5-2. Rule Writing

In this lab we are going to be looking at how we can write YARA Rules.

> https://github.com/HuskyHacks/PMAT-labs/tree/main/labs/5-2.RuleWriting

main ▾ **PMAT-labs** / labs / **5-2.RuleWriting** /

husky release v1.0	
..	
.gitkeep	release v1.0
Malware.yara1.exe.malz.7z	release v1.0
password.txt	release v1.0
sha256sum.txt	release v1.0

We are already provided with a template, but there are few simple sections that we need to kee in mind.

1. There is always a Rule Name from where the YARA RULE begins. For example

```
rule PE32_Files_Checker {


}
```

2. Within the rule section we have 3 different sections.
    1. Meta ⟶ This contains the metadata.
    2. Strings ⟶ These contains the unique and uncommon strings to be looked for.
    3. Conditions ⟶ The logic behind detection is here.

```
rule PE32_Files_Checker {

        meta:
                Author:
                Description:
                Date:

        strings:
                $a = "Hellow World!"
                $b = "Hello World!"


        condition:
                $a or $b
}
```

More can be learnt by reading through the YARA documentation.

> https://yara.readthedocs.io/en/stable/gettingstarted.html#

Rule For Malicious Sample

We will initially start with writing the YARA rule name and metadata information. Which are as follows.

```
rule Grab_Malware_Sample1 {

    meta:
        last_updated = "2022-3-23"
        author = "Kamran Saifullah - Frog Man (@deFr0ggy)"
        description = "Yara Rule for Malware.yara1.exe.malz Sample"
```

For PE (Portable Executables) or in other words the executables which run on Windows Always begin with MZ (4D 5A) ⟶ Magic Bytes. If we try to look at the hex data of the executable we can find it clearly.

```
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 MZ................
00000013 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 .....@.............
00000026 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...................
00000039 00 00 00 80 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C ...............!..L
0000004C CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E .!This program cann
0000005F 6F 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F ot be run in DOS mo
00000072 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 50 45 00 00 4C de....$.......PE..L
00000085 01 10 00 B4 83 69 61 00 8E 05 00 66 08 00 00 E0 00 07 01 .....ia....f.......
00000098 0B 01 02 22 00 8E 01 00 00 BE 01 00 00 A8 00 00 C0 14 00 ..."...............
000000AB 00 00 10 00 00 00 A0 01 00 00 00 40 00 00 10 00 00 00 02 ...........@.......
```

The MZ is at offset 0×1 and 0×2 respectively. Following that we can observe another line which states.

```
!This program cannot be run in DOS mode
```

This is another common string found in PEs. Which for our sample is starting from the offset 0×4D.

So, we have 2 strings here.

```
strings:
    $a = { 4D 5A }
    $b = "!This program can not be run in dos mode."
```

Finally, we can start building the conditions i.e.

> If (4D 5A) is found at offset 0, trigger the rule

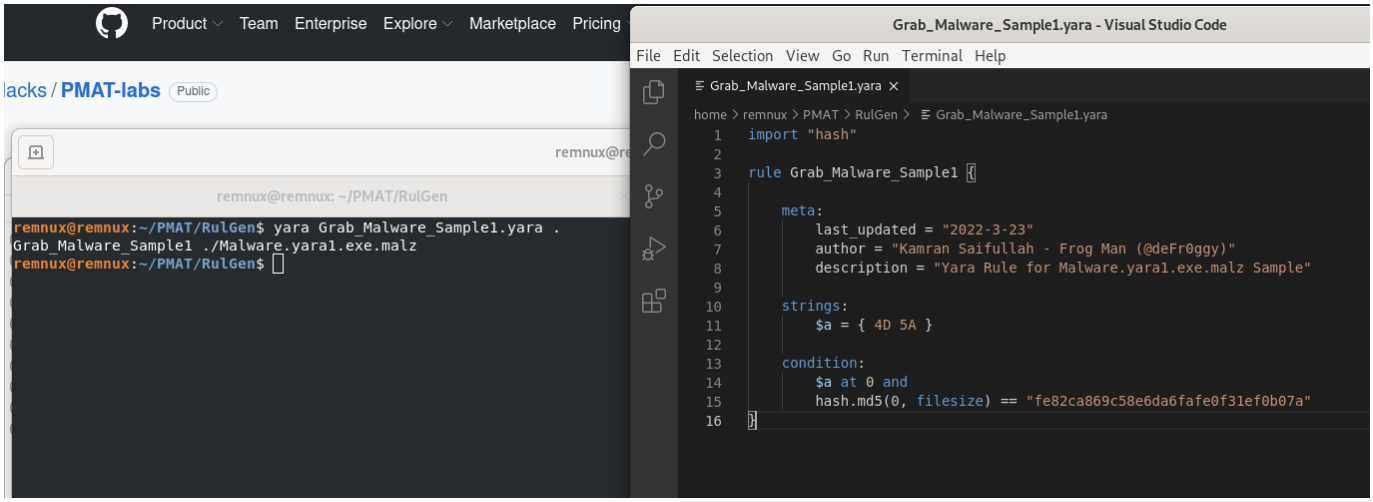
```
condition:

    $a at 0
```

On running our rule we can see that it is indeed working.

```
remnux@remnux:~/PMAT/RulGen$ yara Grab_Malware_Sample1.yara .
Grab_Malware_Sample1 ./Malware.yara1.exe.malz
remnux@remnux:~/PMAT/RulGen$
```

Moving further, we can also use YARA modules. Let's import "hash" module and match the hash of the file.



In similar ways, we can write YARA Rules and also we can use YarGen to automatically generate Yara Rules for us. But remember that needs to be fine tuned as well.

Follow Me

Twitter: https://twitter.com/deFr0ggy

GitHub: https://github.com/deFr0ggy

LinkedIn: https://linkedin.com/in/kamransaifullah