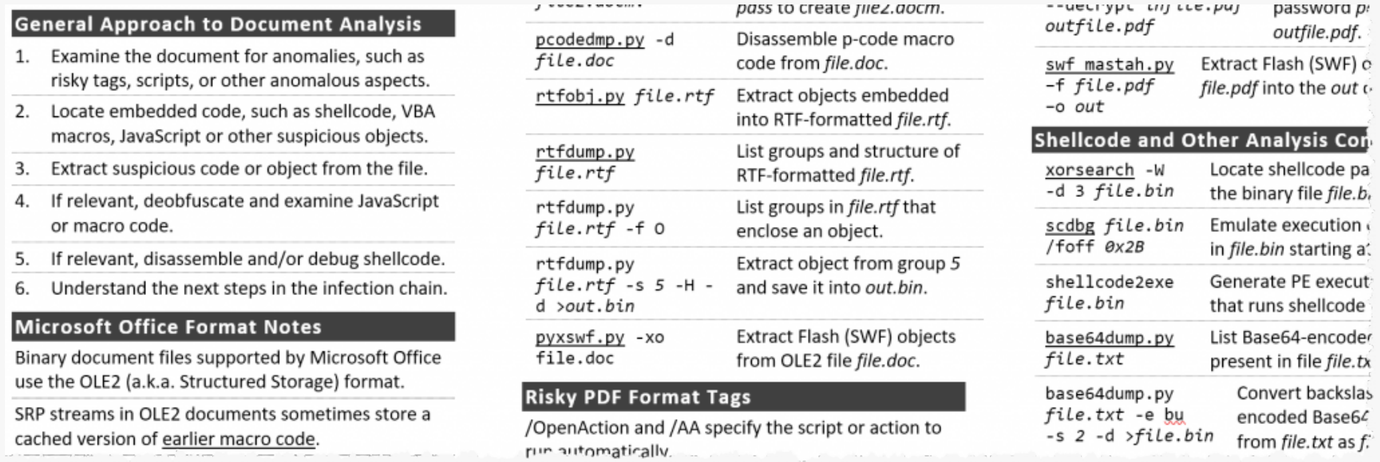
Analyzing Malicious Documents Cheat Sheet



General Approach to Document Analysis

1. Examine the document for anomalies, such as risky tags, scripts, or other anomalous aspects.
2. Locate embedded code, such as shellcode, VBA macros, JavaScript or other suspicious objects.
3. Extract suspicious code or object from the file.
4. If relevant, deobfuscate and examine JavaScript or macro code.
5. If relevant, disassemble and/or debug shellcode.
6. Understand the next steps in the infection chain.

Microsoft Office Format Notes

Binary document files supported by Microsoft Office use the OLE2 (a.k.a. Structured Storage) format.

SRP streams in OLE2 documents sometimes store a cached version of earlier macro code.

[illegible]	*pass* to create *file2.docm*.
`pcodedmp.py -d file.doc`	Disassemble p-code macro code from *file.doc*.
`rtfobj.py file.rtf`	Extract objects embedded into RTF-formatted *file.rtf*.
`rtfdump.py file.rtf`	List groups and structure of RTF-formatted *file.rtf*.
`rtfdump.py file.rtf -f O`	List groups in *file.rtf* that enclose an object.
`rtfdump.py file.rtf -s 5 -H -d >out.bin`	Extract object from group *5* and save it into *out.bin*.
`pyxswf.py -xo file.doc`	Extract Flash (SWF) objects from OLE2 file *file.doc*.

Risky PDF Format Tags

/OpenAction and /AA specify the script or action to run automatically.

`outfile.pdf`	password p *outfile.pdf*.
`swf_mastah.py -f file.pdf -o out`	Extract Flash (SWF) o *file.pdf* into the *out*

Shellcode and Other Analysis Co

`xorsearch -W -d 3 file.bin`	Locate shellcode pa the binary file *file.b*
`scdbg file.bin /foff 0x2B`	Emulate execution in *file.bin* starting a
`shellcode2exe file.bin`	Generate PE execut that runs shellcode
`base64dump.py file.txt`	List Base64-encode present in file *file.tx*
`base64dump.py file.txt -e bu -s 2 -d >file.bin`	Convert backslas encoded Base64 from *file.txt* as *f*

This cheat sheet outlines tips and tools for analyzing malicious documents, such as Microsoft Office, RTF and Adobe Acrobat (PDF) files. To print it, use the one-page PDF version; you can also edit the Word version to customize it for you own needs.

General Approach to Document Analysis

MORE ON

Information Security

Malicious Software

The SANS malware analysis course I've co-authored explains the techniques summarized in this cheat sheet and covers many other reverse-engineering topics.

If you like this reference, take a look at my other IT and security cheat

1. Examine the document for anomalies, such as risky tags, scripts, or other anomalous aspects.
2. Locate embedded code, such as shellcode, VBA macros, JavaScript or other suspicious objects.
3. Extract suspicious code or object from the file.
4. If relevant, deobfuscate and examine JavaScript or macro code.
5. If relevant, disassemble and/or debug shellcode.
6. Understand the next steps in the infection chain.

sheets.

SHARE

Microsoft Office Format Notes

- Binary document files supported by Microsoft Office use the OLE2 (a.k.a. Structured Storage) format.
- SRP streams in OLE2 documents sometimes store a cached version of earlier macro code.
- OOXML documents (.docx, .xlsm, etc.) supported by MS Office use zip compression to store contents.
- Macros embedded in OOXML files are stored inside the OLE2 binary file, which is within the zip archive.
- RTF documents don't support macros, but can contain other files embedded as OLE1 objects.

Useful MS Office File Analysis Commands

unzip *file.pptx*	Extract contents of OOXML file *file.pptx*.
olevba.py *file.xlsm* olevba.py *file.doc*	Locate and extract macros from *file.xlsm* or *file.doc*.

oledump.py *file.xls*	List all OLE2 streams present in *file.xls*.
oledump.py -s *3* -v *file.xls*	Extract macros stored inside stream *3* in *file.xls*.
oledump.py *file.xls* -p plugin_http_heuristics	Find obfuscated URLs in *file.xls* macros.
msoffice-crypt -d -p *pass file.docm file2.docm*	Decrypt OOXML file *file.docm* using password *pass* to create *file2.docm*.
pcodedmp.py -d *file.doc*	Disassemble p-code macro code from *file.doc*.
rtfobj.py *file.rtf*	Extract objects embedded into RTF-formatted *file.rtf*.
rtfdump.py *file.rtf*	List groups and structure of RTF-formatted *file.rtf*.
rtfdump.py *file.rtf* -f O	List groups in *file.rtf* that enclose an object.
rtfdump.py *file.rtf* -s *5* -H -d > *out.bin*	Extract object from group *5* and save it into *out.bin*.
pyxswf.py -xo file.doc	Extract Flash (SWF) objects from OLE2 file *file.doc*.

Risky PDF Format Tags

- /OpenAction and /AA specify the script or action to run automatically.
- /JavaScript and /JS specify JavaScript to run.
- /GoTo changes the view to a specified destination within the PDF or in another PDF file.

- /Launch can launch a program or open a document.
- /URI accesses a resource by its URL.
- /SubmitForm and /GoToR can send data to URL.
- /RichMedia can be used to embed Flash in a PDF.
- /ObjStm can hide objects inside an Object Stream.
- Be mindful of obfuscation with hex codes, such as /JavaScript vs. /J#61vaScript. (See examples.)

Useful PDF File Analysis Commands

pdfid.py *file.pdf*	Scan *file.pdf* for risky keywords and dictionary entries.
peepdf.py -fl *file.pdf*	Examine *file.pdf* for risky tags and malformed objects.
pdf-parser.py --object *id file.pdf*	Display contents of object *id* in *file.pdf*. Add "--filter --raw" to decode the object's stream.
qpdf --password=*pass* --decrypt *infile.pdf outfile.pdf*	Decrypt *infile.pdf* using password *pass* to create *outfile.pdf*.
swf_mastah.py -f *file.pdf* -o *out*	Extract Flash (SWF) objects from *file.pdf* into the *out* directory.

Shellcode and Other Analysis Commands

xorsearch -W -d 3 *file.bin*	Locate shellcode patterns inside the binary file *file.bin*.

scdbg *file.bin* /foff *0x2B*	Emulate execution of shellcode in *file.bin* starting at offset *0x2B*.
shellcode2exe *file.bin*	Generate PE executable *file*.exe that runs shellcode from *file.bin*.
jmp2it *file.bin 0x2B*	Execute shellcode in file *file.bin* starting at offset *0x2B*.
base64dump.py *file.txt*	List Base64-encoded strings present in file *file.txt*.
base64dump.py *file.txt* -e bu -s *2* -d > *file.bin*	Convert backslash Unicode-encoded Base64 string *#2* from *file.txt* as *file.bin* file.

Additional Document Analysis Tools

- SpiderMonkey, V8 and box-js help deobfuscate JavaScript that you extract from document files.
- PDF Stream Dumper combines several PDF analysis utilities under a single graphical user interface.
- ViperMonkey emulates VBA macro execution.
- VirusTotal and some automated analysis sandboxes can analyze aspects of malicious document files.
- Hachoir-urwid can display OLE2 stream contents.
- 101 Editor (commercial) and FileInsight hex editors can parse and edit OLE structures.
- ExeFilter can filter scripts from Office and PDF files.

- REMnux distro includes many of the free document analysis tools mentioned above.

Post-Scriptum

Special thanks for feedback to Pedro Bueno and Didier Stevens. If you have suggestions for improving this cheat sheet, please let me know. Creative Commons v3 “Attribution” License for this cheat sheet version 3.0.

Updated September 6, 2017

DID YOU LIKE THIS?

Follow me for more of the good stuff.

TWITTER RSS FEED NEWSLETTER

About the Author

Lenny Zeltser develops teams, products, and programs that use information security to achieve business results. Over the past two decades, Lenny has been leading efforts to establish resilient security practices and solve hard security problems. As a respected author and speaker, he has been advancing cybersecurity tradecraft and contributing to the community. His insights build upon 20 years of real-world experiences, a Computer Science degree from the University of Pennsylvania, and an MBA degree from MIT Sloan.

Learn more