Version 1.0

Malware Analysis Report

SHA256: -

- Sample Two: c630037470c043751f79456680a9d3277e1d33836888b2417223529d73dd1df1

Presented by: Dakshitha Perera

10519381

pdakshi@our.ecu.edu.au

Table of Contents

..... 1

Malware Sample One 2

Executive Summary..... 2

Identification of malware sample..... 3

Details of architecture targeted by malware. 7

Details of packing or obfuscation 8

Details of malware behaviour 15

Dynamic Analysis:..... 15

Noticeable behaviours:..... 17

Interaction with the File System: 17

Created Files: 17

Removed and Closed Files: 18

Network Communication: 19

Registry Keys:..... 23

Detection Rules 25

YARA Rules 25

Snort Rules 27

Detection and Malware Removal Instructions 28

Running Sheet..... 1

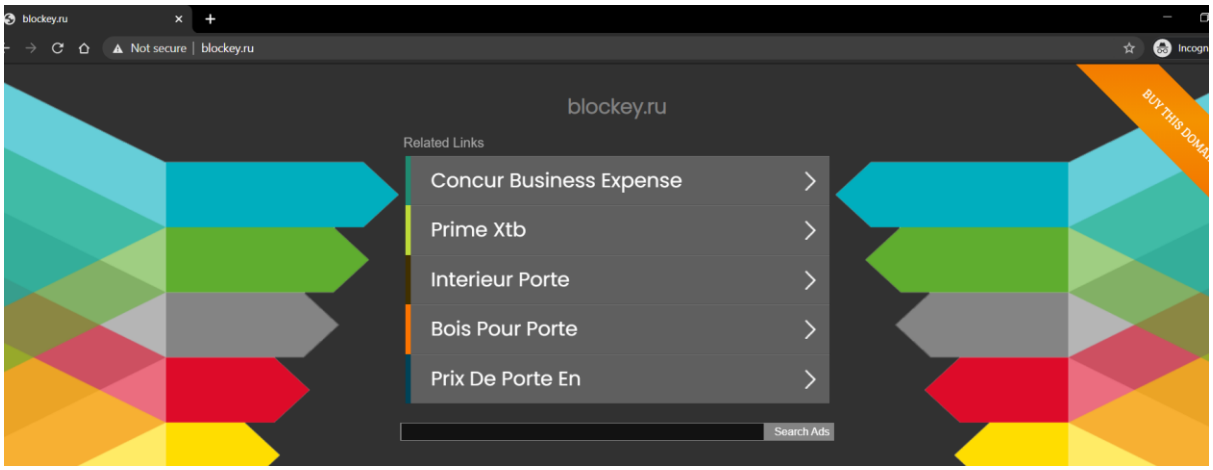
References: 1

Malware Sample One

Executive Summary

The given malware was analyse using various static, dynamic and automated analysis techniques. This is a Windows Operating system target spyware. Like the first malware sample, this malware sample could reach a computer using patched freeware or shareware. However, Social engineering could also play a role to get into a personal computer. The complete malware behaviour was not analysed due to the given limited time. However, with uses of different automated sandboxes, malware behaviour has been analysed up to some extent. Unlike the first malware sample there is not enough evidence to confirm that browser hijacking takes place. Nonetheless, it also collects the victim's sensitive information such as web browser cookies, password hashes, usernames, history, etc, and sends them to a remote server in Russia.

This malware is a 32-bit executable that was created around 2017. However, some of its behaviours are now obsolete. For instance: some domains that it supposes to contact are now for sale.



This malware sample can be considered as an unpacked malware. Even though, it has considerably higher entropy. Moreover, updated windows defender software has the ability to detect this based on its signature and delete it automatically. Therefore, if the PC is up to date. The threat from this malware is extremely low. However, this malware threat increases rapidly as soon as the computer is infected. According to virustotal information, there are 55 engines that have the ability to detect this malware. And this malware normally comes to the PC with the name of "Allavsoft Video Downloader Converter 3.14.6.exe". Interestingly, this malware has a valid signature.

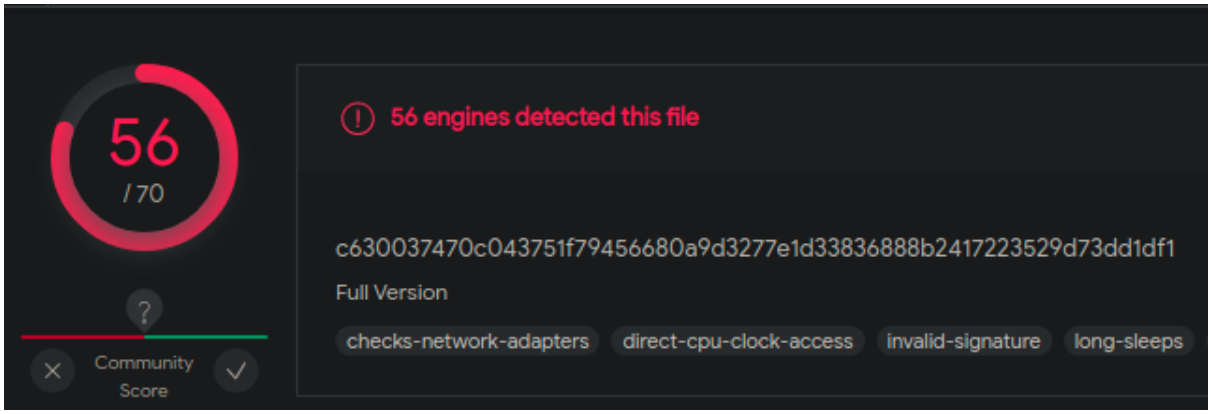
Furthermore, as interesting key behaviours of this malware, it checks victim computer network adapters and monitors victim computer traffic flow. This malware has a long sleep time which allow the malware to stay in PC unsuspiciously.

Identification of malware sample

The Second malware sample is also a 32bit PE executable which targets only Microsoft Window (GUI) with Intel 368 or later process. MS Visual C++ has been used as the main TRiD. This malware's file size is 194.7Kb which is around 199368Bytes. And this is a signed file with valid signatures in it. According to the PE Studio the malware's complied stump is 2017 June 09. Which is a relatively new date.

This malware was written in C++. And this includes raw COFF/OMF content. In Virustotal.com, 47 engines detected this file as a malicious file.

cpu	32-bit
subsystem	GUI
compiler-stamp	0x593A4C58 (Fri Jun 09 00:20:56 2017)
debugger-stamp	n/a
resources-stamp	



-And primarily, Virustotal suggests the name of "Allavsoft Video Downloader Converter 3.14.6.exe" to identify the malware. However, different anti-malware has some different name to this malware. Here are some names by famous engineers and its names.

FireEye	Generic.mg.976fd3e98a0ce54a
Avast	Win32:Downloader-UNP [Drp]
Alibaba	Trojan:Win32/AdLoad.b2514328
Microsoft	TrojanDownloader:JS/Istbar!atmn
Comodo	ApplicUnwnt@#2l9zt7ignzohz
BitDefender	Generic.Application.Adload.B8F92FBC

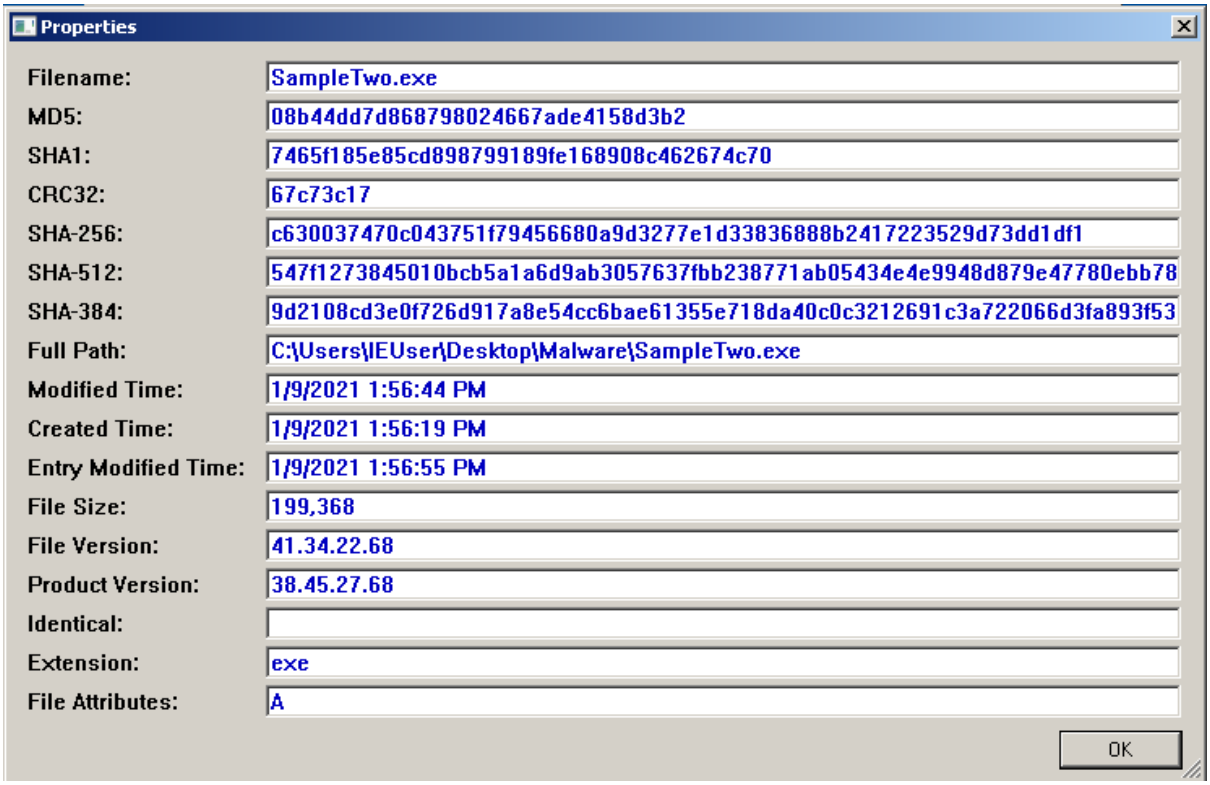
Microsoft Visual C++ 8 can be classified as the compiler or the packer in this malware. Russian has been used as the language of this malware and the computer which this malware was written with. However, some resources of the malware have different languages. For example: the version has Japanese language. Another important detail of this malware is its entropy, this malware contains 6.829 entropy. Thus, this malware is possibly packed. However, This report dive into its depths later.

The basic properties of the malware can be categorized as below.

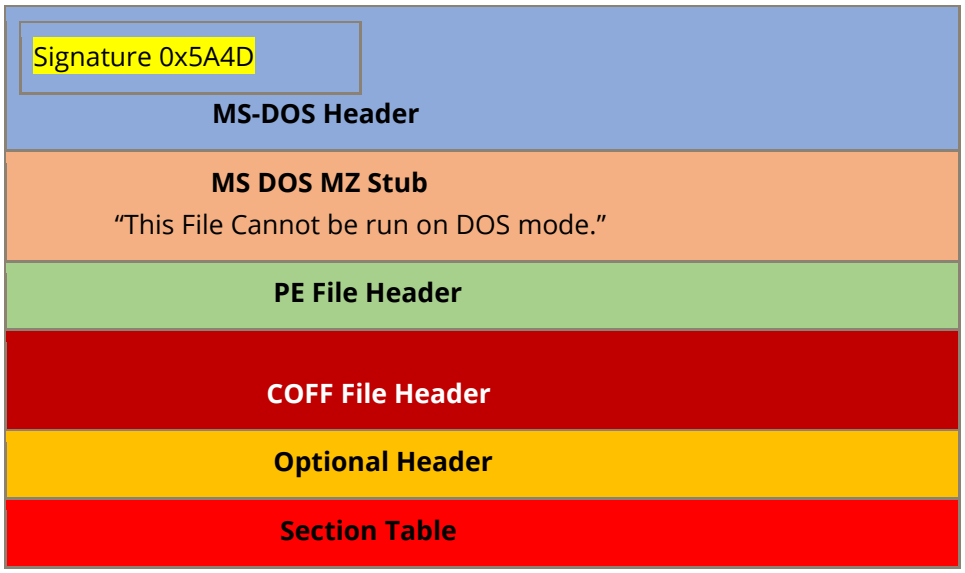
MD5	08b44dd7d868798024667ade4158d3b2
SHA1	7465f185e85cd898799189fe168908c462674c70
SHA256	c630037470c043751f79456680a9d3277e1d33836888b2417223529d73dd1df1

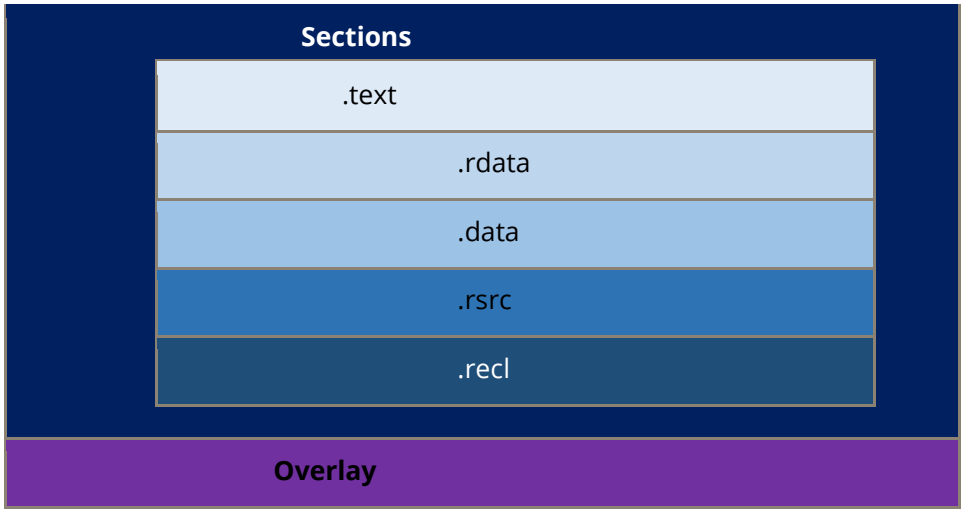
VHASH	015056655d15556az437z1dz6fz
PE header hash	f9ec15ca0a2401b65274823ebd2852ef

And Here is the output of the HashMyFile tool with the second malware sample.



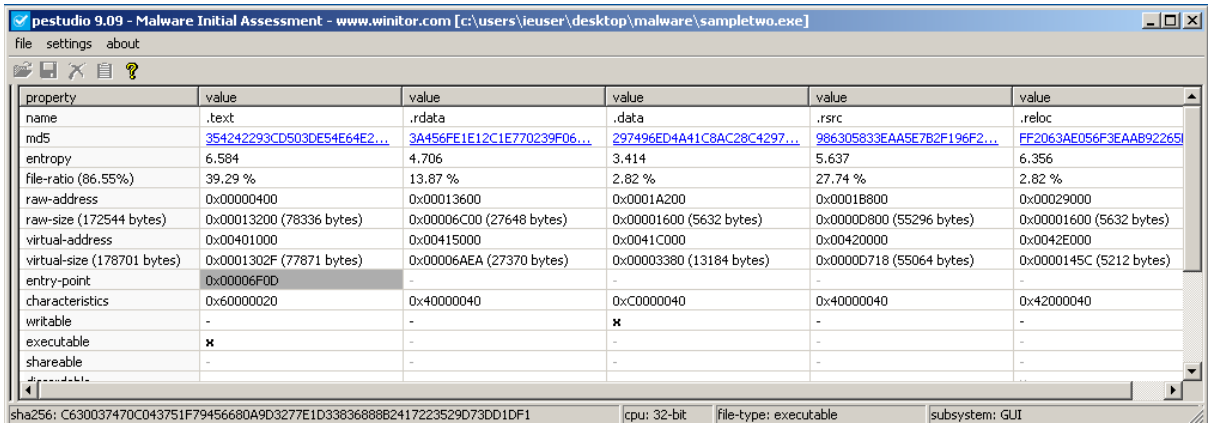
This malware contains rich file headers.





Furthermore, this file contains 5 sections, which are:

1. .text
2. .rdata
3. .data
4. .rsrc
5. recl



However, these sections do not have a high number of entropies.

Sections Name	Address	Size	Entropy
.text	0x0040100	77871 bytes	6.58
.rdata	0x0040500	27370 bytes	4.7
.data	0x0041C000	5632 bytes	3. 41
.rsrc	0x00420000	55296 bytes	5.67
.recl	0x0042E000	5632 bytes	6.356

Details of architecture targeted by malware.

As this report mentioned earlier, This malware only targets Microsoft Windows GUI subsystem with intel 80236 32-bit architecture. Furthermore, the targeted endianness is little endianness. And here is the malware manifest. This reveal which useful information about the target architecture.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0" xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"><dependency><dependentAssembly><assemblyIdentity type="win32" name="Microsoft.Windows.Common-Controls" version="6.0.0.0" language="*" processorArchitecture="x86" publicKeyToken="6595b64144ccf1df" />
</dependentAssembly></dependency><compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
<application>
<!-- Windows Vista -->
<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}"/>
<!-- Windows 7 -->
<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>
<!-- Windows 8 -->
<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}"/>
<!-- Windows 8.1 -->
<supportedOS Id="{1f676c76-80e1-4239-95bb-83d0f6d0da78}"/>
<!-- Windows 10 -->
<supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}"/>
</application></compatibility><trustInfo xmlns="urn:schemas-microsoft-com:asm.v3"><security><requestedPrivileges><requestedExecutionLevel level="requireAdministrator" uiAccess="false"
/></requestedPrivileges></security></trustInfo></assembly>
```

And as normal, this malware does not run-on DOS mode.

Category	Details
Compiler	Microsoft Visual C++ 8
Architecture	Intel 80386 [Windows]
Operating System	Microsoft Windows
Library	Microsoft Visual C++
File-Type	Executable
Subsystem	GUI
Endianness	LE
CPU	32-Bit

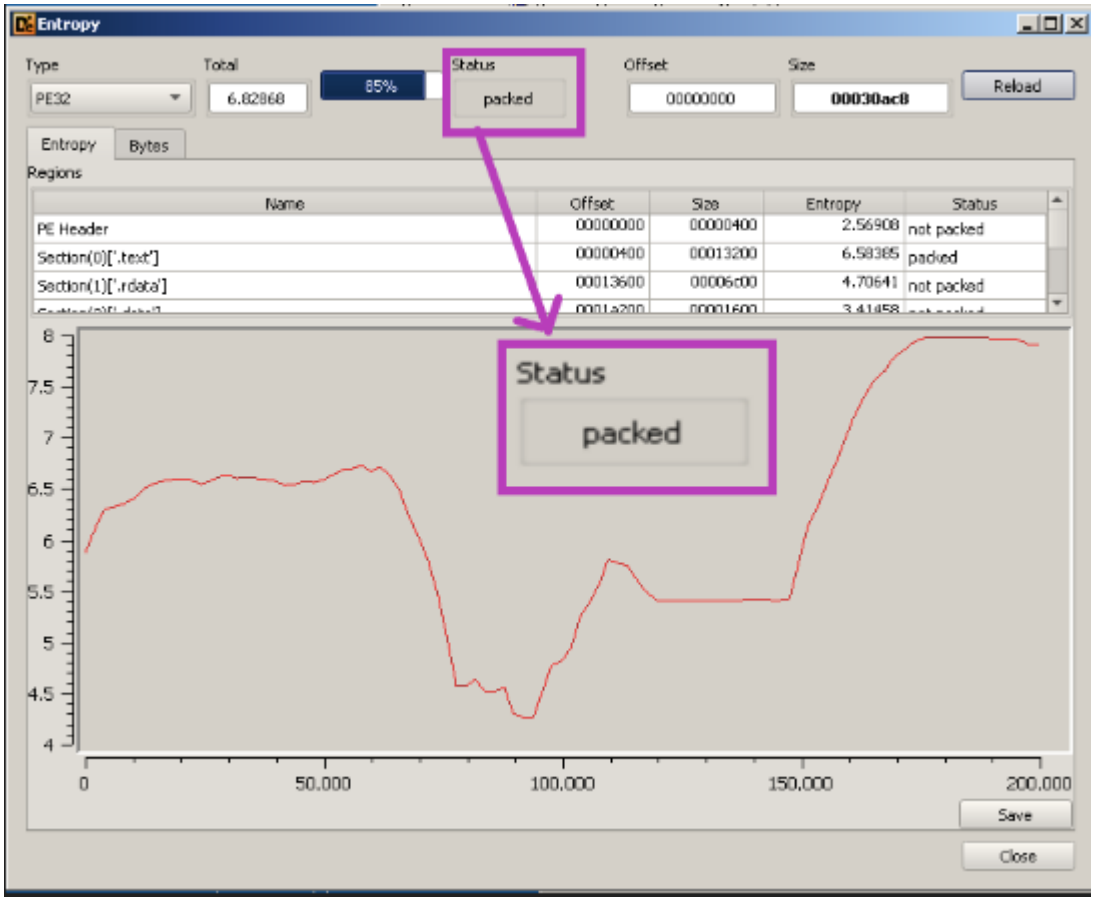
Linker	Microsoft Linker (8.0)
Overlay	

Details of packing or obfuscation

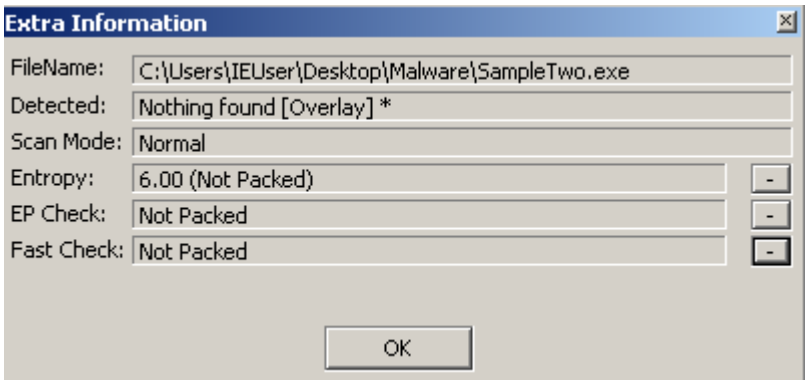
According to the Virustotal and Hybrid-Analysis, this malware packer is Microsoft visual C++ 8. And this is just a redistributable package. This packer does not actually hide any data of the malware. Therefore, All the strings, libraries and imports can be easily found with appropriate tools.

name (74)	group (10)	type (1)
TranslateMessage	windowing	implicit
GetDesktopWindow	windowing	implicit
SendMessageW	windowing	implicit
PeekMessageW	windowing	implicit
DispatchMessageW	windowing	implicit
IsProcessorFeaturePresent	system-information	implicit
IsDebuggerPresent	system-information	implicit
QueryPerformanceCounter	system-information	implicit
WaitForMultipleObjects	synchronization	implicit
EnterCriticalSection	synchronization	implicit
LeaveCriticalSection	synchronization	implicit
DeleteCriticalSection	synchronization	implicit
InitializeCriticalSectionAnd...	synchronization	implicit
GetStringTypeW	memory	implicit
HeapReAlloc	memory	implicit
HeapAlloc	memory	implicit
HeapFree	memory	implicit
HeapSize	memory	implicit
GetProcessHeap	memory	implicit
CreateFileW	file	implicit
FlushFileBuffers	file	implicit
ReadFile	file	implicit
GetSystemTimeAsFileTime	file	implicit
SetFilePointerEx	file	implicit
GetFileType	file	implicit
WriteFile	file	implicit
SetEndOfFile	file	implicit
CreateThread	execution	implicit
Sleep	execution	implicit
GetCommandLineA	execution	implicit
ExitProcess	execution	implicit

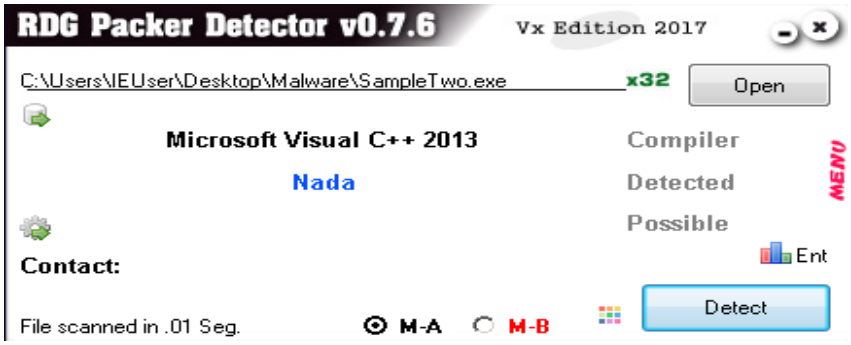
However, if you check the malware sample with DIE (Detect It Easy). DIE illustrates that the malware sample has entropy around 6.8 and it has been packed.



However, PEiD tool confirms that the malware has not been packed. And its entropy is quite lower than what showed by DIE.



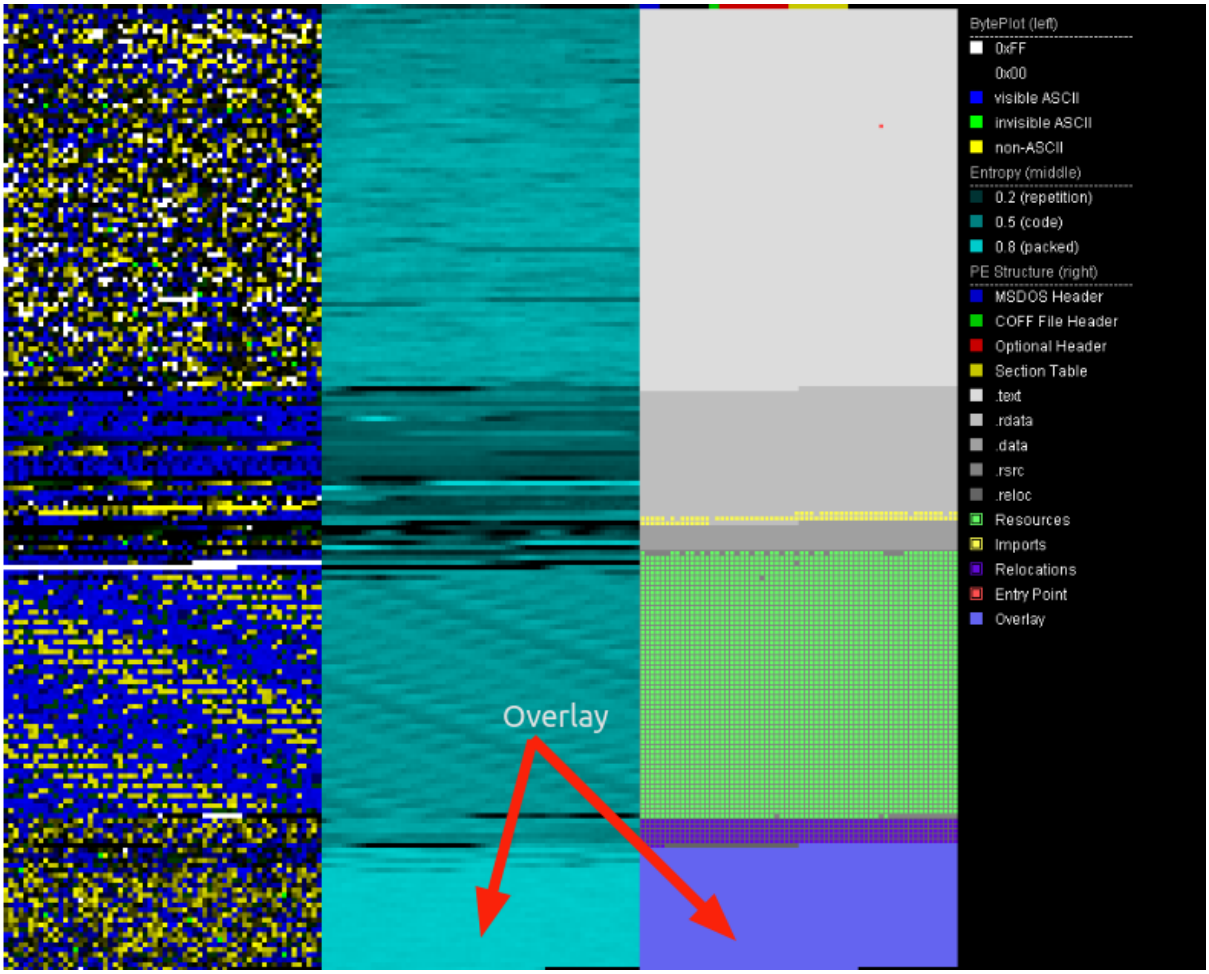
RDG packer Detector also outputs the same output as PEiD did.



Furthermore, this malware does not contain imports such as LoadLibrary and GetProAddress which are useful for packing. Nevertheless, if you use PortEx Analyzer to check it. You will find some areas of the malware sample have been packed.

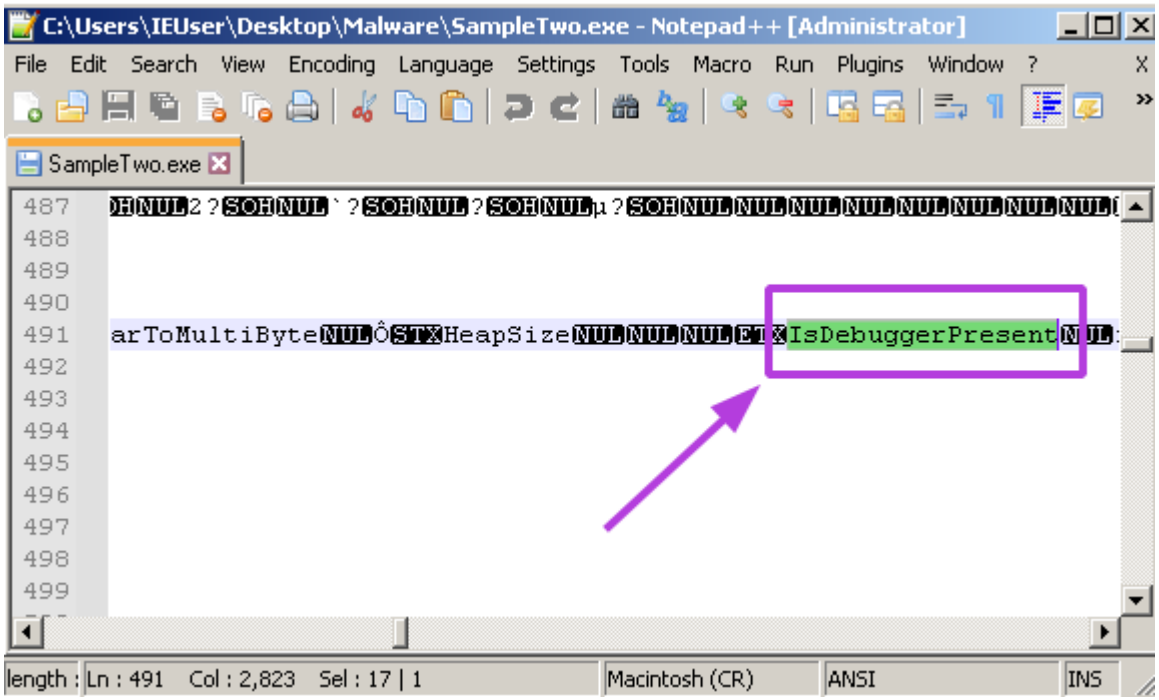
```
C:\Users\IEUser\Desktop>java -jar PortExAnalyzer.jar -o MalwareSampleTwo.txt -p
MalwareSampleTwo.png Malware\SampleTwo.exe
PortEx Analyzer

Creating report file...
Writing header reports...
Writing section reports...
Writing analysis reports...
Report done!
creating visualization...
picture successfully created and saved to C:\Users\IEUser\Desktop\MalwareSampleT
wo.png


C:\Users\IEUser\Desktop>
```

This malware contains an overlay part, and it has a higher amount of entropy.

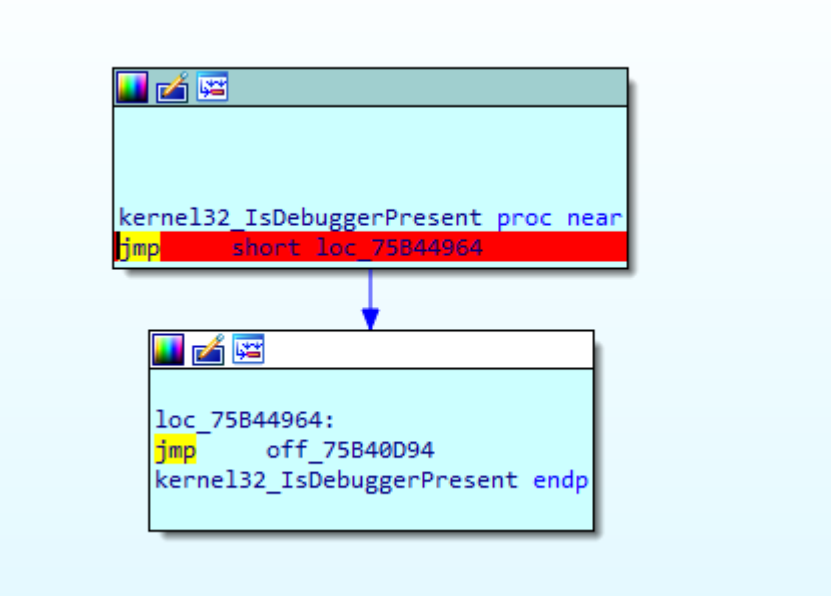


However, in my opinion, this will not impact on understanding the malware behaviour. However, this unpacked area can also be unpacked. But it does not make any difference.

Nonetheless, Unpacking the malware could be hard. Because the malware contains a function called IsDebuggerPresent. (Sikorski, Honig, 2012)



IsDebuggerPresent is a kernel32 function which determines whether a process is being debugged. But this only checks the user-mode debugger. (Yason, 2007)



There are few tricks that can be used to go through this Anti-Debugging obfuscation.

With Some plugin which can be used with OllyDBG, this issue can be resolve very easily. plugins such as

1. Olly Advance
2. Panth0m can be used to avoid this. (Sikorski, Honig, 2012)

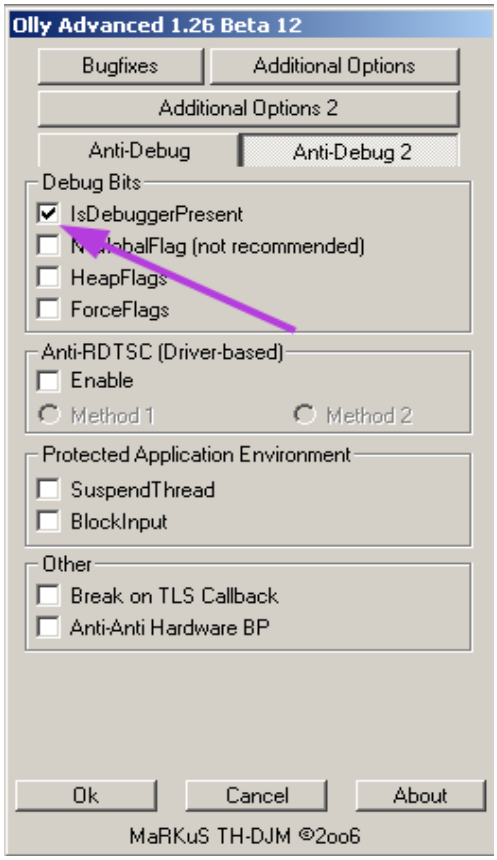


Figure 1: Ollydgb plugin called Olly Advanced.

Or the malware can be patched in order to avoid this. This function returns a one if the process is being debugged.

```
KernelBase.dll:769A394B ; ---------------------------------------------------------------------------
KernelBase.dll:769A394B
KernelBase.dll:769A394B kernelbase_IsDebuggerPresent:
KernelBase.dll:769A394B mov     eax, large fs:18h               ; DATA XREF: kernel32.dll:off_75840D94↑o
KernelBase.dll:769A3951 mov     eax, [eax+30h]
KernelBase.dll:769A3954 movzx   eax, byte ptr [eax+2]
KernelBase.dll:769A3958 retn
KernelBase.dll:769A3958 ; ---------------------------------------------------------------------------
```

Figure 2: Assembly code of IsDebugerPresent funcion

This code checks whether the debugger is available with PEB (ProcessEnvironmentBlock) via TEB(ThreatEnvironmentBlock). With mov large fs:18h self-pointing to TEB. then the offset 0x30 points to the PEB.

4	FS:[0x28]	GS:[0x50]	NT	Active RPC Handle
4	FS:[0x30]	GS:[0x60]	NT	Linear address of Process Environment Block (PEB)

Figure 3: TEB functions

And the second byte of the Process Environment Block structure refers to Beingdebugged. It checks with movzx eax, byte ptr [eax+2], the value stores in the eax registry. (Yason, 2007)

```
typedef struct _PEB {
  BYTE                          BeingDebugged;
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

After that it compares the EAX value with 0. Thus, if you change the value of 1 to 0. You will be able to get through this obfuscation technique.

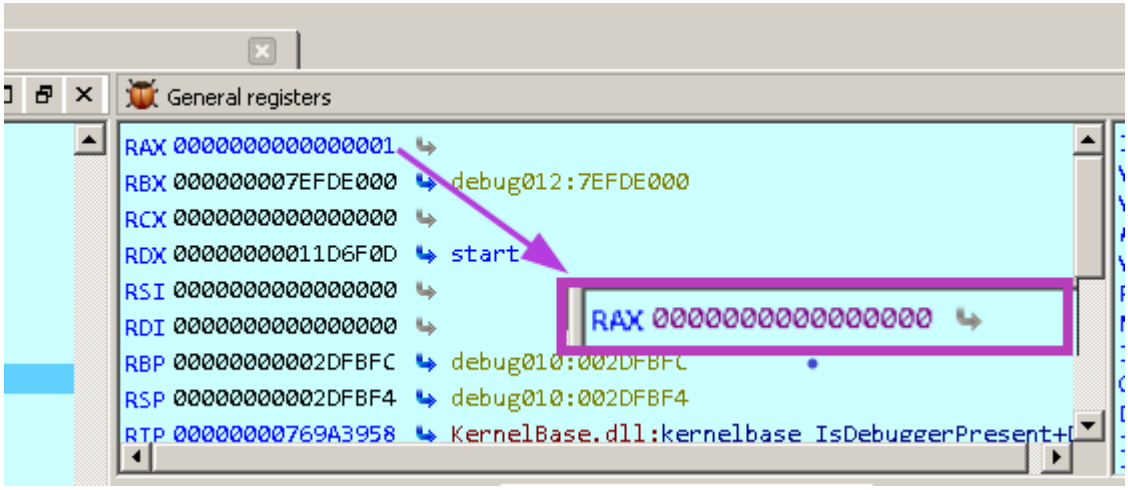


Figure 4: Changing RAX value 1 to 0

Furthermore, this malware has OutputDebugString. This is also a kernel32 function. It also checks whether the program is being debugged or not.

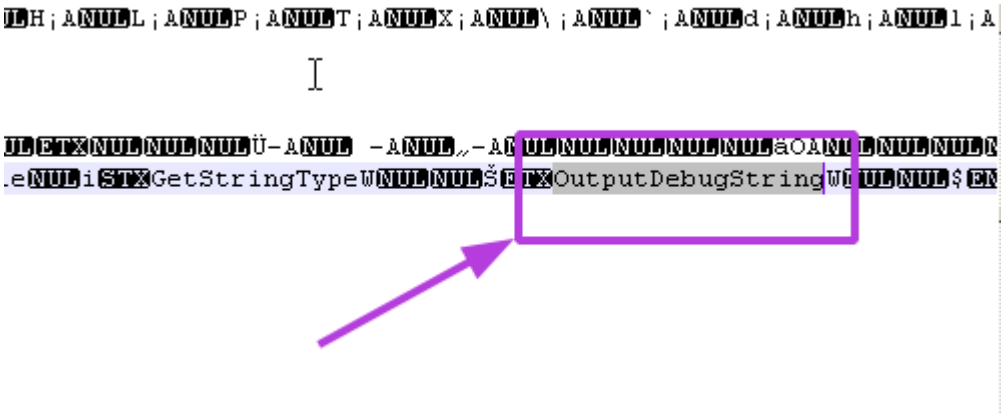
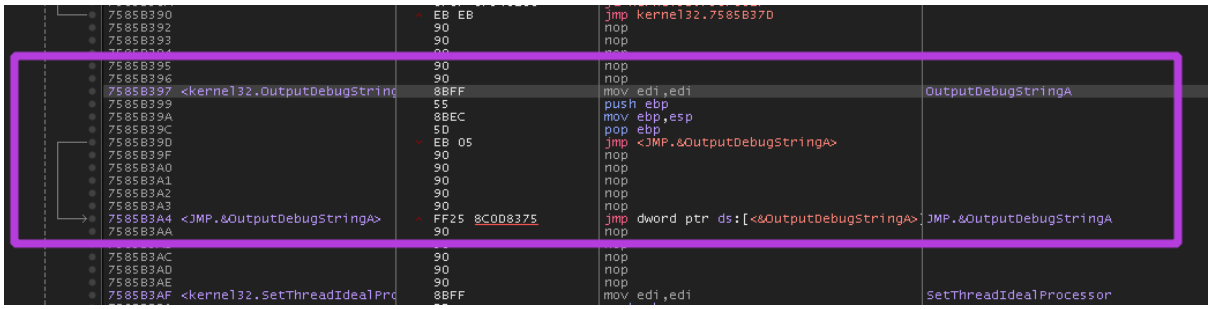


Figure 5:OutPuDebugString with notepade++

This function gets kernel 32 GetLastError() function's return value. And check whether the process is being debugged or not. However, this only works with Windows NT, 2000 or XP. If you use windows vista or later versions, this can be ignored. (peter, 2011)



Details of malware behaviour

This malware behaviour also stretches to a significant range. Network behaviours, interaction with computer files system to registers changes can be experienced with malware. Thus, this malware has characteristics of an adware. When the basic picture of malware is being considered, the malware creates two additional iexplore.exe processes while it is running.

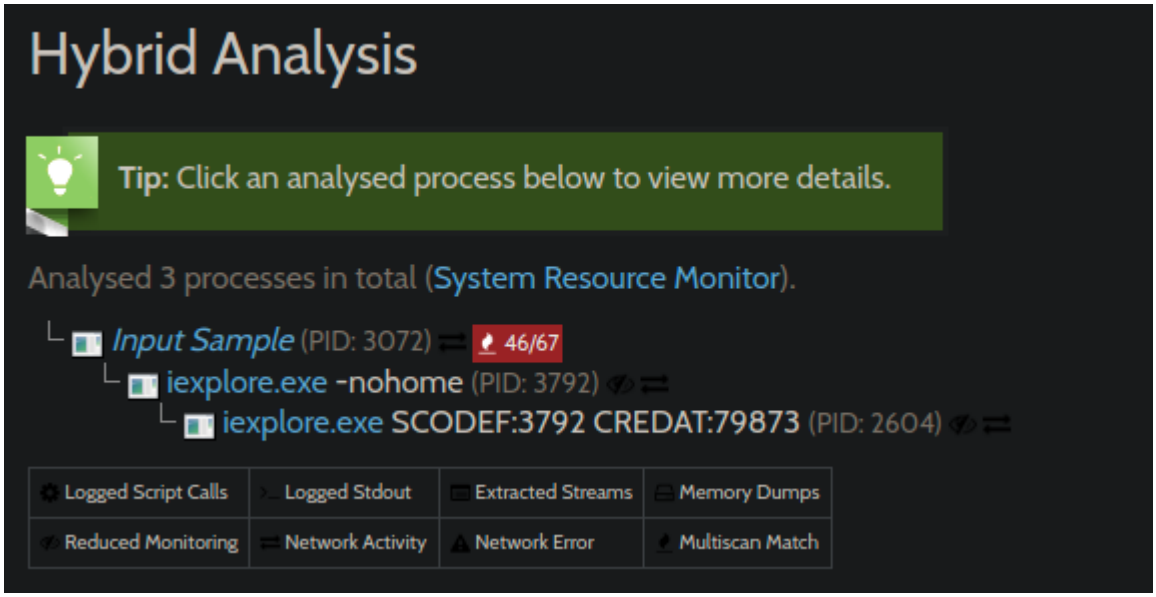


Figure 6: a screenshot from Hybrid-Analysis

Dynamic Analysis:

The malware can be executed normally by double clicking. Within 3-6 seconds the iexplore.exe will open depending on victim computer speed and the default browser after executing malware. And here is a screenshot of Process Hacker, while the malware is running:

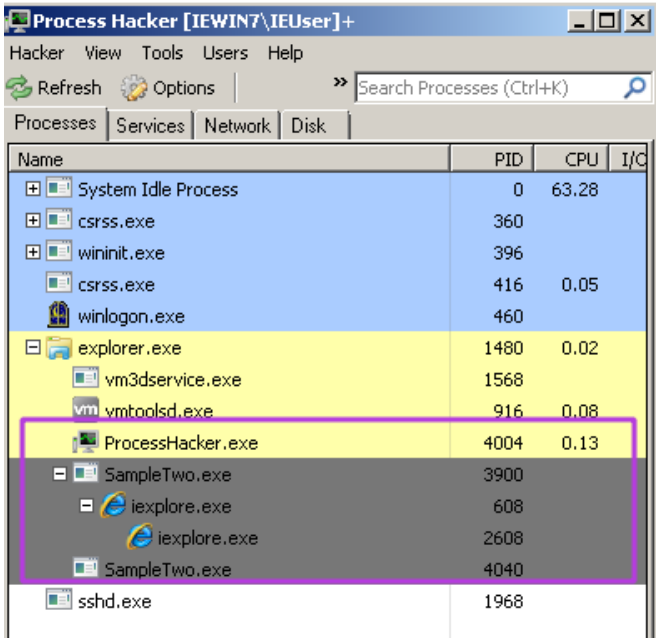


Figure 7: Process Hacker results while the malware is running.

With the process monitor and ProcDot more actions can be recognized from the malware.

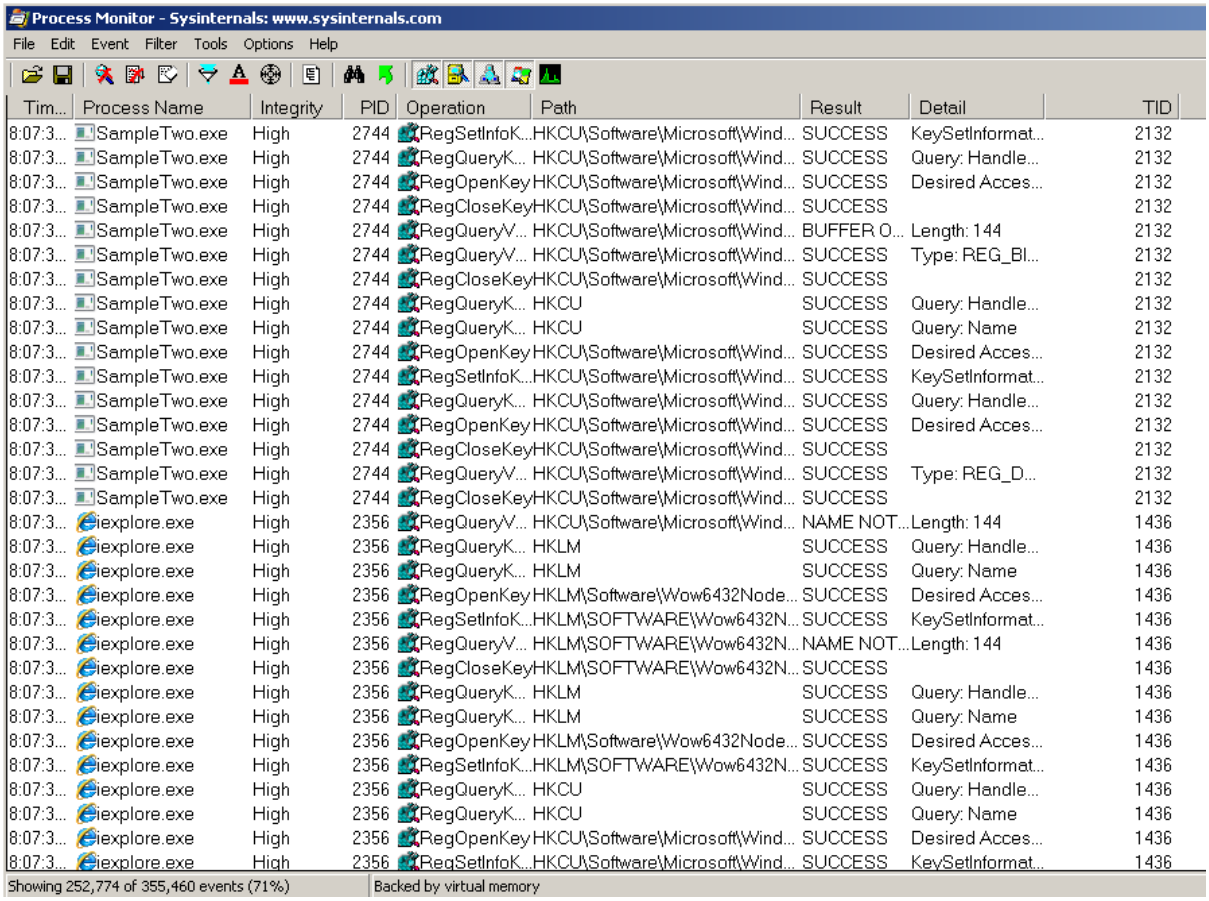
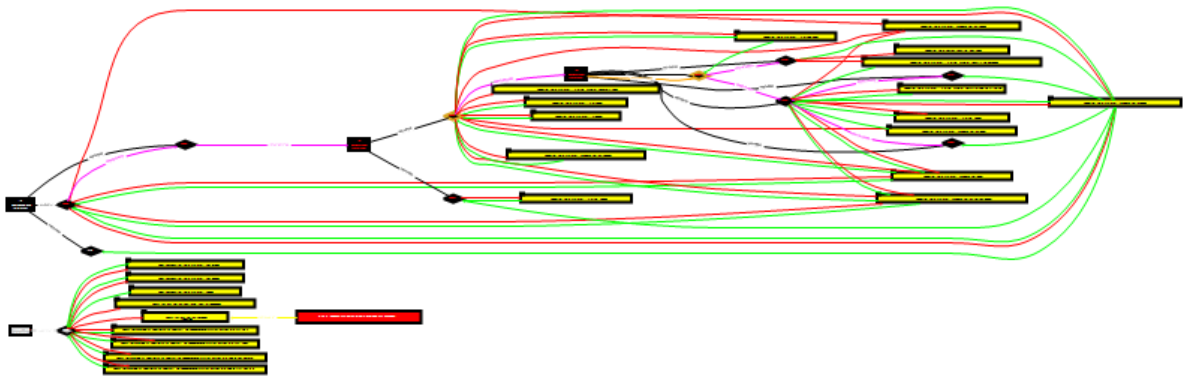


Figure 8: Process Monitor Process results while the malware is running.

Here is the graphical view of malware behaviour with ProDot. (From this you can get an idea about the malware behaviour. Even though it's not readable.)



Noticeable behaviours:

This malware also set some special directories. Some of those set up directories are listed below.

- C:\User\<UserName>\AppData\Local\Microsoft\Temporary Internet Files\
- C:\User\<UserName>\AppData\Local\Microsoft\Temporary Internet Files\Content.IE5
- C:\User\<UserName>\AppData\Local\Microsoft\Temporary Internet Files\History
- C:\User\<UserName>\AppData\Local\Microsoft\Windows\Cookies
- C:\User\<UserName>\AppData\Local\Microsoft\Windows\History\History.IE5

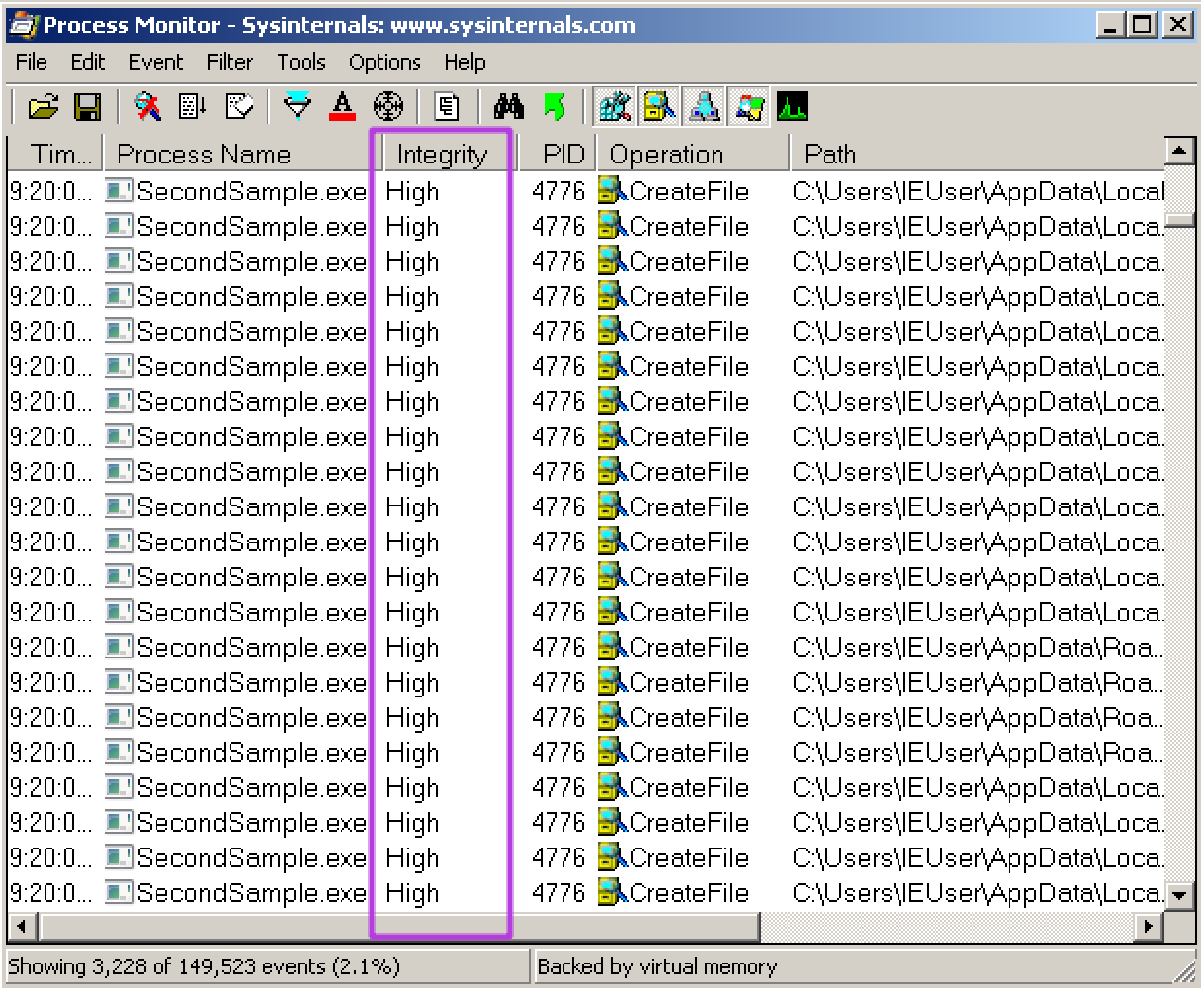
Get TickCount value is another key behaviour of this malware. This allows the malware to get the number of milliseconds that have elapsed since the system was started. After the sleep.

Interaction with the File System:

This malware interacts with the victim computer files system by creating, executing, finding, and removing files.

Created Files:

There are more than 40 files created by this malware. A screenshot of the ProcMon is shown below.



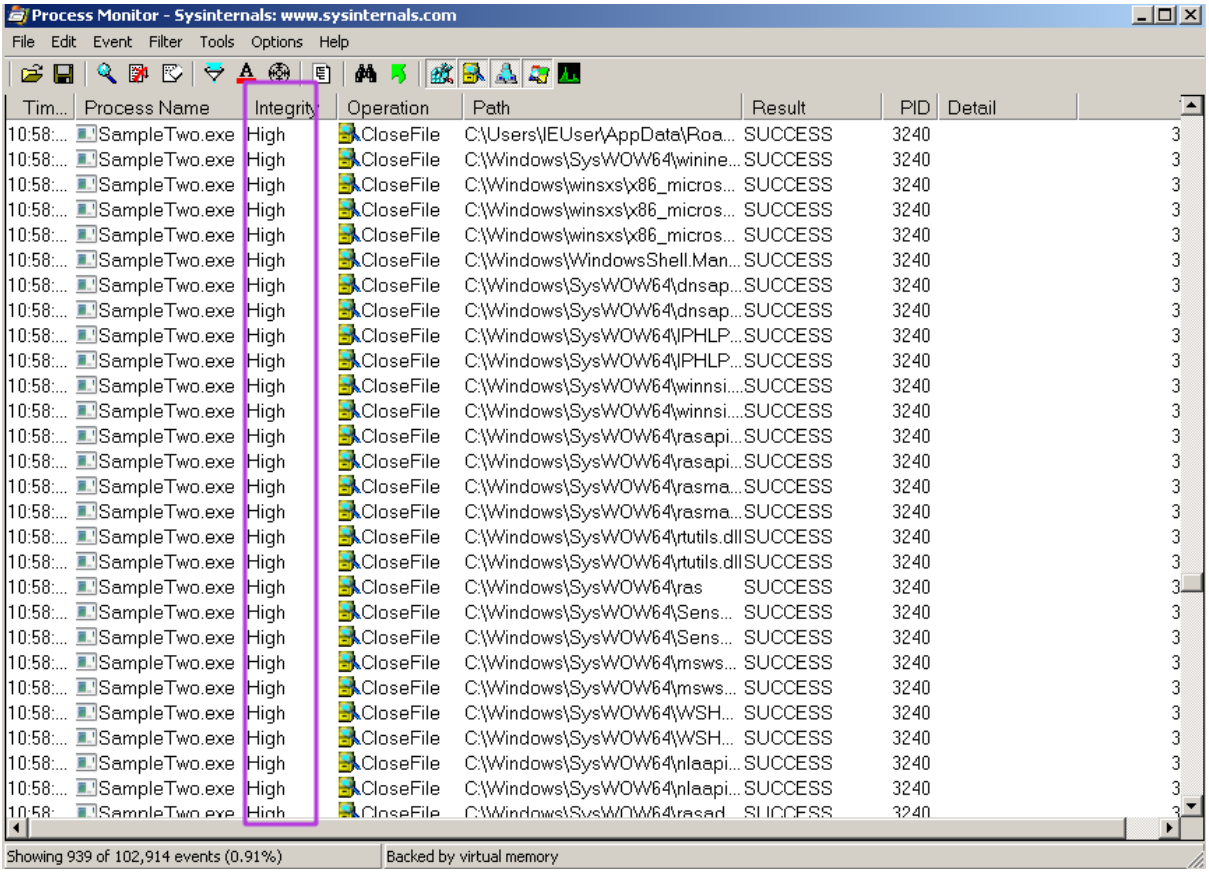
As this screenshot shows the malware interacts with highly confidential files. However, here are the most noticeable files.

- C:\User\<UserName>\AppData\Local\Microsoft\Temporary Internet Files\Content.IE5\C1Os62Ry\QDElURGZQAc
- %TMP%\SampleTwo(H).exe

And this last file which is SampleTwo(H).exe is an executable file.

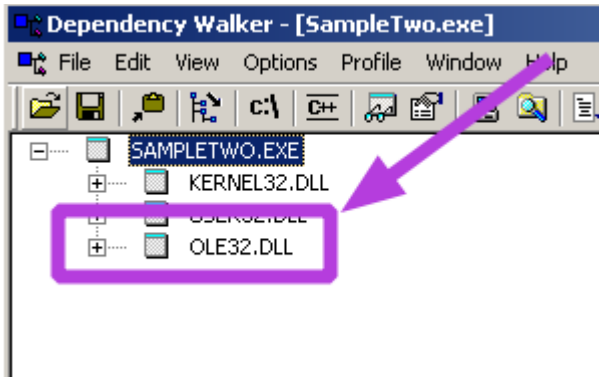
Removed and Closed Files:

As this report mentioned earlier almost every file that was created by the malware is deleted. For instance, the below snapshot will illustrate the files that are closed by the SmapleTwo.exe file.



Among these removed files C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\C1OS62RY\QDElURGZQAc is important. Because this where malware stored all the information about the victim before it sends.

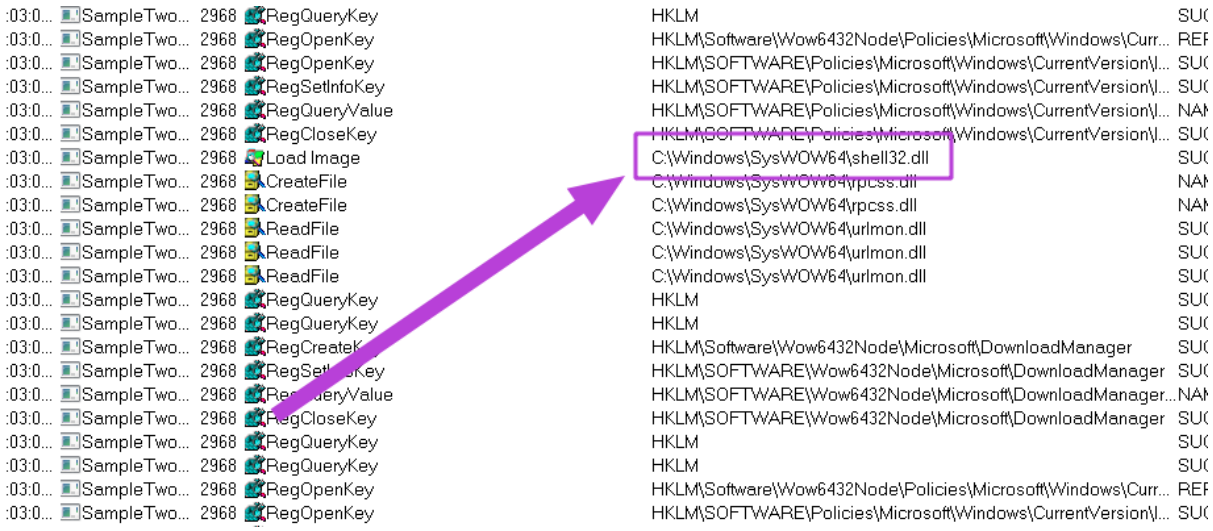
According to the Dependency Walker tool, this malware has ole32.dll.



This allows the malware to embed some object into a document or objects that were created by another process.

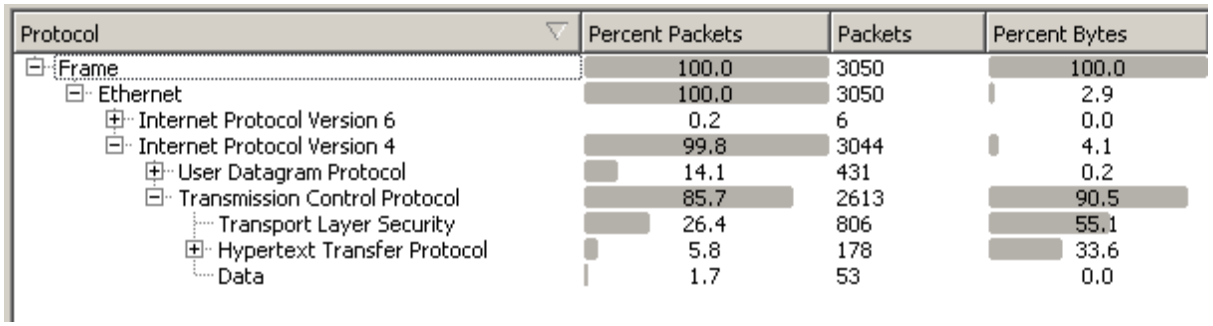
Network Communication:

There few network behaviours can be identified with this malware. This malware opens default web browser and access to some web pages automatically. Once it is executed as I mentioned before. For that this malware uses some dll such as shell32.dll:



According to the Wireshark:

- Almost 86% of network protocols that are used by the malware is TCP. HTTP 5.8% and interestingly enough 1.7% data has been captured that Wireshark cannot really understand.



Protocol	Percent Packets	Packets	Percent Bytes
Frame	100.0	3050	100.0
Ethernet	100.0	3050	2.9
Internet Protocol Version 6	0.2	6	0.0
Internet Protocol Version 4	99.8	3044	4.1
User Datagram Protocol	14.1	431	0.2
Transmission Control Protocol	85.7	2613	90.5
Transport Layer Security	26.4	806	55.1
Hypertext Transfer Protocol	5.8	178	33.6
Data	1.7	53	0.0

This malware makes a DNS requests. Among them are the following IPv4 addresses are rather suspicious.

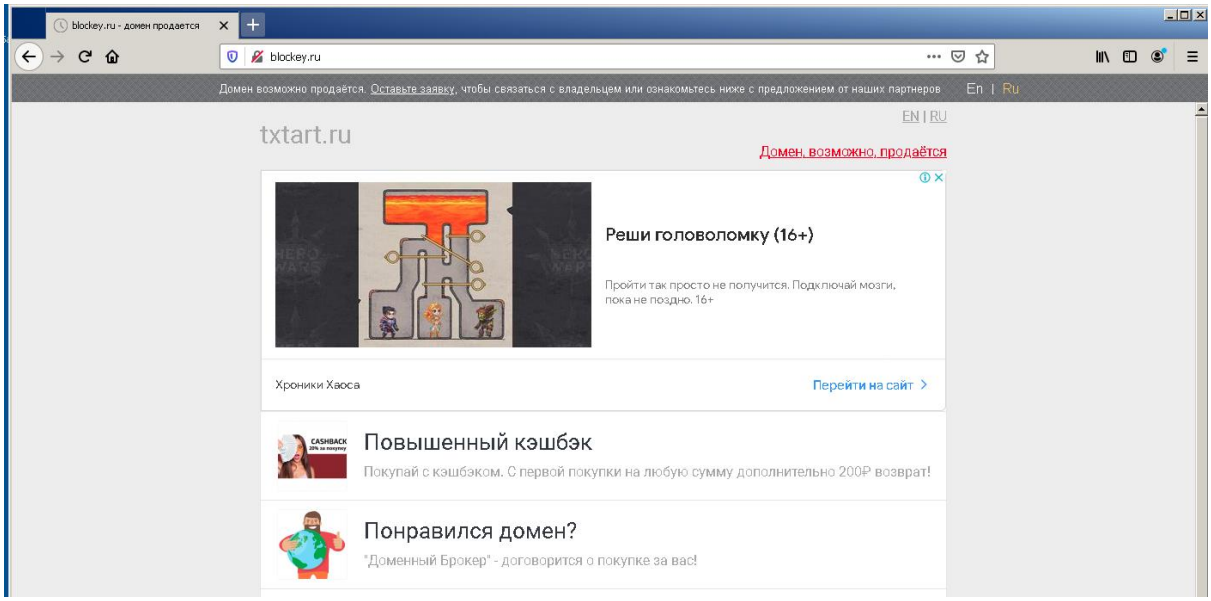
1. 31.31.205.163 → This web site domain name is http://ns1.domainparking.int.reg.ru/. This is based in Russia.

```
role:           Reg.Ru Network Operations
address:        Russia, Moscow, Vassily Petushkova st., house 3, Office 326
remarks:        NOC e-mail: noc@reg.ru
remarks:        User support: support@reg.ru
remarks:        SPAM reports: abuse@reg.ru
phone:          +7 (495) 580-11-11
fax-no:         +7 (495) 491-55-53
admin-c:        ARP-RIPE
tech-c:         ARP-RIPE
tech-c:         AH9460-RIPE
nic-hdl:        RGRU-RIPE
mnt-by:         REGRU-MNT
abuse-mailbox:  abuse@reg.ru
created:        2011-03-30T12:49:27Z
last-modified:  2014-12-23T12:18:22Z
source:         RIPE # Filtered
```

2. 31.13.90.6 → This IP address belong to Facebook, and server is based in Ireland.

```
organisation:   ORG-FIL7-RIPE
org-name:       Facebook Ireland Ltd
country:        IE
org-type:       LIR
address:        4 GRAND CANAL SQUARE , GRAND CANAL HARBOUR ,
address:        D2
address:        Dublin
address:        IRELAND
phone:          +0016505434800
fax-no:         +0016505435325
admin-c:        PH4972-RIPE
mnt-ref:        RIPE-NCC-HM-MNT
mnt-ref:        fb-neteng
mnt-by:         RIPE-NCC-HM-MNT
mnt-by:         fb-neteng
abuse-c:        RD4299-RIPE
created:        2011-04-07T13:16:29Z
last-modified:  2020-12-16T13:18:51Z
source:         RIPE # Filtered
```

3, 216.58.215.34 – this is another Russian based web server. Its domain name is blockey.ru. However, this is ip address belongs to GoDaddy.



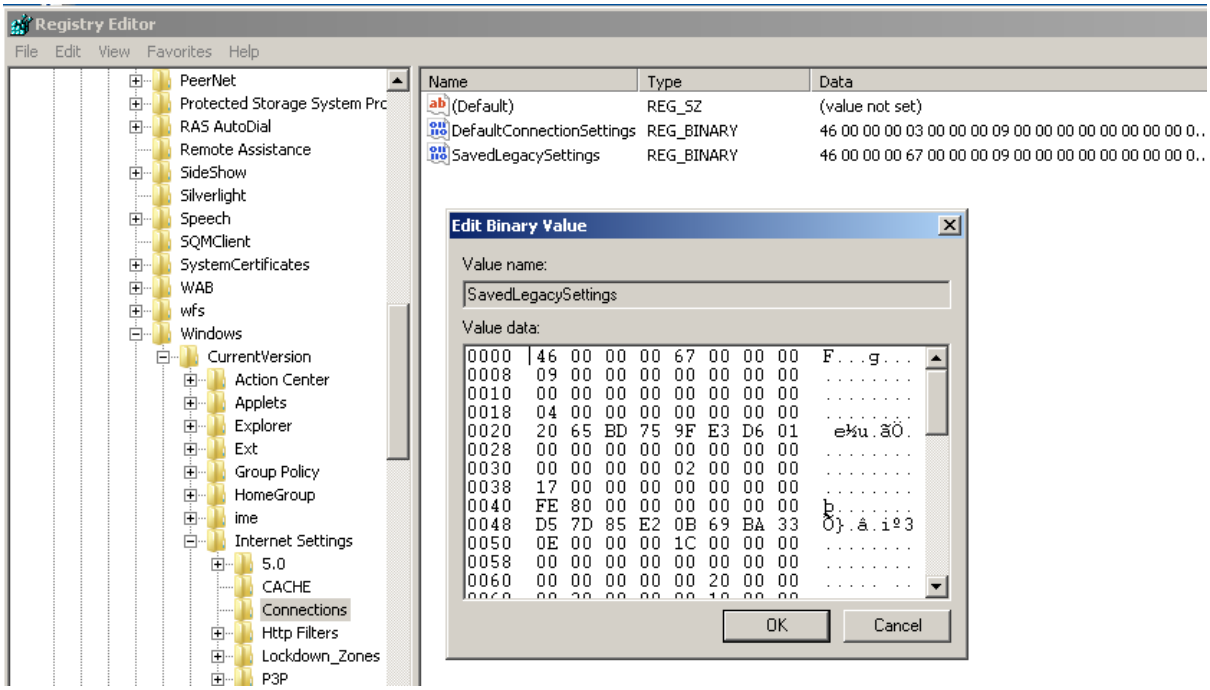
Among these IP addresses most of them are from either Russia or the USA. However, Some of the IP addresses now no longer work. And here are the countries that contacted by the malware according to the Hybrid-Analysis.


Contacted Countries

Registry Keys:

This malware sample also interacts with windows registry considerably. It deletes or modifies very important registries of the windows. For instance:

- This malware modifies “HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections”.

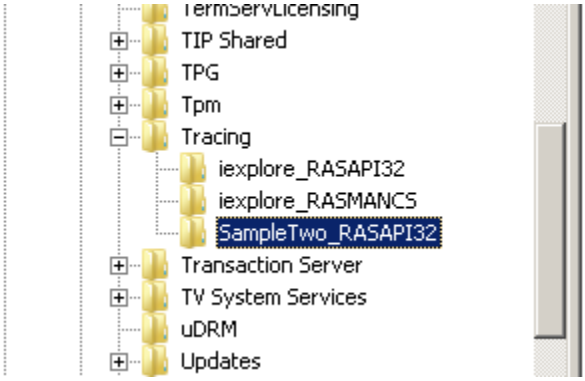


And it deletes.

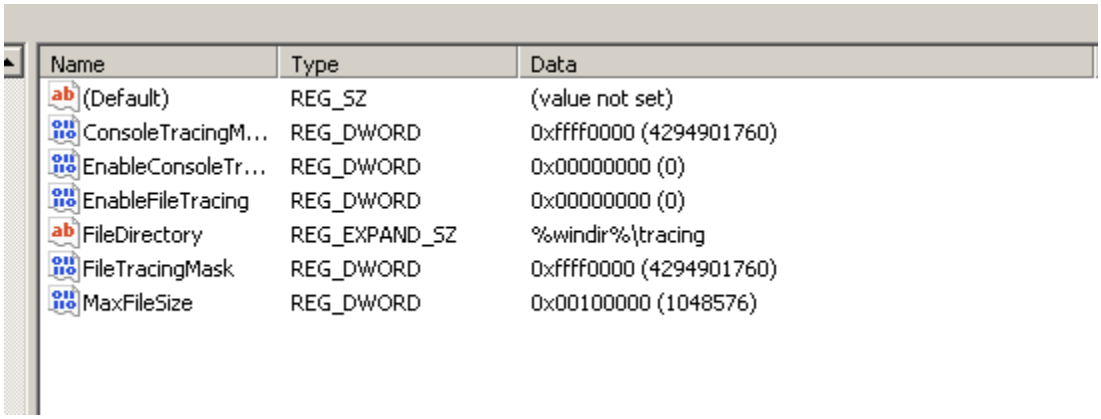
- HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer
- HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyOverride

These Registry keys are used to access the binary value of DefaultConnectionSettings which has information about proxy configuration. This malware removes if the computer uses any proxy or VPN connections.

Moreover, another noticeable behaviour of this malware windows registry. Is that it creates a sub registry in HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Tracing.



They registry has quite values as well. Thus, this reg key is used by malware itself to tracing file once it wakes up from a long sleep.



Name	Type	Data
(Default)	REG_SZ	(value not set)
ConsoleTracingM...	REG_DWORD	0xffff0000 (4294901760)
EnableConsoleTr...	REG_DWORD	0x00000000 (0)
EnableFileTracing	REG_DWORD	0x00000000 (0)
FileDirectory	REG_EXPAND_SZ	%windir%\tracing
FileTracingMask	REG_DWORD	0xffff0000 (4294901760)
MaxFileSize	REG_DWORD	0x00100000 (1048576)

Detection Rules

YARA Rules

Unlike the first sample. This malware contains lots of unique strings. However, in order to detect this malware, sample some unique techniques have been utilized. Therefore, following Yara rules has the ability to identify this malware very accurately.

```
import "pe"

private global rule IsPE
{
   condition:
      uint16(0) == 0x5A4D and
      uint32(uint32(0x3C)) == 0x00004550
}

private global rule FileSize
{
   condition:
     filesize > 190KB
}

private rule EntryPoint
{
   condition:
     pe.entry_point == 0x00006F0D
}

private rule FileSections
{
        condition:
                for any i in (0..pe.number_of_sections-1) : ( pe.sections[i].name == ".reloc" )
}

rule SampleTwo : Spyware
{

        meta:
                author = "Dakshitha Perera"
                description = "Yara rules for malware Sample Two"
                data = "02/01/2020"

        strings:
                $MainWebsite = "www.babla.ru"
                $IpAddress1 = "41.34.22.68"
                $IpAddress2 = "38.45.27.68"
```

```
        condition:
                all of them and IsPE and FileSize and FileSections
}
```

First, the pe module has been imported to this rule. Because pe.entry_point feature has been used in this rule. The first rule is to identify all the executable files in the given location.

```
private global rule IsPE
{
   condition:
      uint16(0) == 0x5A4D and
      uint32(uint32(0x3C)) == 0x00004550
}
```

Furthermore, this rule has made global rule. Which gives the ability to impose restriction in all rules at once.

Second rule checks the file size. Normally this file is 195KB. However, if the malware comes as a Trojan and the file size should definitely be increased. That is the reason why this rule checks more than 190KB size of files.

```
private global rule FileSize
{
   condition:
     filesize > 190KB
}
```

With the third rule, Yara checks file entry points. Pe Yara module has been imported for this rule.

```
private rule EntryPoint
{
   condition:
     pe.entry_point == 0x00006F0D
}
```

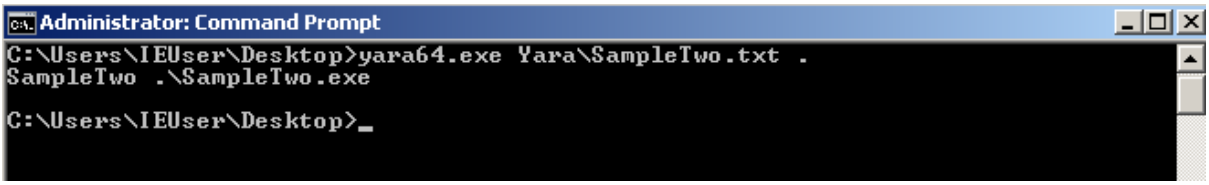
Moreover, this Yara rule is a private rule. When the private rules are used, Yara does not report other file matches.

Finally, it checks some unique strings such as the webpages:

```
rule SampleTwo : Spyware
{

        meta:
                author = "Dakshitha Perera"
                description = "Yara rules for malware Sample Two"
                data = "02/01/2020"

        strings:
```

```
        $MainWebsite = "www.babla.ru"
        $IpAddress1 = "41.34.22.68"
        $IpAddress2 = "38.45.27.68"

    condition:
        all of them and IsPE and FileSize and FileSections
}
```

And the final condition made the Yara to search an executable file which has the entry point of 0x0006F0D, and size is more than 190KB and babla.ru string included.



Snort Rules

The main network communication of this malware, which is the HTTP request http://discriminate.blockey.ru/ is fine way to detect this malware using snort rules.

```
var MALIP[185.80.54.15,185.53.178.7]

alert TCP any 80 -> $MALIP 1:1024 (flags: S; msg: "Code Read: SYN packet to a
malicious website; \
reference: "discriminate.blockey.ru/")
```

Above mentioned snort rule will check for any network communication to that malicious website. Moreover, this rule looks for SYN flags. Thus, it can detect the malware at the very early stage. Before it makes any connection to this malicious web site.

Detection and Malware Removal Instructions

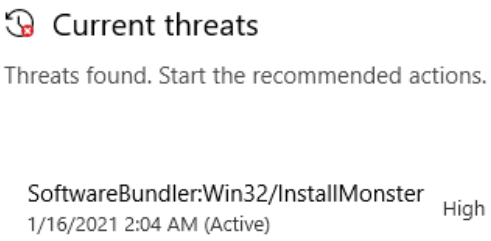
This malware can be identified as another type of Spyware. This malware writes victim's sensitive data to remote processes without users' knowledge. Nonetheless, the only detection that can be seen by a non-technical person is opening the web browser and accessing the http://discriminate.blockey.ru/ website automatically. However, there are so many actions going on behind the scenes as this report mentioned during the Malware Behaviour part.

This spyware can reach the computer through two main ways.

1. Via freeware or sharewares. Downloading and installing freeware to the computer is always risky. Most freeware and cracked versions are patched with malicious code. Therefore, avoiding freeware, sharewares and cracked wares is the best way to avoid this malware.
2. Infected websites are another way of this malware reaching your computer. And also, social engineering could be involved in this way.

This malware can easily be detected by default windows anit-virus software which is Windows Defender. Once the Windows is infected by this malware windows defender shows a warning popup like this.

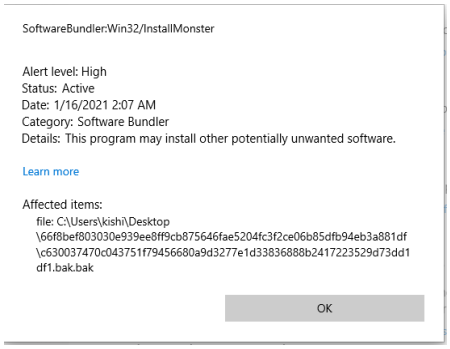
And when the system is scanned with Windows Defender. The threat can be found:



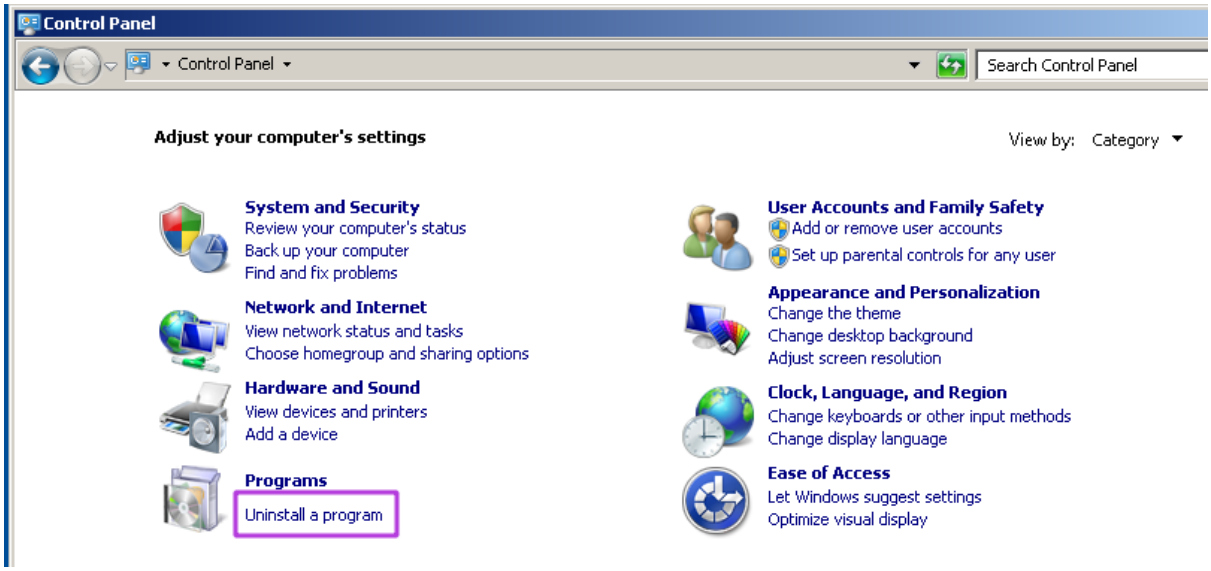
Therefore, enabling and updating the windows defender or any other antivirus software could stop such infections. Moreover, updating your windows and default browser could also help to avoid or reduce the damages.

There are few measures that can be taken in order to remove this malware from the system.

1. Turn on the Windows Defender or use any other Antivirus programs and scan the whole file system. The windows defender is capable of detecting and removing the malware automatically.



2. Uninstall unwanted Programs from the computer. - Navigate to the control panel on the windows machine. And then go to the Uninstall program.



After that you can delete unwanted programs from your computer.

3. Clear whole browser data. If you are using chrome browser. You can reset chrome by navigating to three dots at the top right  and navigate to the setting on the pop up. After that go to Advanced and click Reset and Clean-up. And it will reset the whole browser.
4. Or this python program can be used to remove the malware from the device.

```
# Autor: Dakshitha Perera
# Data : 19/01/2021
# Title : program to remove Malware Sample Two.
from winreg import *
import tempfile
import psutil
import sqlite3
import winreg
import os
import re


def ProcessMonitoring():
        tempDir = tempfile.gettempdir()
```

```
        malProcess = "svchost.exe"
        for proc in psutil.pids():
                p = psutil.Process(proc)
                if(p.name()) == malProcess:
                        try:
                                Procfiles = p.open_files()
                        except:
                                # print("couldn't find the malware process on the
running processes")
                                continue
                        for root, dirs, files in os.walk(tempDir):
                            for file in files:
                                searchFiles = re.compile(r'.*[.]txt')
                                matches = re.findall(searchFiles, file)
                                if matches:
                                    if(matches[0][-11:] == "(H).txt.txt"):
                                        malwareRealName = matches[0]
                                        print("The malware name is
{}".format(malwareRealName))
                                        try:
                                                p.kill()
                                                print("Running Malware Processes PID
{}was killed".format(proc))
                                                os.remove(os.path.join(tempDir,
malwareRealName))
                                        except:
                                                print("Couldn't kill the Process!!!")

def ChromeHistryDelete():
        username = os.getlogin()

        db = sqlite3.connect('c:\\Users\\{}\\AppData\\Local\\Google\\Chrome\\User
Data\\Default\\History'.format(username))
        c = db.cursor()

        URLPattern = re.compile(r"https?://([^/]+)/")
        domains = {}

        result = True
        id = 0
        while result:
            result = False
            ids = []

            for row in c.execute('SELECT id, url, title FROM urls WHERE id > ? LIMIT
1000', (id,)):
                result = True
                match = URLPattern.search(row[1])
                id = row[0]

                if match:
                    domain = match.group(1)
                    domains[domain] = domains.get(domain, 0) + 1
                    # clean if this is true
                    if "discriminate.blockey.ru" in domain:
                        ids.append((id,))
```

```
            c.executemany('DELETE FROM urls WHERE id=?', ids)
            db.commit()

        db.close()

def ClearChromeCookies():
        username = os.getlogin()

        db = sqlite3.connect('c:\\Users\\{}\\AppData\\Local\\Google\\Chrome\\User
Data\\Default\\Cookies'.format(username))
        c = db.cursor()

        result = True
        while result:
                result = False
                ids = []

                for row in c.execute('SELECT creation_utc, host_key, name FROM
Cookies'):
                        id = row[0]
                        if row[1] == '.discriminate.blockey.ru:
                                ids.append((id, ))

                c.executemany('DELETE FROM Cookies WHERE creation_utc=?', ids)
                db.commit()

        db.close()

def ClearChromeCache():
        username = os.getlogin()

        db = ('C:\\Users\\{}\\AppData\\Local\\Google\\Chrome\\User
Data\\Default\\Cache'.format(username))

        for root, dirs, files in os.walk(db):
            for filename in files:
                pattern = re.compile(r'^f.')
                matches = pattern.finditer(filename)
                for match in matches:
                        os.remove('C:\\Users\\{}\\AppData\\Local\\Google\\Chrome\\User
Data\\Default\\Cache\\{}'.format(username, filename))
                        print("Deleting {}".format(filename))

def DeleteRegistry():
        keyVal = r"SOFTWARE\Wow6432Node\Microsoft\Tracing"

        aKey = OpenKey(HKEY_LOCAL_MACHINE, keyVal, 0, KEY_ALL_ACCESS)
        try:
            i = 0
            while True:
                asubkey = winreg.EnumKey(aKey, i)
                # print(asubkey)
                searchReg = re.compile(r'^(?!IpHlpSvc|iexplore|setup|Squirre).*')
                matches = re.findall(searchReg, asubkey)
                if matches:
```

```
                        GuessingMalwareRealName = matches[0].split("_")[0]
                        print("The malware name is
{}".format(GuessingMalwareRealName))
                        userInput = input("Would you take it and delete the
regkey(y/n) : ")
                        if userInput == "y":
                                print("regkey will be delete shortly!! ")
                                RegValue = str(asubkey[0])
                                RegData = str(asubkey[1])
                                winreg.DeleteValue(hKey, RegValue)
                                break

                        else:
                                print("unable to delete Malware Registry!!!")
                i += 1
        except:
                pass

ChromeHistryDelete()
ClearChromeCache()
ClearChromeCache()
DeleteRegistry()
ProcessMonitoring()
```

Running Sheet

Action	Description	Result	Justification
Create the secure virtual environment.	Download the virtual windows 7 ISO file.	Successfully downloaded the windows 7 iso file.	Most of the malware target windows OS. Thus, it always to better to use windows 7 rather that windows 10. Due to its lack of security and resources.
	Install windows 7 on VMWare.	Successfully install windows 7 on the VMware.	To isolate the virtual machine from the host machine.
	Disable copy & paste options on the VMware	Secure the virtual environment.	
	Disable drag & Drop option on the VMware.		
	Encrypted the virtual machine.		
	Download flare VM.	Successfully downloaded it	To get some useful tool to the VMware.
	Take a snapshot of the clean Virtual machine	Successfully took the snapshot	Just in case if the virtual machine crashes.
Static Analysis.	Uploaded the malware all two malwares to the virustotal.com and hybrid-analysis.com	Detected the malware by both virustotal and hybrid analysis.	To check if any anti-malware engine detects this malware.
	Uploaded both malware to the DIE.	Found that both malwares are packed. (according to the DIE results.)	To check whether the malwares are packed or not.
	Uploaded both malware to the RDG detector	Found that no malware is packed.	
	Uploaded both malware to the PEiD	Found that one malware is packed, and other malware is not.	

	Used PortEx. 	Found that some parts of both malwares are packed.	
	Uploaded both malware to the PE Studio	Found few useful information about both malwares.	To obtain some information about the malware.
Unpacking the malware.	Uploaded to x32dgb, Ollydbg and IDA	Found the tail jump.	To find the tail jump.
	Checked or any anti-debugging APIs	Found that both malwares have IsDebuggerPresent API.	To go through any obfuscation techniques.
	Tried patch the malware once it is using IsDebuggerPresent Option.	Successfully, patched the malware.	
	Tried dump the unpacked file. Using Ollydbg and a its plugging called DumpOlly.	Successfully, Dumped the unpacked file.	To get the unpacked malware file.
	Tried to fix PE header and import table using Scylla software.	Successfully, Fixed the PE header and import table.	
Dynamic Analysis.	First run both malware on ANY.RUN sandbox.	Run both malwares without any errors. And show some behaviours.	To figure out what the malwares does.
	After realizing these are adware, I run both on virtual machine.		in order to more and detailed behaviours of samples.

Reverse Engineering.	Uploaded to IDA.	Obtained the assembly code of the malware.	To understand what the malware does.
	Uploaded to the Ghidra	Successfully, Decompiled the code.	
Writing the report.	Started writing the report.	Wrote the report.	To represent finding with these malwares.

References:

1. Yason, M. V. (2007). The art of unpacking. Retrieved Feb, 12, 2008.
2. Peter, F. (2011). The Ultimate Anti-Debugging Reference
3. Kendall, K., & McMillan, C. (2007, August). Practical malware analysis. In Black Hat Conference, USA (p. 10).
4. Sikorski, M., & Honig, A. (2012). Practical malware analysis: the hands-on guide to dissecting malicious software. no starch presses.
5. Sharif, M., Lanzi, A., Giffin, J., & Lee, W. (2009, May). Automatic reverse engineering of malware emulators. In 2009 30th IEEE Symposium on Security and Privacy (pp. 94-109). IEEE.
6. Debray, S., & Patel, J. (2010, October). Reverse engineering self-modifying code: Unpacker extraction. In 2010 17th Working Conference on Reverse Engineering (pp. 131-140). IEEE.