

## CHEAT SHEET - MALICIOUS FILE ANALYSIS

This cheat sheet summarises the processes and tools to perform static and dynamic analysis on malicious files.

Warning - make sure to always handle malicious files in a safe isolated virtual environment:

- [Remnux VM](#) - for static malware analysis
- [Flare VM](#) - for dynamic malware analysis

The complete description of the tools with detailed step by step processes and examples is available [here](#)

Additional information [github.com/sylvainhirsch/malware](https://github.com/sylvainhirsch/malware)

Author: Sylvain Hirsch ([sylvain@sylvainhirsch.ch](mailto:sylvain@sylvainhirsch.ch))

Date: June 2021 | Version: 1.0

### Static Malware Analysis - Generic Steps

General steps determine if the file is known as malicious, identify malicious strings and confirm the file format.

**Hash Value** - a digital fingerprint of a file.

- **certutil.exe -hashfile file SHA256** - Windows
- **sha256sum file** - Linux

**Hash Lookup** - on opensource malware databases (e.g. [VirusTotal](#), [Hybrid Analysis](#), etc.) to identify if the file is known as malicious and obtain addition information.

**Strings Extraction** - permits to extract strings from the file to identify interesting IOCs (e.g. url, command lines, encoded strings, file paths, registry keys, etc.).

- **strings file**
  - n to specify the minimum string length (default=3)
  - | **grep 'pattern'** to find specific pattern (e.g. 'http')
  - | **egrep 'regex'** to find specific regex (e.g. url regex)

**File Signature/Magic Number** - specific number embedded in the file, indicating the file format. Check the magic number of the file to verify its format.

- **hexdump.exe file | head** - Windows
- **xxd file | head** - Linux

### Microsoft Office OLE2 file - D0CF 11E0

Office OLE2 files can contain macros and embedded objects. Search and extract them to analyse their content.

[oleid](#) - parses OLE files to identify specific elements (e.g. VBA Macro, Flash Object, etc.).

- **oleid file**

[oletimes](#) - parses OLE files to identify the creation and modification timestamps of streams and objects.

- **oletimes file**

[oledump](#) - analyses all the streams in an OLE file. 'M' indicates a macro and 'O' an embedded object.

- **oledump file** - provides a summary of the streams
- **oledump -M file** - displays the metadata
- **oledump -s id -v file** - extracts the content of a stream
- **oledump -s id -v file > file.vba** - extracts it

[olevba](#) - parses OLE and OpenXML files to detect VBA macros, extracts their code in clear text and identifies specific malicious patterns and IOCs (e.g. URLs, IPs, etc.).

- **olevba file** - provides a summary of the streams
- **olevba [...] file** - extracts vba macro
  - deobf - deobfuscates VBA expressions
  - decode - decodes encoded elements
  - reveal - replaces deobfuscated code with original code

[mraptor](#) - identifies malicious VBA macros. Exit code: 0: No Macro, 2: Macro OK, 20: SUSPICIOUS.

- **mraptor file** - scans the file to detect malicious VBA macros
- **mraptor "directory\_name/\*"** - scans all files in a directory

[ViperMonkey](#) - VBA emulator that analyses and deobfuscates malicious macros.

- **vmonkey -a file** - extracts a summary of the streams
- **-a** - for alternative emulate engine

[LazyOffice](#) - uses WinAppDbg to extract URL, VBA and JavaScript from office documents. Use LazyOffice in a safe environment.

- **python loffice.py file\_type exit\_mode file -p path\_office**
  - Exit mode: url, proc, thread and none

[msoffcrypto-crack](#) - recovers the password of encrypted Office files.

- **python msoffcrypto-crack.py file**

[msoffcrypto-tool](#) - decrypts encrypted Office files.

- **msoffcrypto-tool file --test -v** - tests if file is encrypted
- **msoffcrypto-tool file.enc file.dec -p pwd** - decrypts file

### Microsoft Office Open XML file - 504B 0304

OOXML files are collections of XML files stored in archive format. They can contain macros and embedded objects.

[unzip](#) - unzips the docx file.

- **unzip file**

[egrep](#) - greps specific pattern.

- **egrep -r -i -n 'regex' \*** - find my regex [here](#) (to identify urls, command lines, their encoded versions and more)

[xlmdeobfuscator](#) - decodes obfuscated XLM macros.

- **xlmdeobfuscator --file file**

cf. Microsoft Office OLE2 file tools (e.g. olevba, etc.)

### PDF - 2550 4446

PDF files can contain embedded automatic actions, JavaScripts and Shellcodes. Search and extract them to analyse their content.

Interesting elements:

- /OpenAction or /AA - runs a script automatically
- /JavaScript or /JS - contains JavaScript code
- /ObjStm - indicates an embedded object within a stream
- /Launch - counts launched actions

[pdfid](#) - scans pdf to identify specific keywords. Be careful if there is an OpenAction (AA) and a JavaScript (JS).

- **pdfid file**
  - n - to output only present keywords

[pdf-parser](#) - parses pdf to identify fundamental elements.

- **pdf-parser file**
  - a - displays statistics
  - O - parses stream of /ObjStm objects
- **pad-parser --search openaction file**
  - search - searches for specific element
- **pad-parser --object id --filter --raw file**
  - object id - selects an object id
  - filter (-f) - decodes stream objects
  - raw (-w) - outputs the raw data
- **pad-parser --object id -f -w --dump=object\_id file**
  - dump - dumps the stream object

[peepdf](#) - parses and extracts the general information.

- **peepdf -i file** - provides a summary of the pdf properties
  - i - interactive console
- **--object id** - views the content of the object id
- **--object id > object\_id\_output** - extracts an object
- **--js\_analysis** - analyses JavaScript (cf. JS section)

## RTF - 7B5C 7274

RTF files contain linked and embedded objects stored in a hexadecimal format in "objdata" sub-destination.

[rtfobj](#) - automatically extracts embedded objects.

- **rtfobj file** - extracts embedded objects

[rtfdump](#) - enables to display all components of the rtf document and to deobfuscate them.

- **rtfdump file** - prints all the items in the rtf file

- **rtfdump -f O file** - prints all the object items  
-f - filters the items output

- **rtfdump -s id -H file** - extracts information about items

- **rtfdump -s id -H -c 'offset:length' file** - outputs the content at the selected offset

-s - selects a stream

-H - decodes hexacode

-i - provides information for a selected item

-S - deobfuscates hex characters

- **rtfdump -s id -H -c 'offset:length' -d file>file.bin**  
-d - dumps content

## Jar File - 504B 0304

[unzip](#) - unzips the jar file.

- **unzip file**

[Java Decompiler](#) - decompiles and displays Java source codes of .class files.

- **jd-gui**

[AdWindDecryptor](#) - a simple decryptor for encrypted files in the AdWind/jRAT/jBifrost Java RAT.

- **java -jar AdWindDecryptor.jar -a mega.download -r sky.drive -i drop.box -o decrypted-file**

-a - AES key file path

-r - serialized RSA KeyRep file

-i - input file path

-o - decrypted output file path

## JavaScript

[js\\_analyse](#) - [peepdf](#) - analyses JavaScript, decodes it and extracts relevant IOCs.

- **peepdf -i**

- **js\_analyse file file.js**

[SpiderMonkey](#) - Mozilla's JavaScript engine to debug malicious JavaScript.

- **js file**

- **strings write.bin.logs**

## ShellCode

A shellcode is a small piece of hexadecimal code used as a payload to perform various actions.

[shellcode2exe](#) - converts shellcode to executable file.

- **shellcode2exe 32/64 shellcode output.exe**

[scdbg.exe](#) - a shellcode emulator that outputs the Windows API calls performed by the shellcode.

- **wine scdbg.exe -f binfile**

[xorsearch](#) - identifies strings, embedded objects and shellcodes.

- **xorsearch -W file** - look for shellcode patterns

## Useful Services

[VirusTotal](#) - malicious file database and sandbox

[Hybrid Analysis](#) - online sandbox

[Any.run](#) - online sandbox

[URLhaus](#) - malicious file database

[UrlScan.io](#) - to visualise a website

[CyberChef](#) - to decode strings

[Regex101](#) - to create regular expressions

## Dynamic Malware Analysis

The analysis can be performed with [Flare VM](#).

[Process Monitor](#) - monitors and records changes in file system, registry, processes and other activities performed on the machine. Filters can be used to identify events of interest (e.g. WriteFile, RegSetValue, etc.).

- Launch Process Monitor before spawning a malicious file and observe the changes and actions generated on the system.

[Process Explorer](#) - monitors and displays all processes running on a system, including parent-child relationships, the loaded DLLs, the processes spawned, suspended and kills, and other activities.

- Launch Process Explorer before spawning a malicious files and identify all the processes spawned, suspended and killed, DLLs loaded, command lines run, etc.

[FakeNet-NG](#) - simulates legitimate network services, intercepts and redirects all the network traffic.

- FakeNet-NG can be used to identify the network IOCs in a safe and isolated environment.

[Regshot](#) - compares two Registry snapshots.

- Regshot can be run before and after spawning the malicious file to identify the changes (e.g. added, modified or deleted keys or values).

[Autoruns](#) - enumerates all programs that automatically start on a Windows system.

- Autoruns can be used to identify all persistence mechanisms.

## Sandbox

Sandboxes can be used for the automatic dynamic file analysis.

It is not recommended to upload sensitive files on any external sandboxes.

- [Cuckoo](#), [VirusTotal](#), [Hybrid Analysis](#), [Any.run](#)