

IMT4116 - Mandatory Assignment 1

Ole André Hauge

April 26, 2022

For this assignment, we are asked to analyze *assignment2.exe* using basic static analysis and basic dynamic analysis to produce a hypothesis. In order to do that we use an analysis environment consisting of 2 virtual machines (VM). A Windows 7 operating system and several tools for both dynamic and static malware analysis are installed on one virtual machine. Remnux, a Linux tool suite, is installed on the other VM. Although it has many tools, we will mainly use it for internet simulation and packet capturing. To achieve this the virtual machines are configured in a LAN together and are otherwise not connected to any other devices or the Internet. The paper is divided into two sections: Basic Static Analysis and Basic Dynamic Analysis.

1 Basic Static Analysis

In this section, we apply basic static analysis techniques to find probable indicators of compromise (IoC), which are then used to create a hypothesis regarding *assignment2.exe*'s potential purpose/functionality. We focused on strings and imported functions, hoping to find answers to the following questions:

- What is it?
 - Is it a PE file?
 - How is it compiled?
- Are there any easily identifiable traits or features?
 - Interesting characteristics in sections, imports, or strings?
 - Is it signed, and is the signer trusted?
- Do we believe this needs more investigation?

1.1 Tools

For the basic static analysis and dual-tool verification, we relied on Exeinfo PE, PEiD, PeStudio 8.87, and CFF explorer VIII.

1.2 Indicators of Compromise

We open *assignment2.exe* in Exeinfo PE to get a quick overview of what we're looking at. As shown in Figure 1, it recognizes the PE-file as *assignment2.exe* and that it is a 32bit executable with a timestamp of 2003. We can also see that it is 43,35KB in size and has an overlay of 184 bytes. Such overlays can be used to conceal malicious code because they are ignored when the executable is loaded into memory. If the malware gains access to the system, it will be able to execute the data stored in the overlay because it already knows where to look for it. Exeinfo states that it does not appear to be packed, so we should be able to examine the file's contents with other tools as well. It recognizes the file's signature as *LCC_Win32_v1.x*, which appears suspicious given that a quick Google search (without entering links) yields a variety of previous malware analyses and legitimate results. PEiD is used for dual-tool data verification. The only difference was that PEiD did not provide any timestamp information.

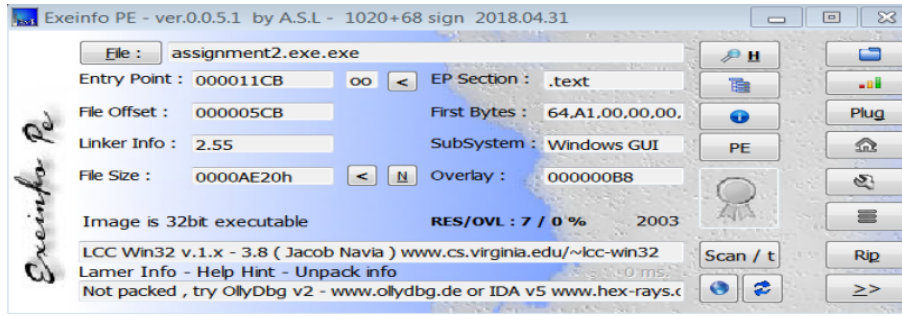


Figure 1: Exeinfo PE: Result

As a result, we continued to check the PE file with PeStudio 8.87 to gain access to more data. This tool confirms our previous findings and provides additional information about the compiler stamp of April 17, 2003, and the resource stamp of May 2, 1996. The compiler stamp matches the stamps found with the previous tools, but the resource stamp appears to have a strange discrepancy from the compiler stamp. When we use CFF Explorer VIII for dual-tool verification, we see that the file was created, modified, and accessed on January 8, 2014. What is more, it identifies a possible internal and original name for the file, *wuaumgr.exe*, as shown in 2. This could indicate that the file contains another executable file.

Furthermore, there is a discrepancy in the headers. The following section headers are common: *.text*, *.rdata*, *.data*, *.rsrc*, and *.reloc* [1, p. 21-22]. Looking at *assignment2.exe*, we can see that *.text*, *.data*, and *.rsrc* are present, but *.bss* and *.rdarta* are possibly hidden, and *.idata* is added. This could be an IoC of program modification.

We can also confirm that the program was compiled with Microsoft Visual C++ 3.0 by inspecting it with CFF explorer, highlighting the importance of dual-tool verification. PeStudio and CFF Explorer are used for dual-tool verification throughout the rest of the section.

Property	Value
CompanyName	Microsoft Corporation
FileDescription	Generic Host Process for Win32 Services
FileVersion	5.1.2700.0 (NT client.010817-1148)
InternalName	wuaumgr.exe
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	wuaumgr.exe
ProductName	Microsoft® Windows® Operating System

Figure 2: CFF Explorer VIII: File data

1.2.1 Libraries

There are a couple of interesting libraries listed for the program, and although *kernel32.dll* and *user32.dll* are fairly common, it is interesting to see the high amount of imports.

library (7)	blacklist (2)	missing (...)	type (1)	imports (110)	file-description
wsock32.dll	×	-	implicit	23	Windows Socket 32-Bit DLL
winmm.dll	×	-	implicit	1	MCI API DLL
shell32.dll	-	-	implicit	1	Windows Shell Common Dll
kernel32.dll	-	-	implicit	46	Windows NT BASE API Client DLL
user32.dll	-	-	implicit	9	Multi-User Windows USER API Client DLL
advapi32.dll	-	-	implicit	7	Advanced Windows 32 Base API
crt.dll	-	-	implicit	23	Microsoft C Runtime Library

Figure 3: PeStudio: Library Results

The *Windows Sockets API*, which is used by most Internet and network applications to handle network connections, is contained in *sock32.dll*. Based on the large number of imports, the program may attempt to connect to a remote Internet server. The library *winmm.dll* is used to control and communicate with multimedia devices. The imports from *Shell32.dll* indicate that the program has the ability to launch other programs, which is a feature shared by both malware and legitimate programs. The software can open and manipulate processes and files thanks to *Kernel32.dll*. *User32.dll* can be used to import functions for interacting with the GUI. What is also interesting is the imports from *Advapi32.dll* indicate that the program can access the registry, which leads us to believe that we should look for strings that look like registry keys. After doing some research, we discovered that *crtdll.dll* is used by C++ compiled programs for multi-threading support, so we might see some strings or behavior related to that later on [1, p. 17], [2].

1.2.2 Imported functions

We begin by inspecting the functions imported from the *kernel32* library. The first IoC we notice is functions that allow the program to search directories and move files:

- `GetModuleFileNameA`: Retrieves the fully qualified path for the file that contains the specified module. The module must have been loaded by the current process [2]¹.
- `FindClose`, `FindFirstFileA`, `FindNextFileA`, `MoveFileA`, `SetFileAttributesA`, `DeleteFileA`

This is a common IoC for malware that could be ransomware looking for files to encrypt. It could, however, also be a keylogger attempting to capture and store data, or malware, in general, attempting to move files to folders. Following that, we see that functions for creating processes and threads are imported, which could be IoCs for malware that wants to start other programs. Some of the most interesting functions are listed below, along with some additional information about them [2]²

- `GetCurrentProcess`: Retrieves a pseudo handle for the current process.
- `GetExitCodeProcess`: Retrieves the termination status of the specified process.
- `OpenProcess`: Opens an existing local process object.
- `TerminateProcess`: Terminates the specified process and all of its threads.
- `TerminateThread`: Terminates a thread.
- `CreateProcessA`: Creates a new process and its primary thread. The new process runs in the security context of the calling process.
- `CreateThread`: Creates a thread to execute within the virtual address space of the calling process.

The imports from the *user32* library are even more interesting, as we can see that several functions related to key-mapping and keyboard events, as well as additional functions for collecting information about foreground windows, are imported [2]³.

- `GetKeyState`: Retrieves the status of the specified virtual key. The status specifies whether the key is up, down, or toggled.
- `GetAsyncKeyState`: Determines whether a key is up or down at the time the function is called and whether the key was pressed after a previous call to `GetAsyncKeyState`.
- `MapVirtualKeyA`: Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

This appears to be an IoC of a local keylogger that records keystrokes using functions imported from *user32.dll* and may be able to record graphical movement, such as window switching.

¹The information is added for educational purposes.

²The information is added for educational purposes.

³The information is added for educational purposes.

The *shell32* library imports only one function, *ShellExecuteA*, which allows it to perform operations on a given file. This could mean that the malware is attempting to run commands from the command shell on the localhost. The ability of the program to create processes and access the shell may also indicate the use of reverse shell, as malware frequently uses *CreateProcess* to create a simple remote shell with a single function call. This function could be used by malware to set values to a socket by piping data from standard output to the socket. We also see the possibility of employing threads. The malware may use two threads, one for input and one for output, effectively piping standard output and input to a socket so that the actor can communicate with the running application [1, p. 148-149].

We find functions (*WSAStartup*, *listen*, *recv*, *send*, *WSAGetLastError*, *accept*) related to opening and closing sockets for communication with another server in the *wsock32* library imports. The program is most likely attempting to find an IP address by using functions in the file's strings (*gethostbyname*, *gethostbyaddr*, *getsockname*, *htonl*, *ntohs*). At this point, we don't know where or what the IP address is, but we might see it among the strings later [1, p. 313].

More IoCs are discovered in the functions imported from the *advapi32* library, indicating that the program appears to attempt to edit registry key values.

- *RegCreateKeyA*: Creates the specified registry key. If the key already exists in the registry, the function opens it.
- *RegSetValueExA*: Sets the data and type of a specified value under a registry key.

Malware frequently attempts to edit registry values such as the *Software\...\CurrentVersion\Run*, which is a registry key that controls which programs are automatically launched when Windows boots up. This allows the malware to gain persistence on the system, meaning that it will not be removed by a reboot [1, p. 20].

Finally, a few of other functions that appear suspicious but whose likely applications we are still unsure of, along with some further information about the functions [2]⁴:

- *GlobalMemoryStatus*: Retrieves information about the system's current usage of both physical and virtual memory.
- *rand*, *srand*: Generates a pseudorandom number.
- *mciSendStringA*: sends a command string to an MCI device.
- *DuplicateHandle*: Duplicates an object handle

1.2.3 Strings

There are many strings that look interesting, among them are strings that look like command line commands (*opencmd*, *c:%s.exe*, *makedir*, *reboot*, *disconnect*). We have done our best to include the most interesting strings that could imply compromises.

Looking at the strings we can observe that there is one string for another library, *MPR.DLL*, which is not listed among the other libraries. This is a library that can be used to try to obtain more information about the network of the host machine, further supporting our indications that the program will attempt to access the Internet.

Furthermore, we can identify several executable filenames:

- *EDU_Hack.exe*
- *Sitebot.exe*
- *Winamp_Installer.exe*
- *PlanetSide.exe*
- *DreamweaverMX_Crack.exe*

⁴The information is added for educational purposes.

- FlashFXP_Crack.exe
- FlashFXP_Crack.exe
- Postal_2_Crack.exe
- Red_Faction_2_No-CD_Crack.exe
- Renegade_No-CD_Crack.exe
- Generals_No-CD_Crack.exe
- Norton_Anti-Virus_2002_Crack.exe
- Porn.exe
- AVP_Crack.exe
- zoneallarm_pro_crack.exe
- wuaumgr.exe
- NETSTAT.EXE
- TASKMGR.EXE
- MSCONFIG.EXE
- REGEDIT.EXE

This could be an IoC of a program that runs another program, such as *EDU_Hack.exe*, and/or other commands in a command shell. A quick internet search of the executables reveals that many of the executables discovered as strings appear to be cracked versions of games, software, and antivirus software from the years 2002-2004. The timestamps we saw earlier correlate to this timeline. There is also a misspelling in the *zoneallarm_pro_crack.exe* file, which could imply that the user downloaded the wrong file by accident. As these appear to be cracked programs, it is possible that harmful malware was hidden in the executables. A *Porn.exe* program is also there, which is unusual as porn often are stored as multimedia files. The software could use the final five listed programs to identify the network, run Task Manager, and edit registries.

What is intriguing is that if you were active on the internet in the early 2000's one might recognize *Kazaa* as a peer-to-peer torrent site that was used for downloading all sorts of files. It was also widely known that using the service included a high chance of downloading harmful software. Given the number of games and cracked files, this malware sample could be a collection of pirated games and software downloaded from the site in the past.

We can identify that the program have strings related to all special keys on a keyboard: *[Num Lock]*, *[Down]*, *[Right]*, *[Up]*, *[Left]*, *[End]*, *[Del]*, *[Home]*, *[Scroll Lock]*, *[Print Screen]*, *[WIN]*, *[Insert]*, *[CTRL]*, *[TAB]*, *[F12]*, *[F11]*, *[F10]*, *[F9]*, *[F8]*, *[F7]*, *[F6]*, *[F5]*, *[F4]*, *[F3]*, *[F2]*, *[F1]*, *[ESC]*, *[Pg Dn]*, and *[Pg Up]*. This could be an IoC of a key-logger as it is often not possible to record special keys and would need to describe the pressed key in a textual manner.

We can identify two IP-addresses in terms of the IoCs of the software attempting to connect to a server:

- 209.126.201.22
- 209.126.201.20

The strings also indicate several hard-coded responses in the program that may be used to update the user on its progress. Among these are strings that, as seen in Figure 4, specify the state of the operating keylogger in plaintext.

ascii	4	-	-	QUIT
ascii	16	-	-	Keylogger stoped
ascii	23	-	-	Keylogger logging to %s
ascii	36	-	-	Keylogger active output to: DCC chat
ascii	30	-	-	Keylogger active output to: %s
ascii	60	-	-	error already logging keys to %s use "stopkeylogger" to stop

Figure 4: PeStudio: String Result - text revealing potential keylogger activity

We can further observe several strings related to socket and data transfer activity in Figure 5:

ascii	14	-	-	PRIVMSG %s :%s
ascii	23	-	-	Process has terminated
ascii	33	-	-	Could not read data from process
ascii	7	-	-	SFT05%i
ascii	22	-	-	Searsing for passwords
ascii	23	-	-	PWD14438136782715101980
ascii	6	-	-	PWD715
ascii	38	-	-	Server uploaded to kuangserver IP: %s
ascii	50	-	-	PRIVMSG %s :Server uploaded to kuangserver IP: %s
ascii	45	-	-	Server uploaded to sub7server IP: %s port: %i
ascii	57	-	-	PRIVMSG %s :Server uploaded to sub7server IP: %s port: %i
ascii	29	-	-	Found poort %i open at ip:%s
ascii	41	-	-	PRIVMSG %s :Found poort %i open at ip:%s
ascii	15	-	-	HTTP/1.0 200 OK
ascii	17	-	-	Server: SpyBot1.2
ascii	48	-	-	HTTP server listing on poort: %i root dir: %s\

Figure 5: PeStudio: String Result - text revealing potential transmission and keylogger activity

We may deduce from these strings that the application is searching for network information and the IP addresses given above, as well as attempting to transfer data to the *kuangserver*, *sub7server*, or *SpyBot1.2* servers over some port. The strings also indicate that the application is listening for or sending data on a certain port and that the program may use *PRIVMSG* to inform the actor of the transmission status. Furthermore, we can see indications of collections of passwords that could be in line with a keylogger. However, this could also indicate ransomware with predefined passwords for encipherment, but that would be less likely and not so common, especially as these are short passwords in plaintext.

There are strings indicating that the program is looking for and trying to access files (*File doesn't exists*), as well as processes. We also discover IoCs, which add to our confidence in the program including a keylogger with the string: *%s (Changed window*. This appears to be data that may be written to a file to notify the computer system of a window change done by an unsuspecting user.

We can also see IoCs that could be related to the program attempting to access and modify the registry, as well as attempting to disguise file alteration by changing the last-modified dates.

- Date: %s %s GMT
- Last-Modified: %s %s GMT

Several strings indicate that the system could be used in a syn flooding attack, which could indicate that the machine is involved in multiple malicious behaviors or that it is part of a botnet. This could be related to the *SpyBot1.2* server.

Finally, the presence of multiple misspellings in the strings is a clear indicator that the program was developed by someone who does not speak English natively. A less likely explanation is that it was written in a hurry or by someone with poor grammar skills.

1.3 Preliminary Hypothesis

This appears to be malicious software capable of connecting with external servers using the IP addresses 209.126.201.20 and 209.126.201.22. These addresses could resolve to one of three servers: *kuangserver*, *sub7server*, or *SpyBot1.2*. It appears to build persistence by altering registry entries, and it may use *cmd.exe* and *privmsg* to create a shell with C2 functionality. We also believe it may install 14 applications (cracked executables) inside *SOFTWARE\KAZAA\LocalContent*, and/or hide them as *kazaabackupfiles* in the same or another directory. It appears to be capable of launching a keylogger that records keystrokes and GUI operations, as well as listening for passwords and where they are used (email).

We can further speculate that the recorded data is saved in a text file named *keylog.txt* and that this data is sent to the remote server along with C2 update responses. There are additional hints of the ability to perform a synflood attack and port scanning, leading us to believe that it is not just a keylogger but might be a type of rootkit containing multiple dangerous programs. However, the majority of the IoCs found indicate that this is a keylogger.

In short, *likely* a local keylogger communicating with remote servers, *unlikely* a rootkit with a software suite. Both with some sort of persistence on the system.

This program needs further analysis.

2 Basic Dynamic Analysis

In this section, we apply the five-step basic dynamic analysis process to Figure out what the malware sample is doing to our system. The data are used to confirm or refute our preliminary hypothesis from the basic static analysis section.

2.1 Tools

On the Windows VM, we used Process Hacker to keep track of the processes started by the malware when it was executed. Regshot was used to discover changes to the file system including the registry, and Procmon was used to capture the registry, process, and thread activity. On the Remnux VM, we used Inetsim to simulate the network responses that the malware might listen for, and Wireshark to capture the network traffic.

2.2 Using Regshot

We can see that 18 files have been added to the system using Regshot. The first two files are disregarded because they are part of normal system functionality. As shown in Figure 6, the malware adds 16 files and one folder. We notice that the files we assumed would be saved on the system are instead saved in the newly created folder. In addition, *keylog.txt* and *wuauqmqr.exe* are added to the *System32* directory. These files are also the same as the ones mentioned in the strings. *keylog.txt* is likely the file that the malware writes to, and *wuauqmqr.exe* is probably the first program that is executed by the malware. Because it is placed in *System32*, the malware is most likely attempting to hide its presence, and it might be used to execute after a system reboot if the malware has persistence.

```

-----
Files added: 18
-----
C:\ProgramData\Microsoft\Windows Defender\Scans\History\Service\Detection.log
C:\Users\All Users\Microsoft\Windows Defender\Scans\History\Service\Detection.log
C:\Windows\System32\kazaabackupfiles\AVP_Crack.exe
C:\Windows\System32\kazaabackupfiles\DreamweaverMX_Crack.exe
C:\Windows\System32\kazaabackupfiles\EDU_Hack.exe
C:\Windows\System32\kazaabackupfiles\FlashFXP_Crack.exe
C:\Windows\System32\kazaabackupfiles\Generals_No-CD_Crack.exe
C:\Windows\System32\kazaabackupfiles\Norton_Anti-Virus_2002_Crack.exe
C:\Windows\System32\kazaabackupfiles\PlanetSide.exe
C:\Windows\System32\kazaabackupfiles\Porn.exe
C:\Windows\System32\kazaabackupfiles\Postal_2_Crack.exe
C:\Windows\System32\kazaabackupfiles\Red_Faction_2_No-CD_Crack.exe
C:\Windows\System32\kazaabackupfiles\Renegade_No-CD_Crack.exe
C:\Windows\System32\kazaabackupfiles\Sitebot.exe
C:\Windows\System32\kazaabackupfiles\Winamp_Installer.exe
C:\Windows\System32\kazaabackupfiles\zoneallarm_pro_crack.exe
C:\Windows\System32\keylog.txt
C:\Windows\System32\wuaumqr.exe
-----
Folders added: 1
-----
C:\Windows\System32\kazaabackupfiles

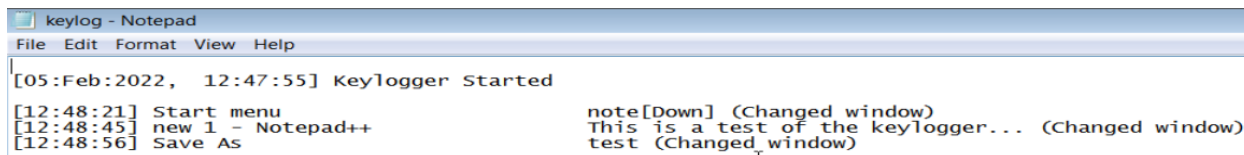
```

Figure 6: Regshot Result: Files and Folders Added

It is, however, too early to make any conclusions regarding the other executable files. They could include further malware or, as the directory suggests, act as backup programs for the malware if *wuaumqr.exe* is destroyed.

2.3 Writing text to a file while the malware is running

Except for those made by Notepad++ and the native system, opening a text document and writing while the malware was active did not affect the Regshot result. However, as illustrated in Figure 7, viewing the text file *keylog.txt* afterward indicates that the keylogger has logged the text, cursor movement, and changes in windows.



```

keylog - Notepad
File Edit Format View Help
[05:Feb:2022, 12:47:55] Keylogger Started
[12:48:21] Start menu
[12:48:45] new 1 - Notepad++
[12:48:56] Save As
note[Down] (Changed window)
This is a test of the keylogger... (Changed window)
test (Changed window)

```

Figure 7: Content of C:\Windows\System32\keylog.txt

2.4 Executing and stopping the malware 3 times

Running the malware three times yielded the same Regshot results as shown in Figure 6, with 18 files and one folder added. There was no change in what processes were started in Process Hacker either, meaning that we first see *assignment2.exe* run for a short while before *wuaumqr.exe* is started and continues to run until we kill the process.

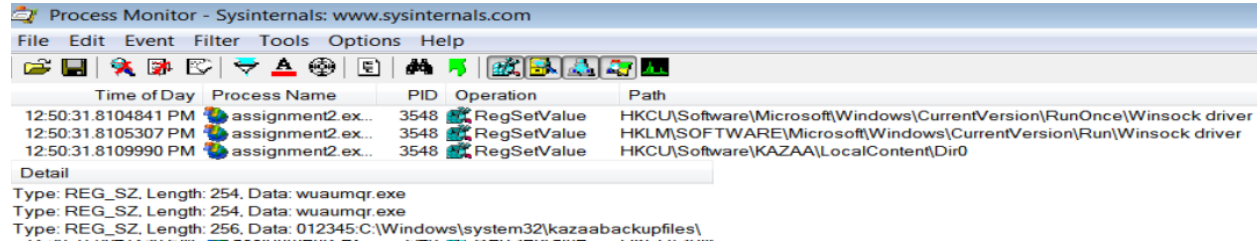
The procedure for starting the malware was the same each time:

1. Start Windows VM
2. Start Process Hacker
3. Start Regshot on C:\
4. Run malware as administrator
5. Wait one minute.
6. In Process Hacker → Right-click on malware process → Terminate tree
7. Click on “2nd shot” in Regshot
8. Hit “compose” Regshot.

9. After analysis → Reset VM to clean snapshot.
10. Repeat

2.5 Chain of events

In this subsection we use process monitor to explain the chain of events caused by executing the malware. According to the findings, *assignment2.exe* is started from the desktop, which is consistent with the fact that we launched it with administrator privileges from the desktop. It then creates the first thread, 904, before modifying the register values displayed in Figure 8 and creating the file *C:\Windows\System32\wuauqmqr.exe*.



Process Monitor - Sysinternals: www.sysinternals.com

Time of Day	Process Name	PID	Operation	Path
12:50:31.8104841 PM	assignment2.exe...	3548	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\Winsock driver
12:50:31.8105307 PM	assignment2.exe...	3548	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Winsock driver
12:50:31.8109990 PM	assignment2.exe...	3548	RegSetValue	HKCU\Software\KAZAA\LocalContent\Dir0

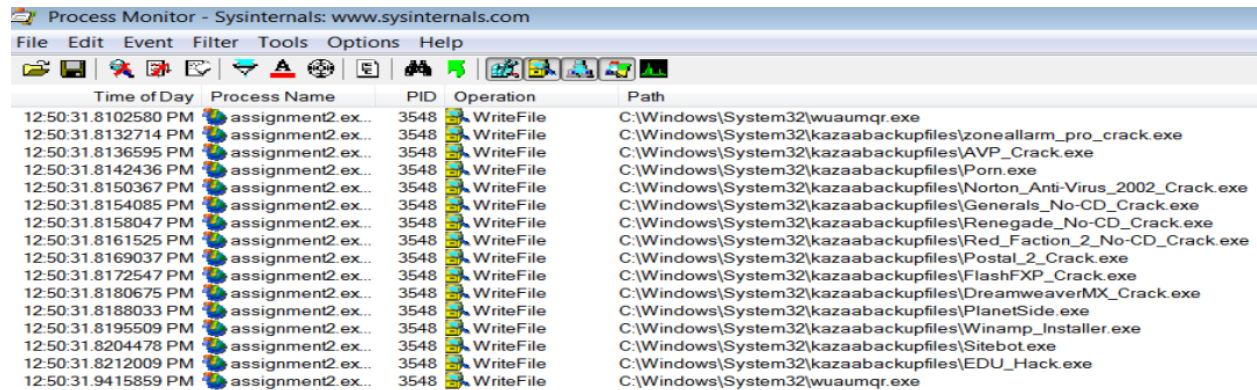
Detail

Type: REG_SZ, Length: 254, Data: wuauqmqr.exe
Type: REG_SZ, Length: 254, Data: wuauqmqr.exe
Type: REG_SZ, Length: 256, Data: 012345.C:\Windows\system32\kazaabackupfiles\

Figure 8: Procmon Result: RegSetValue NB! The screenshot has been edited to better fit the paper. Detail tab is moved down left.

The registry value for *HKCU\...\RunOnce\Winshock driver* and *HKLM\...\Run\Winshock driver* are both set to *wuauqmqr.exe*. This indicates that the malware is trying to achieve persistence on the host system. The *HKCU\...\Dir0* is set to the path of the *kazaabackupfiles* folder. The *RunOnce* register value is, as the name suggests, executed once before it is deleted. Furthermore, as this malware is run on a Windows 7 installation the *RunOnce* register value can be activated by two instances, either each time the computer starts or after a device is installed [2]. This made us wonder if the malware is chaining a driver or device to execute the program.

After that, the rest of the files in Figure 9 are created. The remaining 14 executable files saved in the backup folder appear to be similar implementations of the keylogger malware that can be run if *wuauqmqr.exe* is destroyed, according to a quick check with Regshot. Threads 1020, 3720, 3084, 3116, and 3124 are created by the program (see Figure 12).



Process Monitor - Sysinternals: www.sysinternals.com

Time of Day	Process Name	PID	Operation	Path
12:50:31.8102580 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\wuauqmqr.exe
12:50:31.8132714 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\zoneallarm_pro_crack.exe
12:50:31.8136595 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\AVP_Crack.exe
12:50:31.8142436 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Porn.exe
12:50:31.8150367 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Norton_Anti-Virus_2002_Crack.exe
12:50:31.8154085 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Generals_No-CD_Crack.exe
12:50:31.8158047 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Renegade_No-CD_Crack.exe
12:50:31.8161525 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Red_Faction_2_No-CD_Crack.exe
12:50:31.8169037 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Postal_2_Crack.exe
12:50:31.8172547 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\FishFXP_Crack.exe
12:50:31.8180675 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\DreamweaverMX_Crack.exe
12:50:31.8188033 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\PlanetSide.exe
12:50:31.8195509 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Winamp_Installer.exe
12:50:31.8204478 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\Sitebot.exe
12:50:31.8212009 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\kazaabackupfiles\EDU_Hack.exe
12:50:31.9415859 PM	assignment2.exe...	3548	WriteFile	C:\Windows\System32\wuauqmqr.exe

Figure 9: Procmon Result: WriteFile

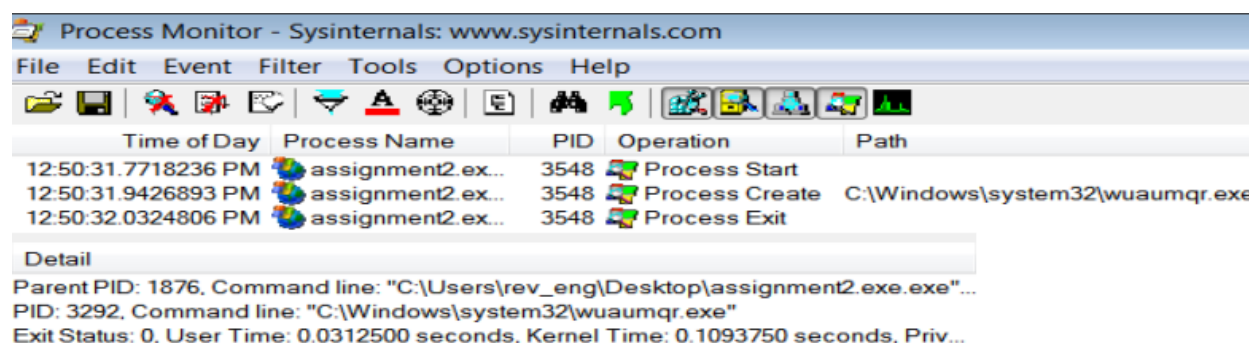
As shown in Figure 10, *assignment2.exe* continues by setting a lot of values in registers, including four related to internet settings, which is interesting because it could be connected to the sockets that we assume it is using for an internet connection based on the static analysis.

12:50:31.9386556 PM	assignment2.ex...	3548	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMa...
12:50:31.9386600 PM	assignment2.ex...	3548	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMa...
12:50:31.9389393 PM	assignment2.ex...	3548	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMa...
12:50:31.9389422 PM	assignment2.ex...	3548	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMa...

Figure 10: Procmon Result: RegSetValue - socket

After setting the values of the registers it writes to the *C:\Windows\System32\wuaumqr.exe* file again before starting it as a process as shown in Figure 11. This is followed by the creation of two more threads, 3080 and 2040 before *assignment2.exe* terminates the processes and threads it has started and exits its process, as seen in Figure 11, and 12.

It writes to the *C:\Windows\System32\wuaumqr.exe* file again after adjusting the register values before starting it as a process, as illustrated in Figure 11. After that, two more threads, 3080 and 2040, are created before *assignment2.exe* as seen in Figures 11 and 12, kills the processes and threads it has spawned and quits its process.



Process Monitor - Sysinternals: www.sysinternals.com

Time of Day	Process Name	PID	Operation	Path
12:50:31.7718236 PM	assignment2.ex...	3548	Process Start	
12:50:31.9426893 PM	assignment2.ex...	3548	Process Create	C:\Windows\system32\wuaumqr.exe
12:50:32.0324806 PM	assignment2.ex...	3548	Process Exit	

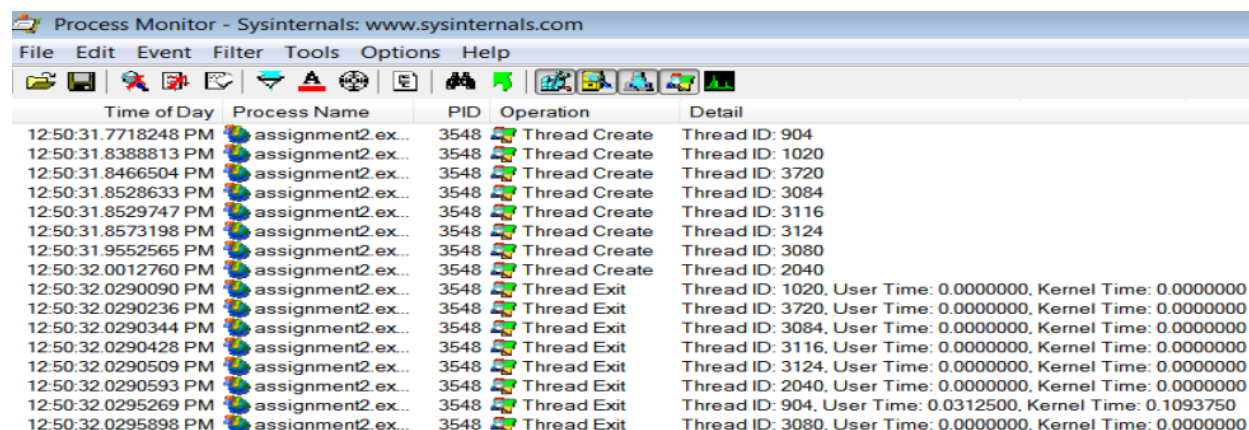
Detail

Parent PID: 1876, Command line: "C:\Users\rev_eng\Desktop\assignment2.exe.exe"...

PID: 3292, Command line: "C:\Windows\system32\wuaumqr.exe"

Exit Status: 0, User Time: 0.0312500 seconds, Kernel Time: 0.1093750 seconds, Priv...

Figure 11: Procmon Result: Process Start, Process Create, Process exit *NB! The screenshot has been edited to better fit the paper. Detail tab is moved down left.*



Process Monitor - Sysinternals: www.sysinternals.com

Time of Day	Process Name	PID	Operation	Detail
12:50:31.7718248 PM	assignment2.ex...	3548	Thread Create	Thread ID: 904
12:50:31.8388813 PM	assignment2.ex...	3548	Thread Create	Thread ID: 1020
12:50:31.8466504 PM	assignment2.ex...	3548	Thread Create	Thread ID: 3720
12:50:31.8528633 PM	assignment2.ex...	3548	Thread Create	Thread ID: 3084
12:50:31.8529747 PM	assignment2.ex...	3548	Thread Create	Thread ID: 3116
12:50:31.8573198 PM	assignment2.ex...	3548	Thread Create	Thread ID: 3124
12:50:31.9552565 PM	assignment2.ex...	3548	Thread Create	Thread ID: 3080
12:50:32.0012760 PM	assignment2.ex...	3548	Thread Create	Thread ID: 2040
12:50:32.0290090 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 1020, User Time: 0.0000000, Kernel Time: 0.0000000
12:50:32.0290236 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 3720, User Time: 0.0000000, Kernel Time: 0.0000000
12:50:32.0290344 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 3084, User Time: 0.0000000, Kernel Time: 0.0000000
12:50:32.0290428 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 3116, User Time: 0.0000000, Kernel Time: 0.0000000
12:50:32.0290509 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 3124, User Time: 0.0000000, Kernel Time: 0.0000000
12:50:32.0290593 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 2040, User Time: 0.0000000, Kernel Time: 0.0000000
12:50:32.0295269 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 904, User Time: 0.0312500, Kernel Time: 0.1093750
12:50:32.0295898 PM	assignment2.ex...	3548	Thread Exit	Thread ID: 3080, User Time: 0.0000000, Kernel Time: 0.0000000

Figure 12: Procmon Result: Thread Create, Thread Exit

2.6 Investigating any network traffic

We tested the malware in two scenarios: one with no network simulator running to respond to a potential request to IP addresses, and another with Inetsim running to simulate the network requests the malware was expecting.

In the first scenario, we observe that the malware, wuaumqr.exe, is first trying to establish a TCP connection to 209.126.201.20, and when that failed it tried to establish a TCP connection to 209.126.201.22. As demonstrated in Figure 13, this behavior repeated itself as long as the malware was unable to establish a connection with one of the targeted remote servers. This pattern is consistent with what we predicted based on the static analysis and IP addresses.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.103	209.126.201.20	TCP	66	49168 → 6667 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
2	0.000049167	209.126.201.20	192.168.56.103	TCP	54	6667 → 49168 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.501687049	192.168.56.103	209.126.201.20	TCP	66	[TCP Retransmission] 49168 → 6667 [SYN] Seq=0 Win=0 Len=0
4	0.501726580	209.126.201.20	192.168.56.103	TCP	54	6667 → 49168 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.001555367	192.168.56.103	209.126.201.20	TCP	62	[TCP Retransmission] 49168 → 6667 [SYN] Seq=0 Win=0 Len=0
6	1.001598165	209.126.201.20	192.168.56.103	TCP	54	6667 → 49168 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	4.517682924	PcsCompu_f0:7e:a1	PcsCompu_f8:14:19	ARP	60	Who has 192.168.56.102? Tell 192.168.56.103
8	4.517698125	PcsCompu_f8:14:19	PcsCompu_f0:7e:a1	ARP	42	192.168.56.102 is at 08:00:27:f8:14:19
9	6.002218817	192.168.56.103	209.126.201.22	TCP	66	49169 → 6666 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
10	6.002263380	209.126.201.22	192.168.56.103	TCP	54	6666 → 49169 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	6.517826570	192.168.56.103	209.126.201.22	TCP	66	[TCP Retransmission] 49169 → 6666 [SYN] Seq=0 Win=0 Len=0
12	6.517871775	209.126.201.22	192.168.56.103	TCP	54	6666 → 49169 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	7.033117114	192.168.56.103	209.126.201.22	TCP	62	[TCP Retransmission] 49169 → 6666 [SYN] Seq=0 Win=0 Len=0
14	7.033163625	209.126.201.22	192.168.56.103	TCP	54	6666 → 49169 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	11.470509294	192.168.56.103	192.168.56.102	DNS	85	Standard query 0xcbb70 A teredo.ipv6.microsoft.com
16	11.47057111	192.168.56.102	192.168.56.103	ICMP	113	Destination unreachable (Port unreachable)
17	12.032863511	192.168.56.103	209.126.201.20	TCP	66	49170 → 6667 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
18	12.032906627	209.126.201.20	192.168.56.103	TCP	54	6667 → 49170 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	12.533213065	192.168.56.103	209.126.201.20	TCP	66	[TCP Retransmission] 49170 → 6667 [SYN] Seq=0 Win=0 Len=0
20	12.533252733	209.126.201.20	192.168.56.103	TCP	54	6667 → 49170 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	13.049139911	192.168.56.103	209.126.201.20	TCP	62	[TCP Retransmission] 49170 → 6667 [SYN] Seq=0 Win=0 Len=0
22	13.049186438	209.126.201.20	192.168.56.103	TCP	54	6667 → 49170 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 13: Wireshark: Without Inetsim

When we run the second scenario, we can see that the TCP connection is established and that wuaumqr.exe begins utilizing the Internet Relay Chat (IRC) protocol to communicate with the first remote server at IP address 209.126.201.20 (see Figure 14).

3	2.081276261	192.168.56.103	209.126.201.20	TCP	66	49168 → 6667 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	2.081324518	209.126.201.20	192.168.56.103	TCP	66	6667 → 49168 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
5	2.081555702	192.168.56.103	209.126.201.20	TCP	60	49168 → 6667 [ACK] Seq=1 Ack=1 Win=65536 Len=0
6	2.082116052	192.168.56.103	209.126.201.20	IRC	127	Request (NICK) (USER)
7	2.082129790	209.126.201.20	192.168.56.103	TCP	54	6667 → 49168 [ACK] Seq=1 Ack=74 Win=64256 Len=0
8	2.085143194	192.126.201.20	192.168.56.103	IRC	99	Response (NOTICE)
9	2.280921935	192.168.56.103	209.126.201.20	TCP	60	49168 → 6667 [ACK] Seq=74 Ack=46 Win=65536 Len=0
10	2.280956470	209.126.201.20	192.168.56.103	IRC	227	Response (NOTICE) (NOTICE) (NOTICE) (NOTICE) (PING)
11	2.281289336	192.168.56.103	209.126.201.20	IRC	76	Request (PONG)
12	2.281317124	209.126.201.20	192.168.56.103	TCP	54	6667 → 49168 [ACK] Seq=219 Ack=96 Win=64256 Len=0
13	2.283647218	209.126.201.20	192.168.56.103	IRC	135	Response (001)
14	2.499879367	192.168.56.103	209.126.201.20	TCP	60	49168 → 6667 [ACK] Seq=96 Ack=300 Win=65280 Len=0
15	2.499915280	209.126.201.20	192.168.56.103	IRC	293	Response (002) (003) (004)
16	2.702541462	192.168.56.103	209.126.201.20	TCP	60	49168 → 6667 [ACK] Seq=96 Ack=539 Win=65024 Len=0

Figure 14: Wireshark: Inetsim

Many of the stings discovered during the static analysis may be recognized by looking at the IRC requests collected by Wireshark, as seen in Figure 15. Despite the fact that the responses are from Inetsim rather than the malicious server, we can observe that the malware is connecting to a distant IRC server and setting its IRC nickname to *rev_eng69*, as shown in Figure 16. Otherwise, it's difficult to tell much more from Inetsim's generic IRC activity.

ascii	7	-	-	-	n/a	PRIVMSG
ascii	7	-	-	-	n/a	NICK %s
ascii	7	-	-	-	n/a	JOIN %s
ascii	7	-	-	-	n/a	PONG %s
ascii	7	-	-	-	n/a	NICK %s

Figure 15: PeStudio Result: Possible IRC Strings

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.103	209.126.201.20	TCP	66	49168 → 6667 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000054253	209.126.201.20	192.168.56.103	TCP	66	6667 → 49168 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.000225471	192.168.56.103	209.126.201.20	TCP	60	49168 → 6667 [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	0.001136445	192.168.56.103	209.126.201.20	IRC	127	Request (NICK) (USER)

▶ Frame 4: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits) on interface enp0s17, id 0
 ▶ Ethernet II, Src: PcsCompu_f0:7e:a1 (08:00:27:f0:7e:a1), Dst: PcsCompu_f8:14:19 (08:00:27:f8:14:19)
 ▶ Internet Protocol Version 4, Src: 192.168.56.103, Dst: 209.126.201.20
 ▶ Transmission Control Protocol, Src Port: 49168, Dst Port: 6667, Seq: 1, Ack: 1, Len: 73
 ▶ Internet Relay Chat
 Request: NICK rev_eng69
 Command: NICK
 Command parameters
 Parameter: rev_eng69
 Request: USER rev_eng69 "hotmail.com" "192.168.56.103" :rev_eng
 Command: USER
 Command parameters
 Parameter: rev_eng69
 Parameter: "hotmail.com"
 Parameter: "192.168.56.103"
 Trailer: rev_eng

Figure 16: Wireshark: IRC (nick)(user) request

This protocol can be used by malware to deliver commands and control (C2) responses to the actor who produced the malware, as it can be used for one-to-one communication with private messages, as well as chat and file transfer [1, p. 309]. This assumption is strengthened by the fact that botnets frequently use IRC for C2 communication because it also permits broadcasting.

We also discovered strings relating to *PRIVMSG* and *DCC SEND*, both of which are utilized by the IRC protocol to send private messages using commands such as *privmsg*, *notice*, or *msg* [3]. This confirms our hypothesis that the malware is attempting to connect to a remote server running IRC to transmit private messages. We see that the established TCP connection is closed when we terminate the running malware.

2.7 Updated hypothesis

Based on the findings of both the basic static and basic dynamic analyses, we revise our hypothesis:

The malware, *assignment2.exe*, is highly likely a spyware program containing the keylogger malware *wuaumqr.exe*. When executed it creates *wuaumqr.exe* in the `\System32\` directory. Before adding the rest of the executable files to the `\System32\kazaabackupfiles` backup directory, it modifies register values to provide *wuaumqr.exe* and the *kazaabackupfiles* directory persistence on reboot. Before exiting, *assignment2.exe* starts the *wuaumqr.exe* keylogger application. The background process of *wuaumqr.exe* produces the *key-log.txt* file under `\System32\`, which stores the captured data. *wuaumqr.exe* attempts to establish a TCP connection to one of two remote servers with the IP-addresses 209.126.201.20 or 209.126.201.22. The IP addresses might resolve to either *sub7server*, *kuangserver*, or *SpyBot1.2*. If successful it starts using TCP and IRC to transmit the recorded data and establish C2 communication via private messages with the actor.

References

- [1] M. Sikorski and A. Honig, *Practical Malware Analysis - The Hands-On Guide to Dissecting Malicious Software*. 38 Ringold Street, San Francisco, CA 94103: William Pollock, 2012, ISBN-13: 978-1-59327-290-6.
- [2] Microsoft, “Developer tools, technical documentation and coding examples,” (Accessed: 12.02.2022). [Online]. Available: <https://docs.microsoft.com/en-gb/>
- [3] J. Oikarinen and D. Reed, “Internet relay chat protocol,” 1993, (Accessed: 20.02.2022). [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc1459>