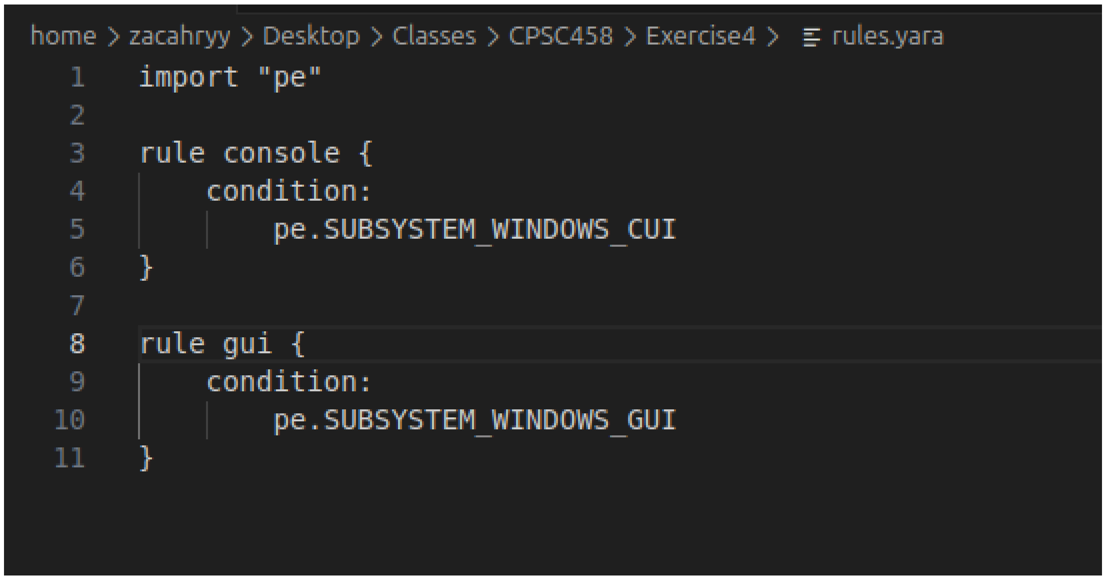
Name: Zachary Faulkner
Course: Fall 2024 CPSC 458-03
Exercise: 4

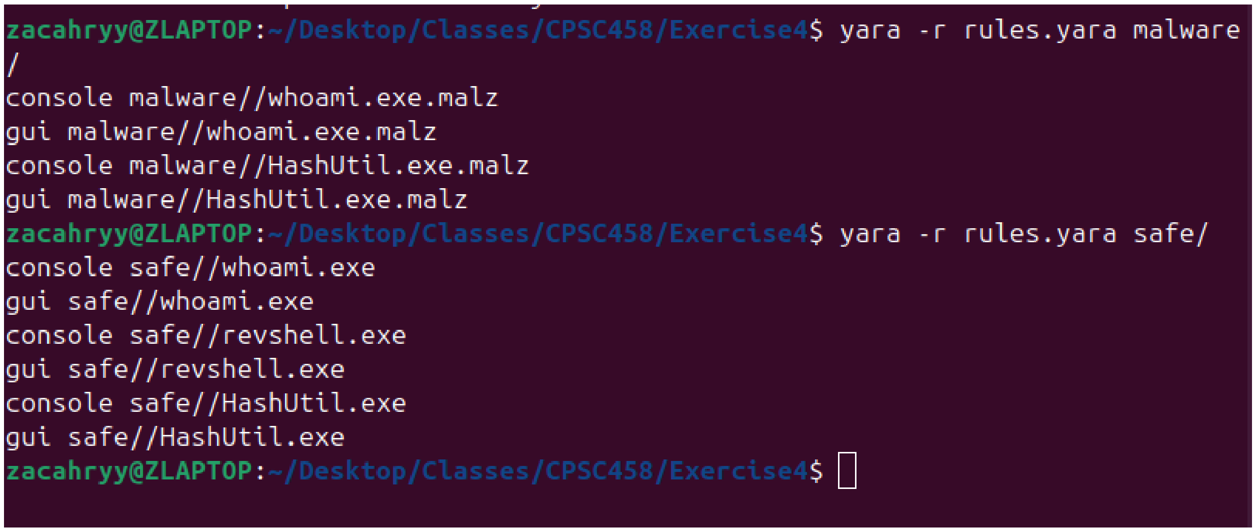
Testing YARA

Before writing the rules to identify the malware family, I wrote a test Yara file to make sure that Yara was functioning properly. This code should identify malware that targets the console or windows subsystem.



```
home > zacahryy > Desktop > Classes > CPSC458 > Exercise4 > rules.yara
import "pe"

rule console {
    condition:
        pe.SUBSYSTEM_WINDOWS_CUI
}

rule gui {
    condition:
        pe.SUBSYSTEM_WINDOWS_GUI
}
```

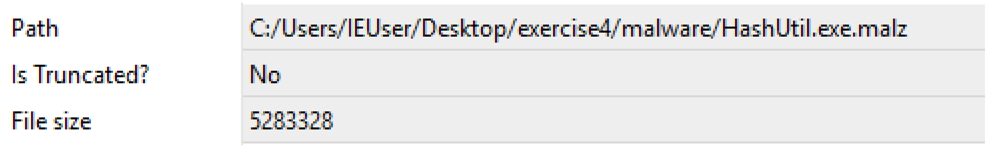
The result we should see is either “console” or “gui” appended with the name of the executable. In this case, all files in the “malware” or “safe” files should be tagged. As you can see from the result below, Yara is working as expected.



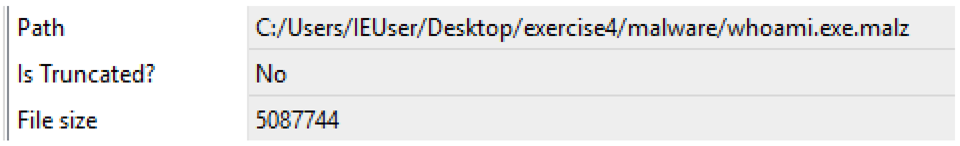
```
zacahryy@ZLAPTOP:~/Desktop/Classes/CPSC458/Exercise4$ yara -r rules.yara malware
/
console malware//whoami.exe.malz
gui malware//whoami.exe.malz
console malware//HashUtil.exe.malz
gui malware//HashUtil.exe.malz
zacahryy@ZLAPTOP:~/Desktop/Classes/CPSC458/Exercise4$ yara -r rules.yara safe/
console safe//whoami.exe
gui safe//whoami.exe
console safe//revshell.exe
gui safe//revshell.exe
console safe//HashUtil.exe
gui safe//HashUtil.exe
zacahryy@ZLAPTOP:~/Desktop/Classes/CPSC458/Exercise4$
```

Identifying The Malware Family Using Yara

In order to identify the malware family correctly without false positives, I had to find characteristics specific to only the malware files that are not present in the safe files. Using some simple static analysis and prior knowledge of these files I was able to accomplish this. The tool I used to find the aforementioned malware characteristics was PEBear. My first step was analyzing the file sizes of both the malware samples and safe samples. Because the malware samples have hidden imports and/or code, they would likely have a much higher file size than the safe samples. I saw that the HashUtil.exe.malz file and the whoami.exe.malz malware samples both had file sizes over 5,000,000. The only safe sample executable that had a file size this large was HashUtil.exe. By adding this rule to Yara, I have already filtered out revshell.exe and whoami.exe. Below are the file sizes of HashUtil.exe.malz and whoami.exe.malz.



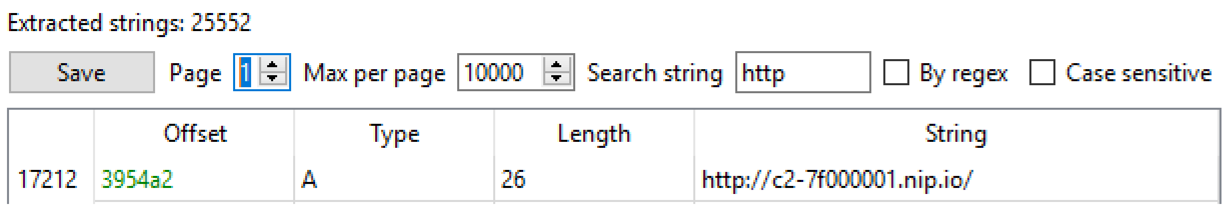
Path	C:/Users/IEUser/Desktop/exercise4/malware/HashUtil.exe.malz
Is Truncated?	No
File size	5283328



Path	C:/Users/IEUser/Desktop/exercise4/malware/whoami.exe.malz
Is Truncated?	No
File size	5087744

My next step was analyzing the strings of each file to find any distinct strings or imports for the malware samples. Because these files had already been analyzed in previous projects, I already had prior knowledge of specific strings that I knew would be distinct. For whoami.exe.malz, I knew the malware made a http request to a command and control server, I was able to find this request in the strings output and add it to the strings list in my yara rules file. While looking for other distinct whoami.exe.malz strings, I also stumbled across the string “(5.c-”. This string is not present in the safe whoami.exe file, so I added it to the strings list to detect whoami.exe.malz. I then moved on to comparing HashUtil.exe.malz and HashUtil.exe. I remembered from previously analyzing HashUtil.exe.malz for the project that it created a mutex. I searched the strings of HashUtil.exe.malz to double check and was able to find it. I also remembered from class that it created a persist.exe file. This file was present in the strings so I added it to the strings list as well. Lastly, I decided to search the strings from HashUtil.exe.malz and HashUtil.exe that I did not know of already. I used filters such as “.exe”, “http”, and “Create”. These filters didn’t provide much that I didn’t already know, however when I started searching for .dll imports like “kernel32.dll”, I noticed that the “@” symbol was being used before these import calls. I then filtered the strings for both HashUtil.exe.malz and HashUtil.exe using the “@” symbol. There were multiple differences between the two files, for simplicity I just decided to use the distinct string “@Out of bounds:” as my last indicator for HashUtil.exe.malz. I then took the malware characteristics that I found for whoami.exe.malz and HashUtil.exe.malz and compared them to “revshell.exe” to make sure there wasn’t any overlap. There wasn’t, so I moved on to creating the rules file. Below are screenshots of the characteristics I used.

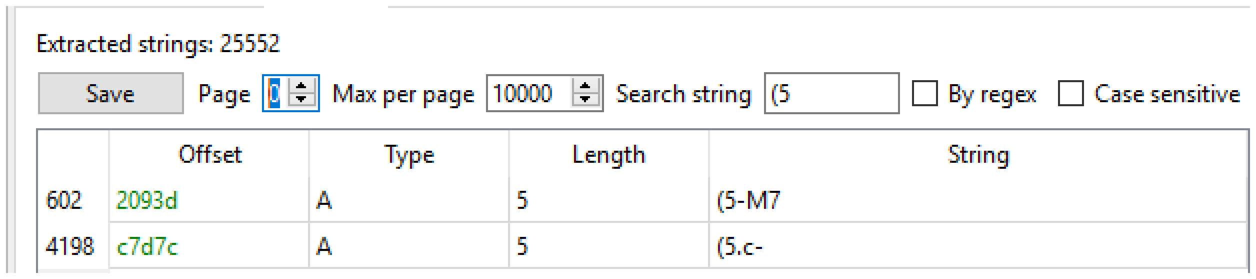
whoami.exe.malz:



Extracted strings: 25552

Page 1 Max per page 10000 Search string http ☐ By regex ☐ Case sensitive

	Offset	Type	Length	String
17212	3954a2	A	26	http://c2-7f000001.nip.io/

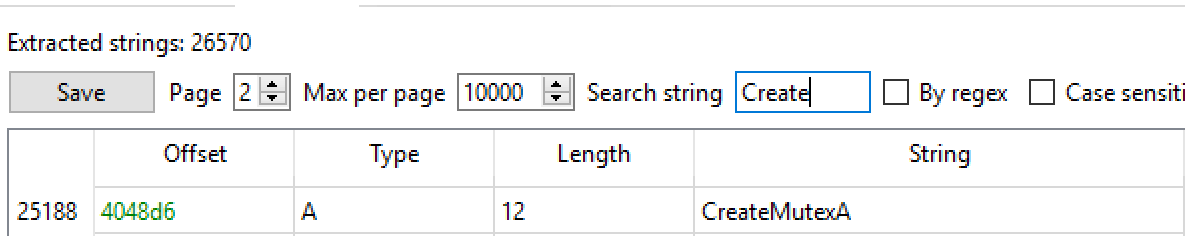


Extracted strings: 25552

Page 0 Max per page 10000 Search string (5 ☐ By regex ☐ Case sensitive

	Offset	Type	Length	String
602	2093d	A	5	(5-M7
4198	c7d7c	A	5	(5.c-

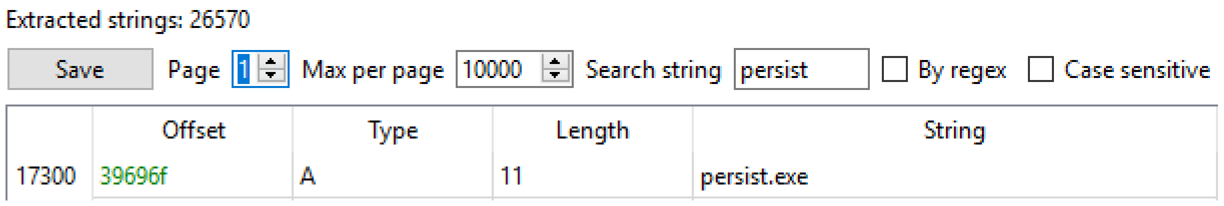
HashUtil.exe.malz:



Extracted strings: 26570

Page 2 Max per page 10000 Search string Create ☐ By regex ☐ Case sensiti

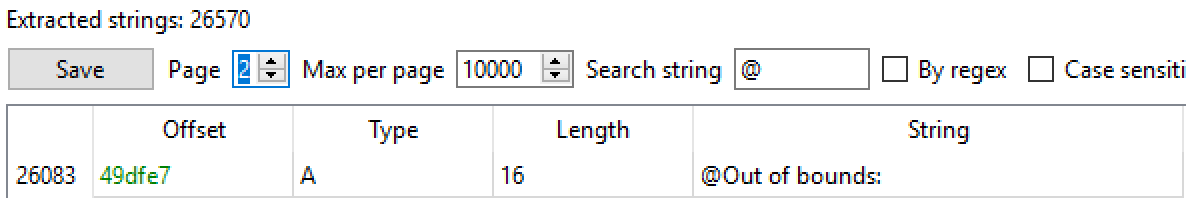
	Offset	Type	Length	String
25188	4048d6	A	12	CreateMutexA



Extracted strings: 26570

Page 1 Max per page 10000 Search string persist ☐ By regex ☐ Case sensitive

	Offset	Type	Length	String
17300	39696f	A	11	persist.exe

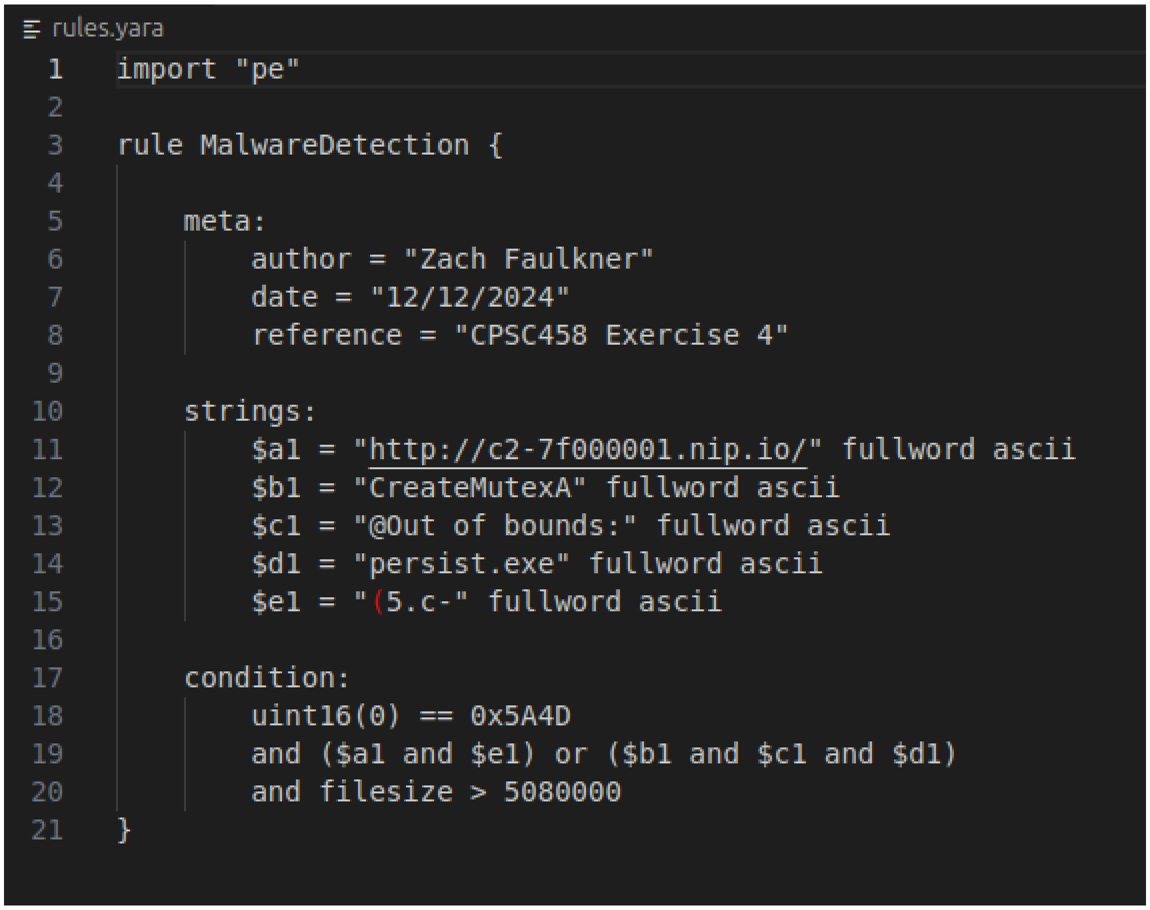


Extracted strings: 26570

Page 2 Max per page 10000 Search string @ ☐ By regex ☐ Case sensiti

	Offset	Type	Length	String
26083	49dfe7	A	16	@Out of bounds:

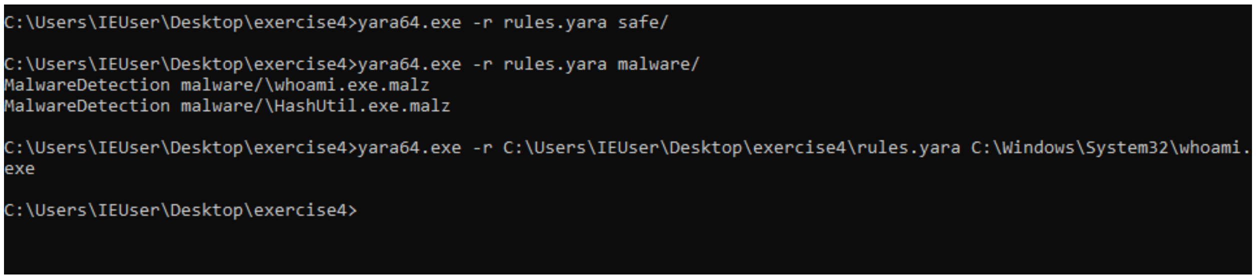
Using the provided article detailing how to create a Yara rules file and the malware characteristics I gathered during analysis, I created the following rules file:



rules.yara

```
import "pe"

rule MalwareDetection {

    meta:
        author = "Zach Faulkner"
        date = "12/12/2024"
        reference = "CPSC458 Exercise 4"

    strings:
        $a1 = "http://c2-7f000001.nip.io/" fullword ascii
        $b1 = "CreateMutexA" fullword ascii
        $c1 = "@Out of bounds:" fullword ascii
        $d1 = "persist.exe" fullword ascii
        $e1 = "(5.c-" fullword ascii

    condition:
        uint16(0) == 0x5A4D
        and ($a1 and $e1) or ($b1 and $c1 and $d1)
        and filesize > 5080000
}
```

Strings “$a1” and “$e1” correspond to identifying whoami.exe.malz. The strings “$b1”, “$c1”, and “$d1” correspond to identifying HashUtil.exe.malz. In the conditions section I then created rules to identify the malware family. The condition “uint16(0) == 0x5A4D” is used to make sure the file being analyzed is a Windows executable. The condition “and ($a1 and $e1) or ($b1 and $c1 and $d1)” is used to see if the file being analyzed matches the distinct malware characteristics I found previously. Lastly, the command “and filesize > 5080000” checks to make sure the file size is above the threshold mentioned previously. After creating the file, I then tested it within my Windows VM to make sure there were no false positives. As you can see, the rules file correctly identified the malware samples without any false positives, completing the objective of the assignment.



```
C:\Users\IEUser\Desktop\exercise4>yara64.exe -r rules.yara safe/

C:\Users\IEUser\Desktop\exercise4>yara64.exe -r rules.yara malware/
MalwareDetection malware/\whoami.exe.malz
MalwareDetection malware/\HashUtil.exe.malz

C:\Users\IEUser\Desktop\exercise4>yara64.exe -r C:\Users\IEUser\Desktop\exercise4\rules.yara C:\Windows\System32\whoami.
exe

C:\Users\IEUser\Desktop\exercise4>
```