Report Malware Analysis

Jorge Amorós Anaya

Report Jorge Amoros

Introduction

In this document, we will track an ongoing malware campaign that is currently going on in Italy. This campaign started on his first stage on the 1st of October and keeps its activity on the day we are writing this 23 of November.
The malware Ursnif / Gozi in Italy is spread with an INPS themed campaign. The text of the message is poorly written, but it is still dangerous. The xls attachment contacts specific URLs from which a DLL is downloaded infecting the victim's PC. This attack is specifically aimed against the country. The DLL can only be downloaded from Italian IPs, even if sometimes the links are reachable from IPs outside in other countries.

Tracking the campaign

For the tracking of this campaign, we are following the tweets of @JAMESWT_MHT. The chain of tweets follows the following pattern :

- First Tweet chain (1 October)

  https://twitter.com/JAMESWT_MHT/status/1311546809710456833

- Second tweet chain (19 November)
  https://twitter.com/JAMESWT_MHT/status/1329249720997457922?s=20

- Third tweet chain (23 November)

  https://twitter.com/JAMESWT_MHT/status/1330757390753476609
  https://twitter.com/JAMESWT_MHT/status/1330761150783516672

Several samples can be found on the tweets and in the following links.

Different Excel files
- https://bazaar.abuse.ch/browse/tag/pw%20mise/
- https://bazaar.abuse.ch/sample/be6a92f7d1f695d18ba9e0661a1fdc3a440a7b87443cfe02a92d3f88ff08cf2c/

Different DLL files
- https://bazaar.abuse.ch/browse/tag/pw%20mise/
- https://bazaar.abuse.ch/sample/e9b8536f66aa5222f1979fea40b25b83f2acb487a0ab61a76378a2128efc0420/

Attack stages and Entry point

Report Jorge Amoros

This malware is distributed via Email, which has a password protected Excel document. When the victim opens it and enables the macros then the document will download a malicious .dll which will after that start the attack.

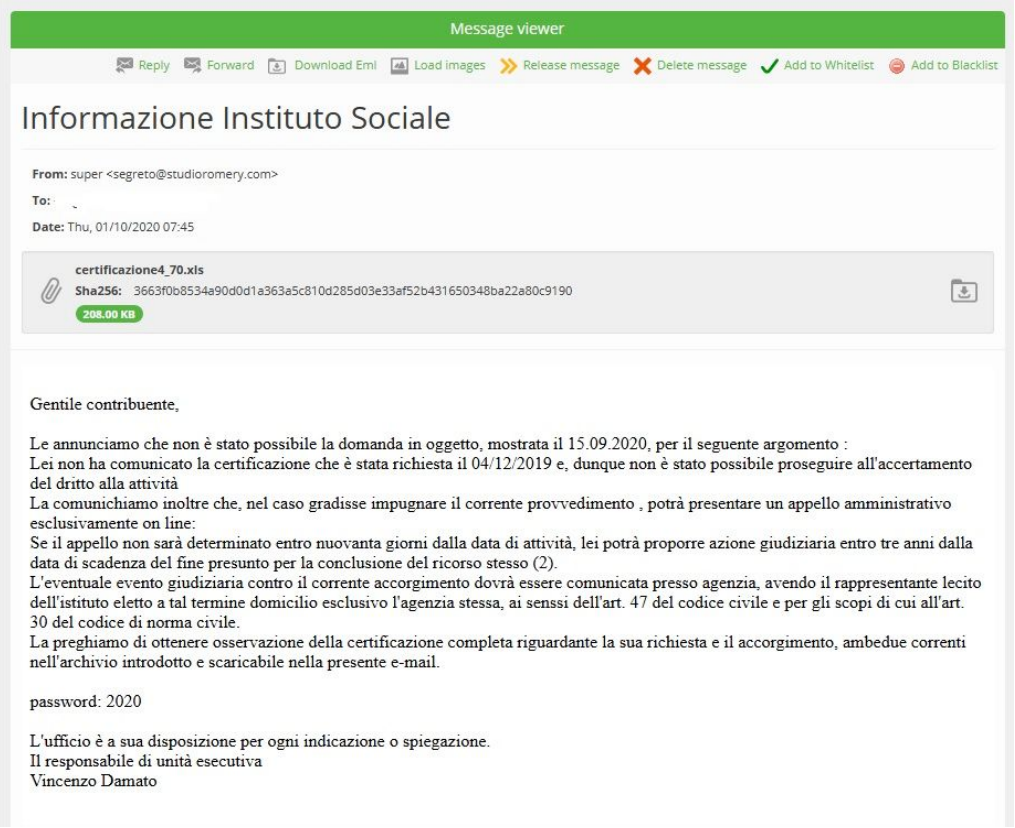

Figure 1 Example of the email

After that, the DLL is executed. We will run this excel into a machine anyRun to see the behaviour and the data that is downloading and what another kind of behaviour is happening.

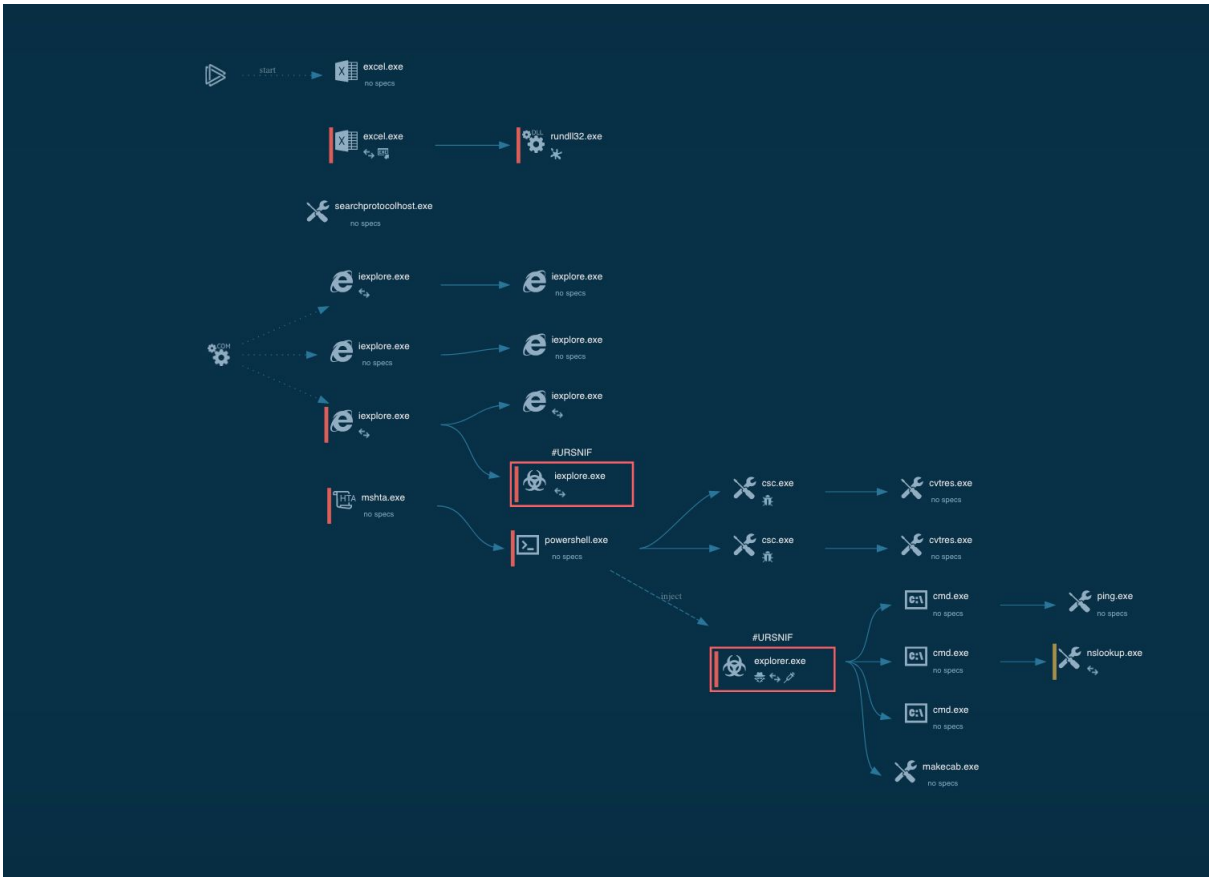

Figure 2. Tree Structure of the attack

As we can see the entry point is the excel file which will execute some macros and download and execute the malware which will at the same time keep downloading and spreading on the machine.

Setting the environment

We will mainly use two environment isolation tools :

- Remmux. REMnux is a Linux toolkit for reverse-engineering and analyzing malicious software. REMnux provides a curated collection of free tools created by the community. Analysts can use it to investigate malware without having to find, install, and configure the tools.
- Windows Virtual Machine with VMware. This tool is a fully customizable, Windows-based security distribution for malware analysis, incident response, penetration testing, etc.

Static analysis

1. Virustotal check

We will upload the malware to virus total.

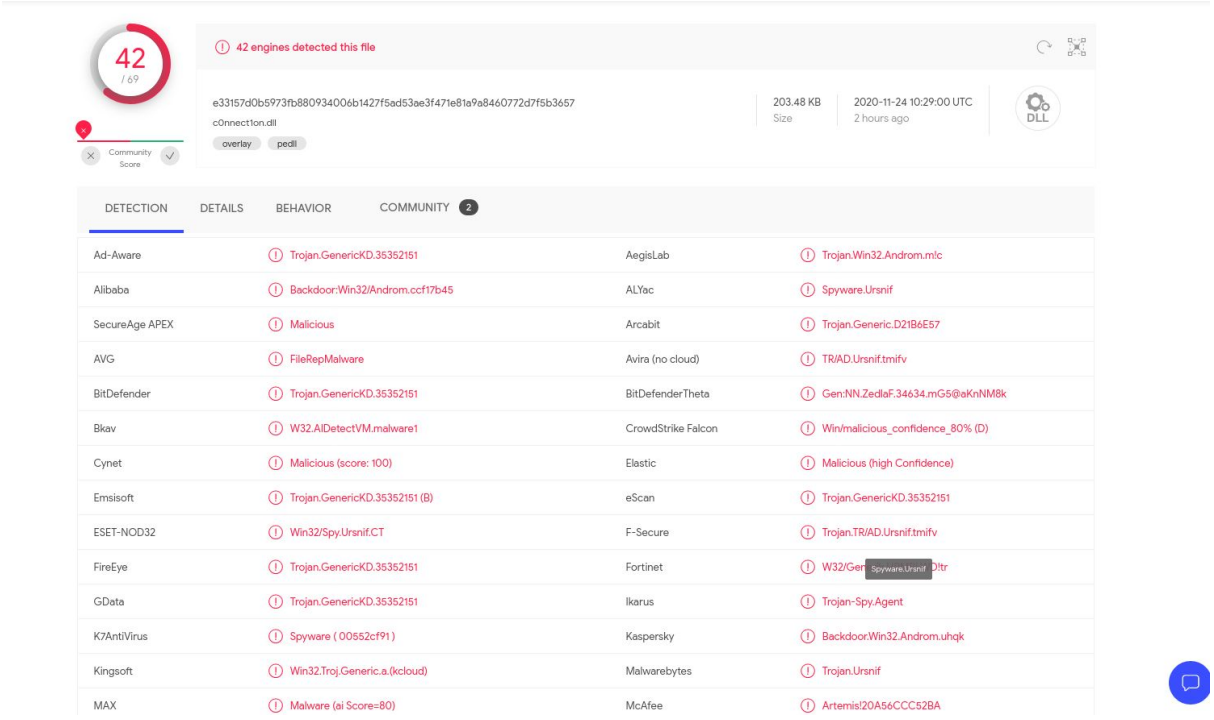

Figure 3. Virustotal scanner

Many of the scanners can track the sample as a malicious one.

2. Determining the file type

First and foremost step of malware analysis is understanding the file type. This can be achieved in many ways ranging from reading the magic bytes in hex-dump to using automated tools like CFF explorer. Also webs like virus total can give us this information.

The best way to tell if a file is truly a Windows PE is to open it up with a hex editor and inspect its “Magic Bytes”. The first few bytes of a file is how most libraries determine what format the file is.

First, we will notice the first two bytes are “4D 5A” or “MZ” (i.e. Coined after the initials of Mark Zbikowski, one of the developers for the PE format). We can also see the text of “This program cannot be run in DOS mode”

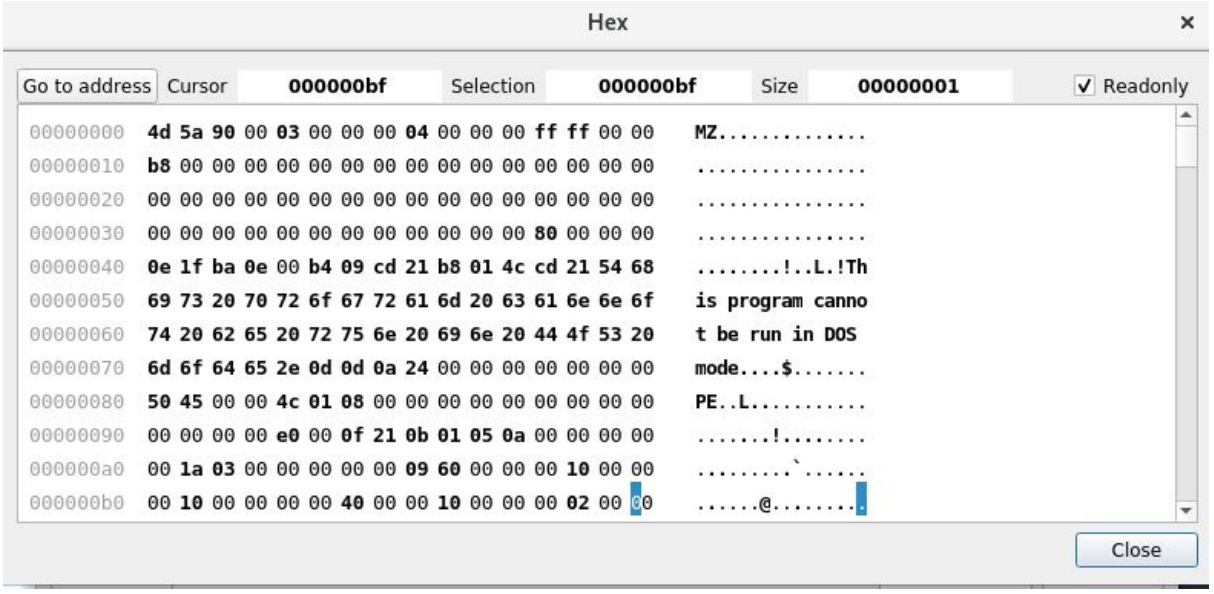

Figure 4. Hexadecimal Value 4d 5a or MZ

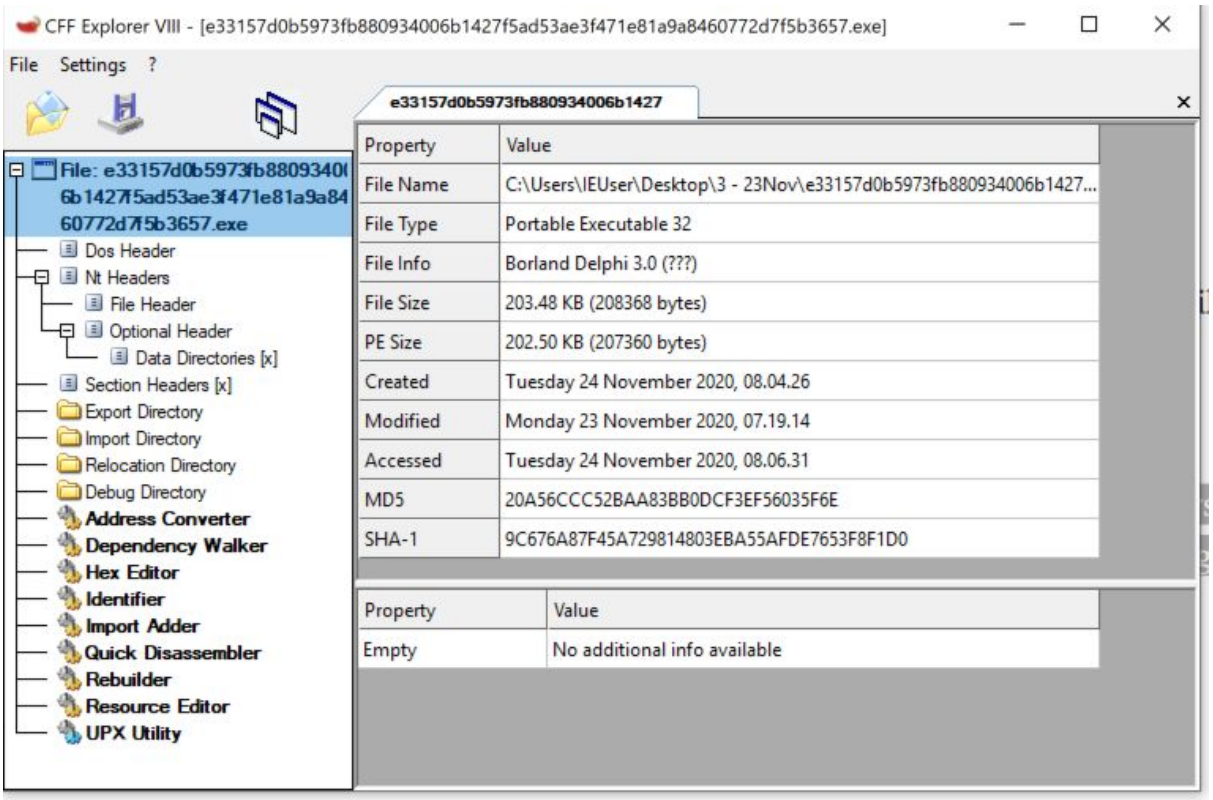

Figure 5. CFF shows Portable Executable 32

3. Determining file obfuscation by examining PE sections.

PE(Portable Executable) section table helps us to determine the type and attributes of different sections in a PE. PE section table has the following fields:

- .text or CODE : Contains executable code
- .data or DATA : Contains read/write data and global variables
- .rdata : Contains read-only data
- .idata : If present, contains the import table. If not present, then the import information is stored in the .rdata section.
- .edata: If present contains export information. If not present, then the export information is found in the .rdata section.
- .rscr: This section contains the resources used by the executable such as icons, dialogues, menus, strings, and so on.

Malware authors use several cryptographic algorithms to obfuscate code, making static de-compiling and understanding more complicated. But in these scenarios, there must be some traces of cryptographic Microsoft API's or some unpacking or decrypting stub. Using these left-overs, these tools detect the type of cryptographic algorithms used.

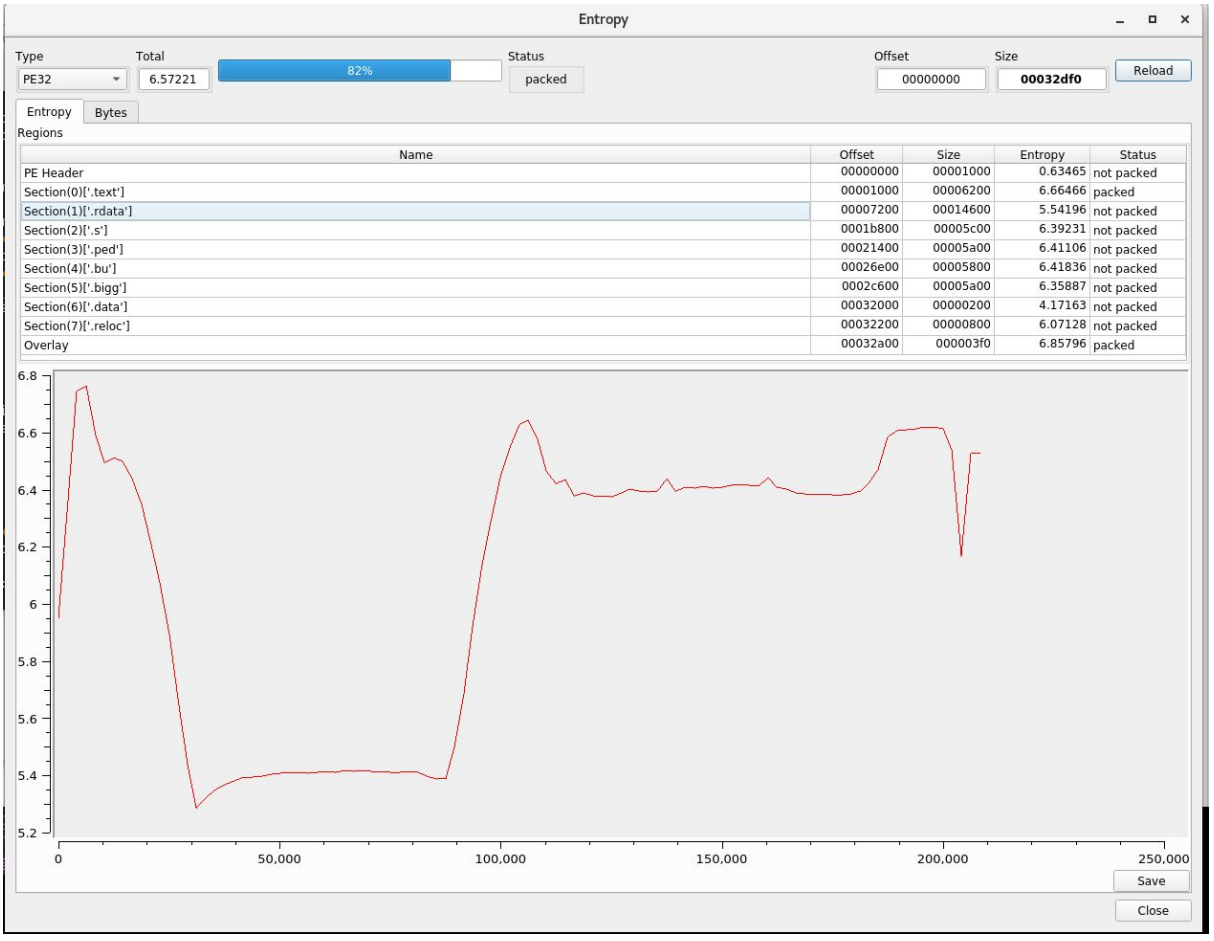

Figure 5. Die analysis of obfuscation

We can say it is packed since it's close to 7. At least the text data which is the important one. Also, overlay. Overlay is data appended to the end which may contain more information for the malware.

We should also check sizes. Typically, **raw-size** and the **virtual-size** should be almost equal, but small differences are normal due to section alignment. But in extreme scenarios, like a case where raw-size is 0 but the virtual section is far bigger than this value(usually seen with UPX Packer), indicates that this section will not take up space on the disk, but virtual-size specifies that, in memory, it takes up more space

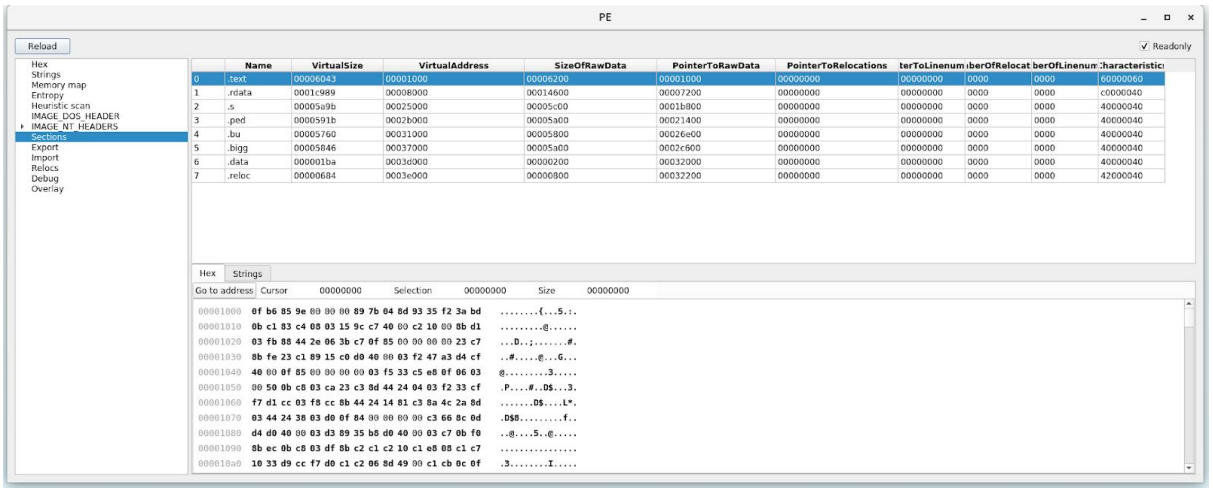

Figure 6. Die analysis of obfuscation

We don't know how it was packed. It's not UPX.

4. Imports

We will try to analyse the nodules and functions that in importing the program so we can guess what will happen when we execute it.

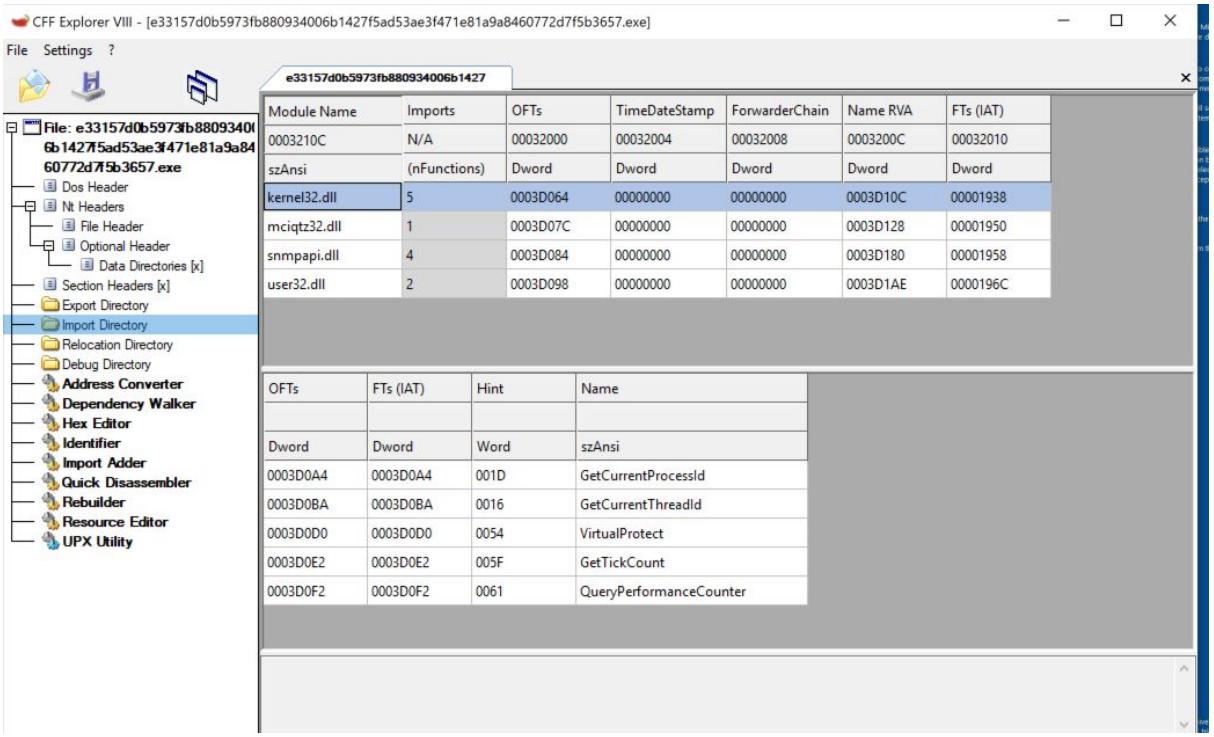

Figure 7. Imports on CFF explorer

When an attacker creates a malware that imports functions on-demand or delayed importing, the import table will usually contain the “Kernel32.dll” library and the functions “LoadLibrary” and “GetProcAddress”. This is not the case. But we can see other interesting functions :

- VirtualProtect: This function is used to change the protection of a region of memory. Malware may use this function to change a read-only section of memory to an executable.

- getTickCount() and QueryPerformanceCounter(). Which is usually called by malware to check for idle state. If the GetTickCount function returns a value that is too small the malware takes a branch that leads directly to a process exit. *QueryPerformanceCounter* reads the performance counter and returns the total number of ticks that have occurred since the Windows operating is running.

  This is an effective means of tracking the system’s uptime, providing the malware binary with an insight into the duration for which the system has been running. This could be an anti-sandboxing mechanism.

- createWindow(). Creating a window to interact with the user.

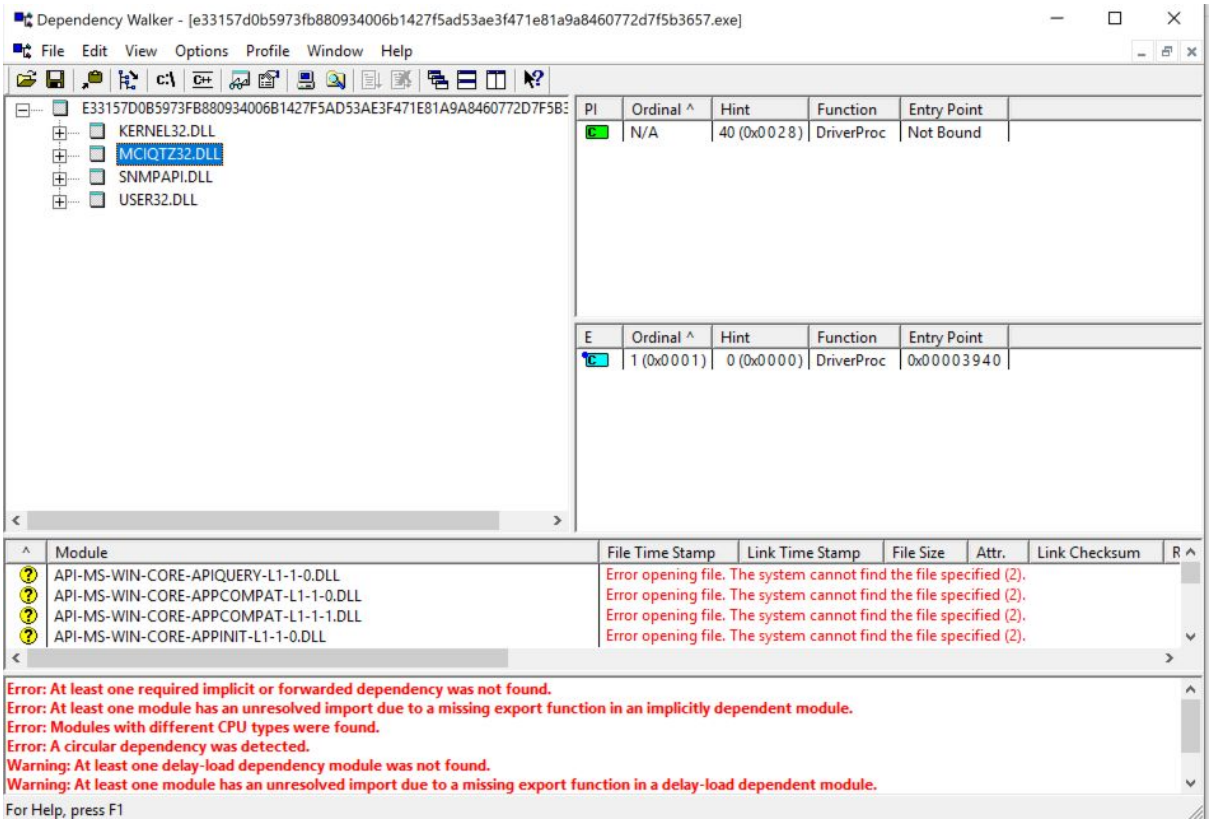

Figure 8. Dependency walker

Most often malware writers use dynamic linking in their code. For example, with the use of the tool Dependency Walker we can see this.

The file snmpapi.dll contains several utility functions that are used by Windows applications when communicating with network devices using SNMP (Simple Network Management Protocol). SNMP is used to perform remote administration of network hardware such as Routers and Hubs.

5. Exports

Exports are functions that a DLL/EXE may “export” or share that allow other programs to leverage.

Tetrapteron
Ductilimeter
Bisext
Acropathy
Mormyrid

Cypressed
Sorbinose
Ceramiaceae
Fanciable
Acyetic
Allotropicity
Ambuscader
Dyotheletian
Uncorrigibleness
Lather
Byzantinis
...

Very random plant names to hide real intentions.

6. Strings

For this, we use the tool FLOSS that uses advanced static analysis techniques to automatically deobfuscate strings from malware binaries.

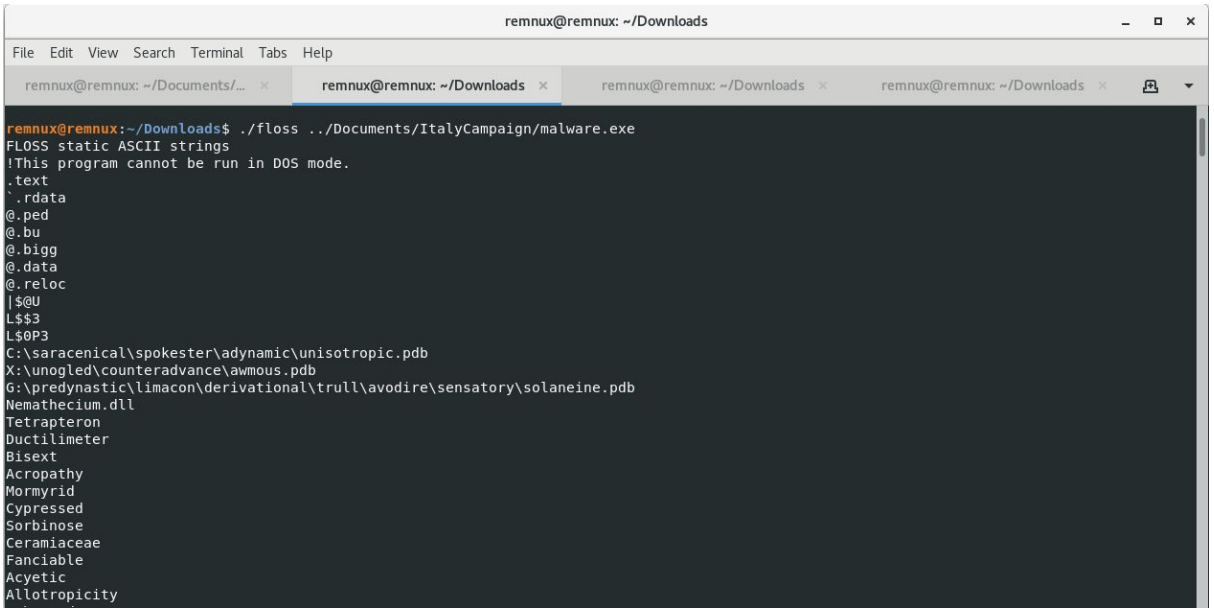

Figure 9. FLOSS tool

The complete list is left behind.

FLOSS static ASCII strings
!This program cannot be run in DOS mode.
.text
`.rdata
@.ped
@.bu
@.bigg
@.data

Report Jorge Amoros

@.reloc
|$@U
L$$3
L$0P3
C:\saracenical\spokester\adynamic\unisotropic.pdb
X:\unogled\counteradvance\awmous.pdb
G:\predynastic\limacon\derivational\trull\avodire\sensatory\solaneine.pdb
Nemathecium.dll
Tetrapteron
Ductilimeter
Bisext
Acropathy
Mormyrid
Cypressed
Sorbinose
Ceramiaceae
Fanciable
Acyetic
Allotropicity
Ambuscader
Dyotheletian
Uncorrigibleness
Lather
Byzantinism
Bryanthus
Unoverhauled
Peculium
Turritella
Medrick
Satinize
Ophiologic
Iddio
Enterotoxication
Adonize
Arrowlike
Truckle
Scabellum
Reviviscible
Preballoting
Birle
Pennill
Shielded
Electricalize
Tundagslatta
Gingili
Redistinguish
Overinventoried
Dagassa

Report Jorge Amoros

Lipography
Pandion
Unprince
Bondar
Attraction
Protopresbytery
Stovewood
Campshedding
DllUnregisterServer
Trogue
Undersaturation
Unmovingly
Deseret
Degradedness
Metapolitics
Tastily
Glaucionetta
Happify
Rombowline
Unchristened
Vacillator
Expressionism
Uveal
Fustin
Outbeg
Foreshape
Teleologism
Tenderling
Limnanthes
Nubilate
Petaloid
Coinstantaneousness
Impersuasible
Outsentry
Ephebic
Ostyak
Urosepsis
Osteolite
Unembezzled
Trimercuric
Unringed
Jeweling
Throughganging
Dracontites
Prompter
Flysch
Disobligingness
Sturnine

Report Jorge Amoros

Sugamo
Outsheathe
Sherryvallies
Cystoadenoma
Bewitchful
Nimbification
Aerobically
Thema
Nontransparency
DllCanUnloadNow
Hematohidrosis
Overslavish
Manyberry
Pseudoimpartial
Fireshine
Nonaerating
Paragnathus
Homostylous
Finnesko
Portugalism
Folkmoter
Sterhydraulic
Chalcis
Beghard
Ironbark
Onoclea
Hydroponics
DllGetClassObject
Gentilesse
Noetic
Mikadoism
Circumparallelogram
Purdah
Aptenodytes
DllRegisterServer
Unifiedness
Denominationally
Uptilt
Recarbonization
Myoneuralgia
Hypnophobic
Idler
SQ:\complexionless\unobedient\intoxication\anglist.pdb
E:\foreshow\unplanished\ridgebone\hemihedrally\glycolic\racegoing\acromiohumeral.pdb
j}j^
<PWj
I?`g
N:\pasquil\leucocytopenia\polycladine\serpolet\nonheading\albarello\lissom.pdb

GetCurrentProcessId
GetCurrentThreadId
VirtualProtect
GetTickCount
QueryPerformanceCounter
kernel32.dll
DriverProc
mciqtz32.dll
SnmpUtilOidFree
SnmpUtilOidAppend
SnmpUtilOidCpy
SnmpUtilOidCmp
snmpapi.dll
CreateWindowExW
SetWindowPos
user32.dll

com1
 microsoft1-0+
$Microsoft Root Certificate Authority0
070403125309Z
210403130309Z0w1
Washington1
Redmond1
Microsoft Corporation1!0
Microsoft Time-Stamp PCA0
    microsoft1-0+
$Microsoft Root Certificate Authority
.e0P
l0G0E
?http://crl.microsoft.com/pki/crl/products/microsoftrootcert.crl0T
H0F0D
8http://www.microsoft.com/pki/certs/MicrosoftRootCert.crt0

7. Examining resource files

No resource files to examine with resource hacker.

8. Yara Rules

We download the yara rules repository from :
https://github.com/Yara-Rules/rules/

and execute all of them like

yara ../../Downloads/rules-master/index.yar malware.exe > yarRules.tx

The rules detected are the followings :

- EH_Save [Tactic_DefensiveEvasion,Technique_AntiDebugging,SubTechnique_SEH] malware.exe
- SEH_Init [Tactic_DefensiveEvasion,Technique_AntiDebugging,SubTechnique_SEH] malware.exe
- MD5_Constants [] malware.exe
- IsPE32 [PECheck] malware.exe
- IsDLL [PECheck] malware.exe
- IsWindowsGUI [PECheck] malware.exe
- HasOverlay [PECheck] malware.exe
- HasDebugData [PECheck] malware.exe
- Microsoft_Visual_Cpp_v50v60_MFC [PEiD] malware.exe
- Borland_Delphi_30_additional [PEiD] malware.exe
- Borland_Delphi_30_ [PEiD] malware.exe
- Borland_Delphi_v40_v50 [PEiD] malware.exe
- Borland_Delphi_v30 [PEiD] malware.exe
- Borland_Delphi_DLL [PEiD] malware.exe

From these rules we can see that is a PE32, DLL file, that will execute some windows interface … We don't know what Borland delphi is. Quick google research tell is some kind of compiler but we are not sure why this rule is here.

9. Ghidra and IDA

We tried to analyse the malware using IDA and Ghridra but it so packed that we dont even know where to start looking into.

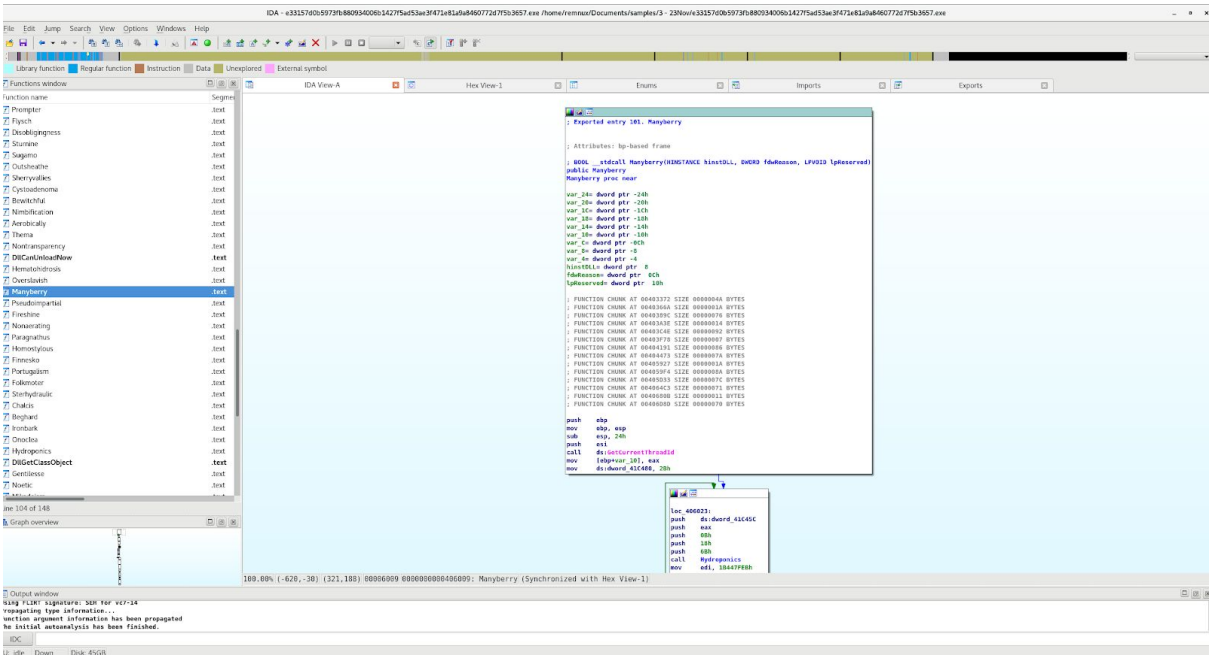

Figure 10. IDA view

Conclusion for static analysis :

We have successfully analyzed the sample and we can derive the following conclusions from this:

- This file is **malicious** (VirusTotal analysis)
- We successfully get the compilation details of the file and some clear text information from the data section.
- We checked the obfuscation details of the file by DIE. It’s highly packaged so it will be hard to gain much information without doing some dynamic analysis.
- It will pop up some windows according to Yara rules and imports.
- Imports also tell us that it will run some process but we can’t know what is it going to be since we don’t have network indicators.

Dynamic analysis

The malware is not possible to replicate. First of all, when we execute the excel file with the macros I think he is not able to download any malicious file since I am not in Italy. Also the dll itelsef cannot be runned since it's packed and we don't know the entry point to provide. We run it like

RUNDLL32.EXE .\c0nnection.dll,entrypoint(function)

Report Jorge Amoros

We first tried to unpack the malware to look for the entry point to execute the malware but with no luck.

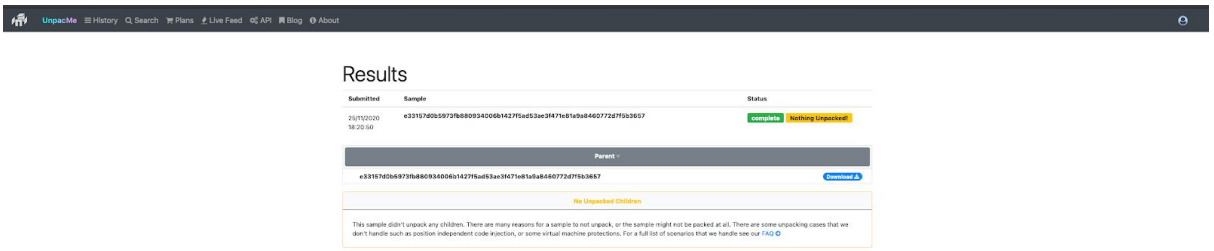

Figure 10. Unpack me website

We also tried to compare the register before and after executing the excel file but we are not sure that the register been changed are changed because of the malware.

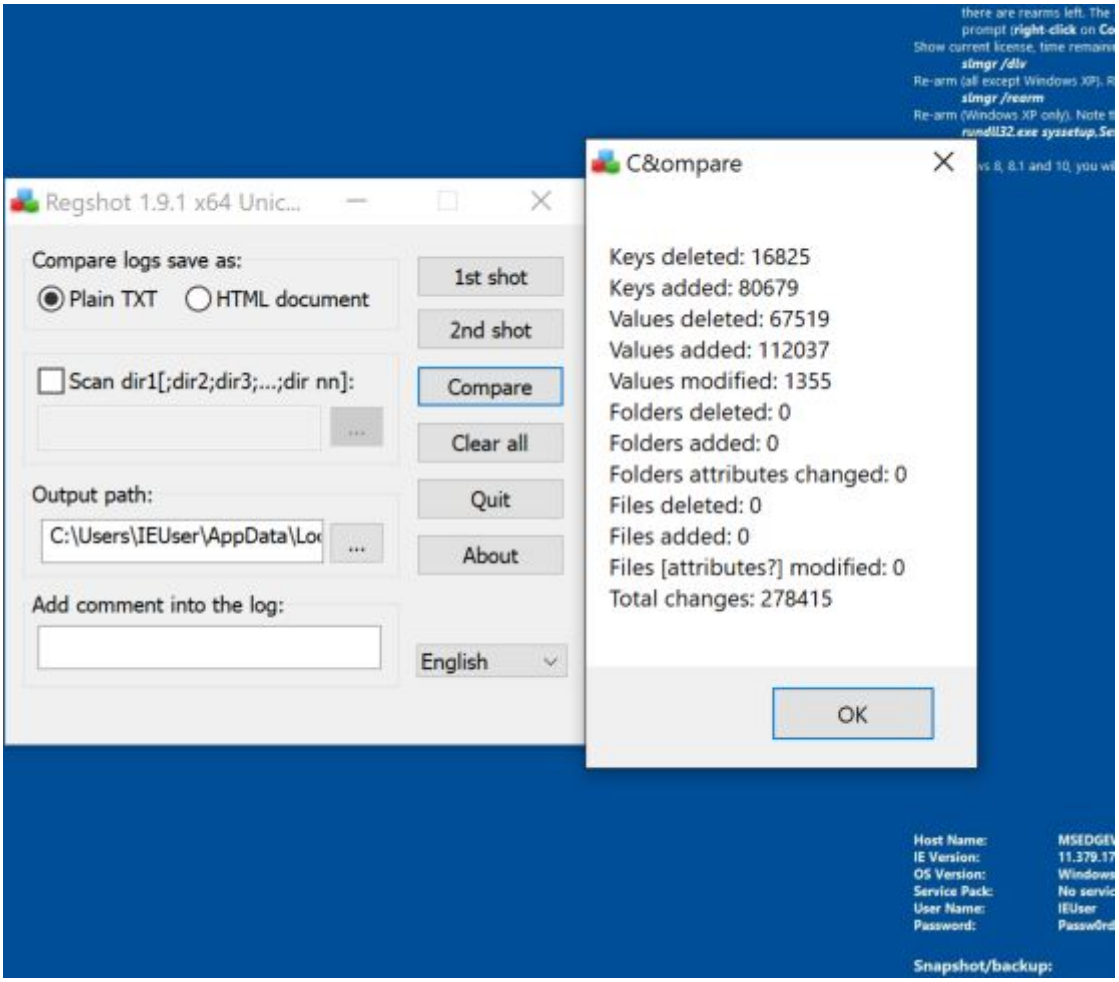

Figure 11. Regshot

Actually and after the presentation these values in the register may be affected by the malicious excel but we are not totally sure about it.

Indicators of compromised

Here is the list of the most interesting indicators that tell us that this file is compromised :

- Virus total detects it. 53/70
- Some strings are suspicious of being malicious like getCurrentProcessiD.
- Some libraries were tagged into the blacklist of libraries like snmpapi and well as most of the functions.
- The TimeStamp of the compiler is suspicious. Year 0.
- File checkSum is invalid.

The full list in the next image provided by PEstudio.

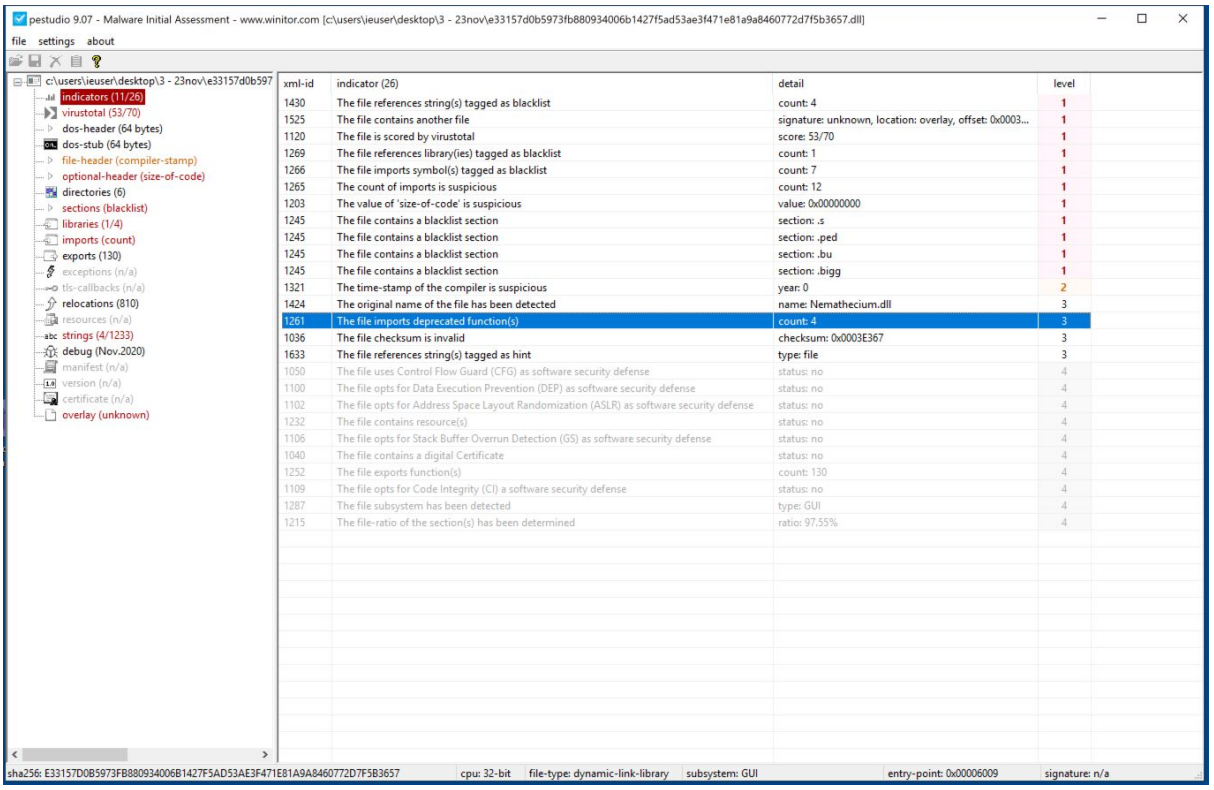

Figure 11. PEStudio