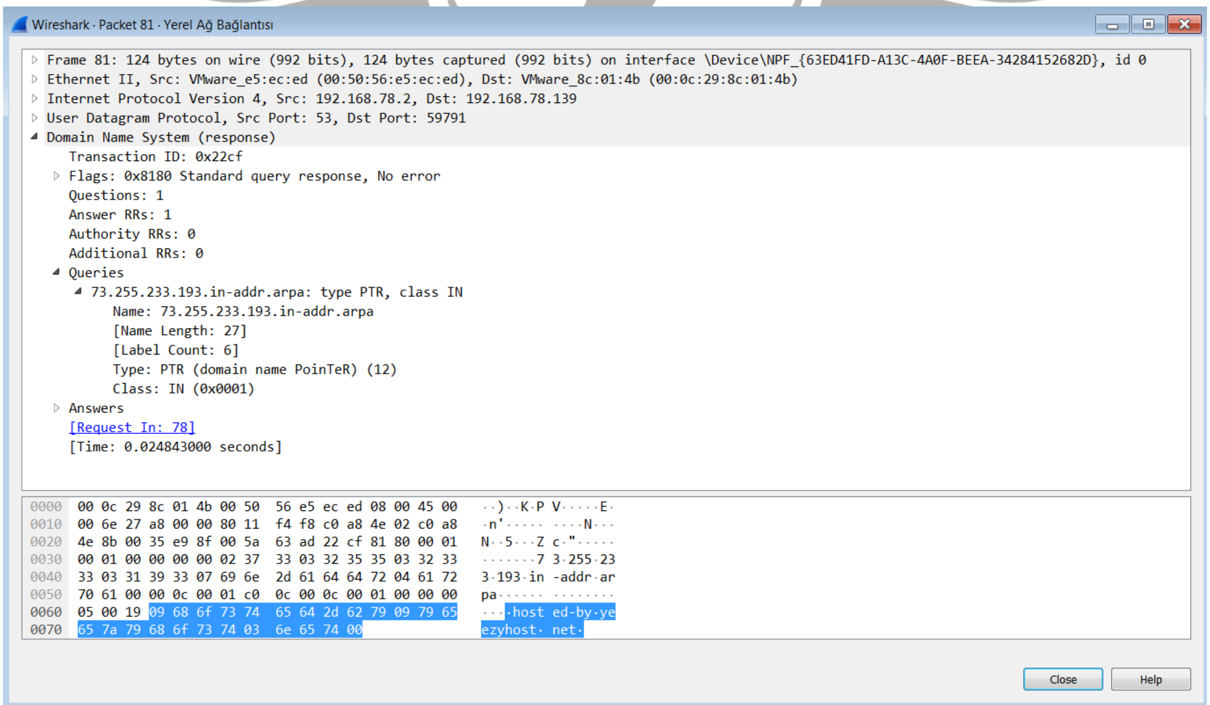


Figure 20 – DNS Request

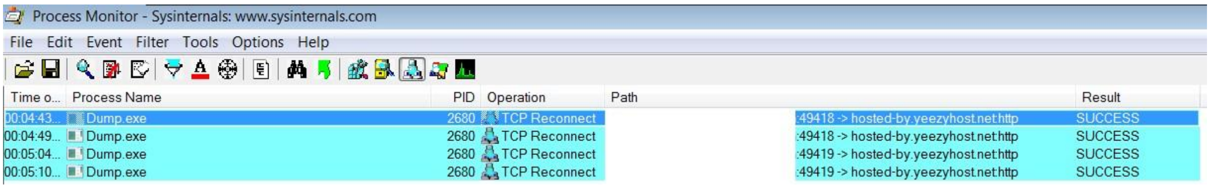
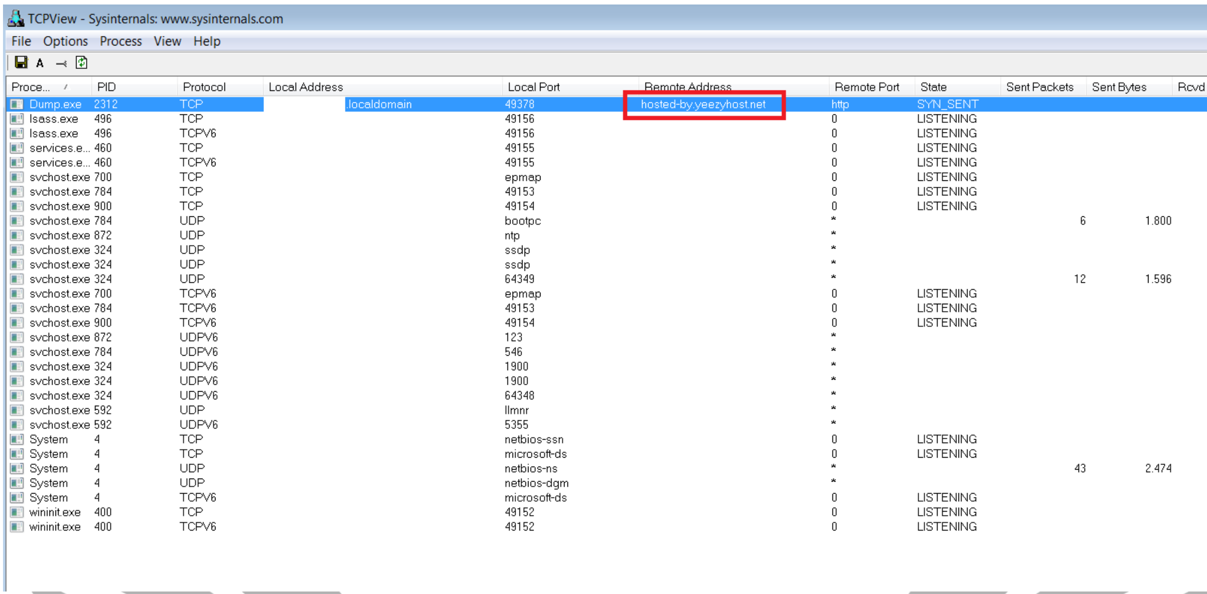


Figure 21 – Process Monitor View



TCPView - Sysinternals: www.sysinternals.com

File Options Process View Help

Proce...	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port	State	Sent Packets	Sent Bytes	Rcvd
Dump.exe	2312	TCP	.localdomain	49378	hosted-by.yeezyhost.net	http	SYN_SENT			
lsass.exe	496	TCP		49156		0	LISTENING			
lsass.exe	496	TCPV6		49156		0	LISTENING			
services.e...	460	TCP		49155		0	LISTENING			
services.e...	460	TCPV6		49155		0	LISTENING			
svchost.exe	700	TCP		epmap		0	LISTENING			
svchost.exe	784	TCP		49153		0	LISTENING			
svchost.exe	900	TCP		49154		0	LISTENING			
svchost.exe	784	UDP		bootpc		*		6	1.800	
svchost.exe	872	UDP		ntp		*				
svchost.exe	324	UDP		ssdp		*				
svchost.exe	324	UDP		ssdp		*				
svchost.exe	324	UDP		64349		*		12	1.596	
svchost.exe	700	TCPV6		epmap		0	LISTENING			
svchost.exe	784	TCPV6		49153		0	LISTENING			
svchost.exe	900	TCPV6		49154		0	LISTENING			
svchost.exe	872	UDPV6		123		*				
svchost.exe	784	UDPV6		546		*				
svchost.exe	324	UDPV6		1900		*				
svchost.exe	324	UDPV6		1900		*				
svchost.exe	324	UDPV6		64348		*				
svchost.exe	592	UDP		llmnr		*				
svchost.exe	592	UDPV6		5355		*				
System	4	TCP		netbios-ssn		0	LISTENING			
System	4	TCP		microsoft-ds		0	LISTENING			
System	4	UDP		netbios-ns		*		43	2.474	
System	4	UDP		netbios-dgm		*				
System	4	TCPV6		microsoft-ds		0	LISTENING			
wininit.exe	400	TCP		49152		0	LISTENING			
wininit.exe	400	TCPV6		49152		0	LISTENING			

Figure 22 – TCPView View

YARA Rules

```
import "hash"

rule MysticStealer_s

{

    meta:

        author = " Barış Tural"

        description = "MysticStealer"

    strings:

        $str1 = "%d 1 JbzaUhs8q1"

        $str2 =
"C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe"
wide

        $str3 = "GYAUs87atedyuw3"

        $str4 = "A8791hbx78iUA"

        $hashing_alg = {D1 E2 8B 45 E8 0F BE 0C 10 C1 E1 10 33 4D F8
89 4D F8}

        $anti_disass1 = {F8 73 02 E8 8B 68 13 15 C8 1D A1 ?? ?? ?? ?? 50
E8 20 30 FF FF}

        $anti_disass2 = {73 04 72 02 E8 A0 68 27 26 65 7B A1}

        $anti_disass3 = {E0 72 04 73 02 E9 9F 6A 04}

    condition:

        all of them or hash.md5(0,filesize) ==
"692a59e85b4c932049ab55cb372a9509" or 3 of ($str*) and 2 of
($anti_disass*) or $hashing_alg
```

```
import "hash"

rule MysticStealer_d

{

    meta:

        author = " Barış Tural"

        description = "MysticStealer"

    strings:

        $str1 = "HH:mm:ss" wide

        $str2 = "morda"

        $str3 =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

        $alg = {33 C8 2B D9 8B CB 8B C3 C1 E1 04 C1 E8 05}

        $ip_resolv = {80 C9 18 89 74 24 44 88 0D ?? ?? ?? ?? B8 F5 2A 00
00 0F B6 C9 66 33 C8 B8 D1 52 00 00 66 2B C8}

    condition:

        hash.md5(0,filesize) == "e561df80d8920ae9b152ddddefd13c7c" or
(2 of ($str*) and ($alg or $ip_resolv))

}
```

MITRE ATTACK TABLE

Reconnaissance	Execution	Discovery	Privilege Escalation	Defense Evasion	Credential Access	C&C	Exfliration
		System Information Discovery (T1082)	Process Injection (T1055)	Obfuscated Files or Information (T1027)		Application Layer Protocol (T1071)	
				Process Injection (T1055)			
				Deobfuscate /Decode Files or Information			

Recommendations

1. Continuously monitor known IP addresses and domains communicated by Mystic Stealer and block access to these addresses.
2. Inform users about the propagation methods of malware like Mystic Stealer. Warn them to be vigilant against phishing attacks and malicious email attachments.
3. Enable two-factor authentication (2FA) wherever possible.
4. Ensure that the antivirus and malware detection tools in use are kept up-to-date at all times.
5. Restrict access to systems and remove unnecessarily high permissions for user accounts. Add extra layers of security for access to sensitive applications like cryptocurrency wallets.

PREPARED BY

Barış TURAL https://www.linkedin.com/in/baristural/