Open in app ↗

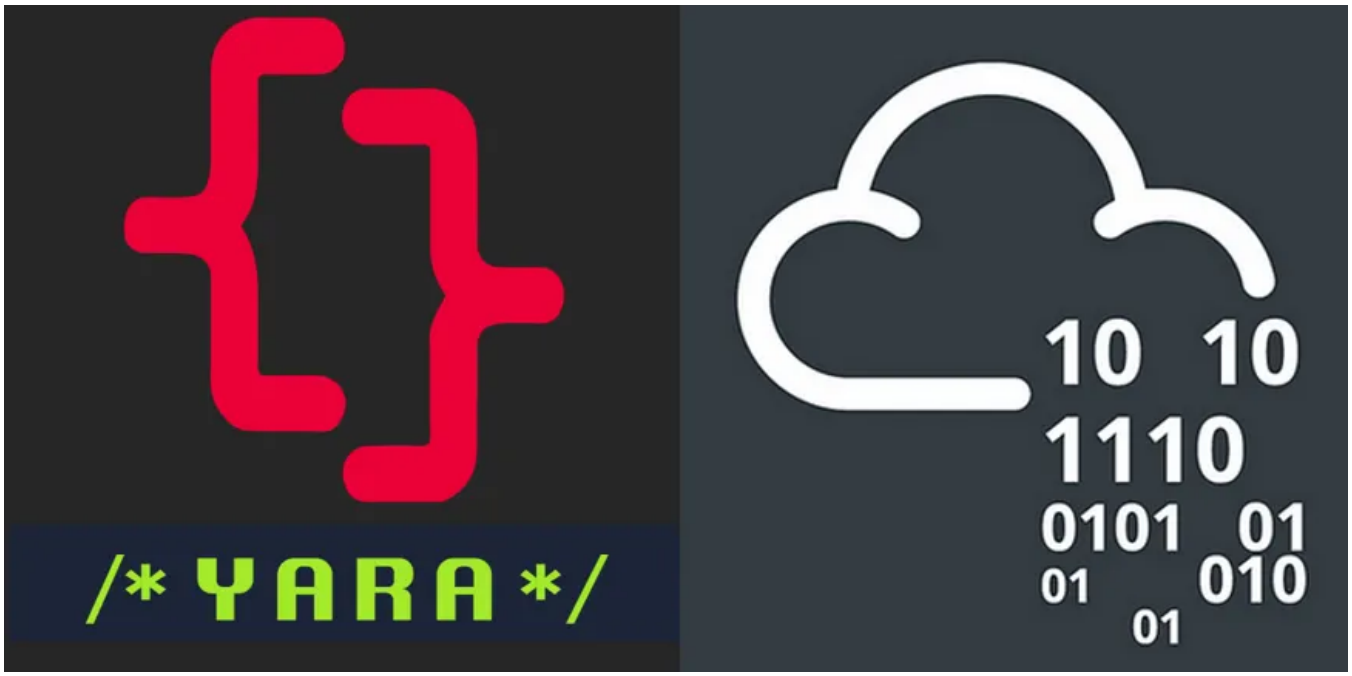
Get unlimited access to the best of Medium for less than $1/week. Become a member

Yara | SOC Level 1 | Cyber Threat Intelligence | TryHackMe Walkthrough

27 min read · Feb 21, 2025

IritT

Learn the applications and language that is Yara for everything threat intelligence, forensics, and threat hunting!



Room URL: https://tryhackme.com/r/room/threatinteltools

Task 1 Introduction Introduction

This room will expect you to understand basic Linux familiarity, such as installing software and commands for general navigation of the system. Moreso, this room isn't designed to test your knowledge or for point-scoring. It is here to encourage you to follow along and experiment with what you have learned here.

As always, I hope you take a few things away from this room, namely, the wonder that Yara (*Yet Another Ridiculous Acronym*) is and its importance in infosec today. Yara was developed by Victor M. Alvarez (**@plusvic**) and **@VirusTotal**. Check the GitHub repo **here**.

Answer the questions below

Let's get started

Task 2 What is Yara?

All about Yara

"The pattern matching swiss knife for malware researchers (and everyone else)" (Virustotal., 2020)

With such a fitting quote, Yara can identify information based on both binary and textual patterns, such as hexadecimal and strings contained within a file.

Rules are used to label these patterns. For example, Yara rules are frequently written to determine if a file is malicious or not, based upon the features — or patterns — it presents. Strings are a fundamental component of programming languages. Applications use strings to store data such as text.

For example, the code snippet below prints "Hello World" in Python. The text "Hello World" would be stored as a string.

```
print("Hello World!")
```

We could write a Yara rule to search for “hello world” in every program on our operating system if we would like.

Why does Malware use Strings?

Malware, just like our “Hello World” application, uses strings to store textual data. Here are a few examples of the data that various malware types store within strings:

Type	Data	Description
Ransomware	12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw	Bitcoin Wallet for ransom payments
Botnet	12.34.56.7	The IP address of the Command and Control (C&C) server

Caveat: Malware Analysis

Explaining the functionality of malware is vastly out of scope for this room due to the sheer size of the topic. I have covered strings in much more detail in “Task 12 — Strings” of my MAL: Introductory room. In fact, I am creating a whole Learning Path for it. If you’d like to get a taster whilst learning the fundamentals, I’d recommend my room.

Answer the questions below

2.1 What is the name of the base-16 numbering system that Yara can detect? (This is also known as “hex”. An example of this is: ff745c9b137d86e)

Hexadecimal is a numeral system that uses **16** symbols, which are:

0 to 9 (representing values 0 to 9)

A to F (representing values 10 to 15)

In Yara, hexadecimal is frequently used to represent binary data in a more human-readable form. When malware researchers are analyzing files, they often encounter raw binary data that needs to be interpreted. Hexadecimal helps convert large

binary numbers into a more compact and understandable format.

If you have a hexadecimal value like **ff745c9b137d86e**, it represents a sequence of binary data. In this case:

ff = 11111111 in binary

74 = 01110100 in binary

5c = 01011100 in binary

And so on.

Yara uses rules that can detect these patterns (often in the form of **hexadecimal**) to identify whether a file is malicious.

Example:
Imagine a piece of malware that communicates with a Command and Control server. The malware might store an IP address in hexadecimal format, such as **12.34.56.7**, which could be represented as **0C 22 38 47** in hex. Yara rules can be written to detect this hexadecimal pattern to identify potential malicious activity based on known indicators.

Thus, the hexadecimal (or hex) numbering system plays a vital role in malware detection by representing binary data, which Yara can search for using patterns.

Answer: **Hexadecimal**

2.2 Would the text “Enter your Name” be a string in an application? (Yay/Nay)

The text **“Enter your Name”** would indeed be considered a **string** in an application. In programming, a string is simply a sequence of characters, typically used to represent textual data.

String Definition: A string is any sequence of characters, which can include letters, numbers, punctuation, and spaces. The text **“Enter your Name”** falls under this definition because it is a sequence of characters, specifically intended for display or input by the user.

Example:

If you are developing a user interface (UI) in an application, you might prompt users to enter their name. The text **"Enter your Name"** would be displayed in a text box or a label, and it would be handled by the application as a string. It could be stored in a variable or used directly to present instructions to the user.

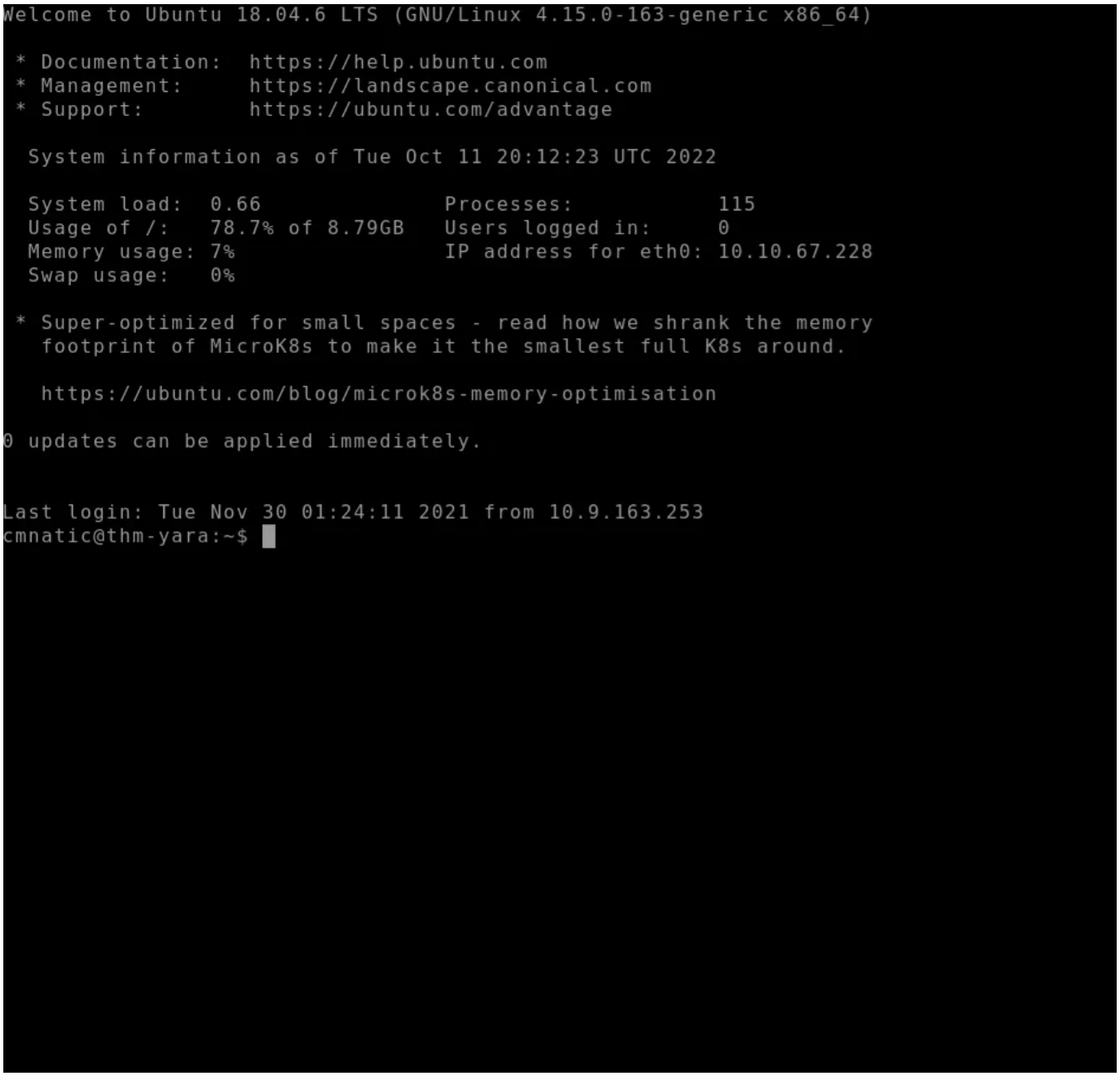
Answer: **Yay**

Task 3 Deploy

This room deploys an Instance with the tools being showcased already installed for you. Press the "Start Machine" button and wait for an IP address to be displayed and connect in one of two ways:

In-Browser (No VPN required)

Deploy your own instance by pressing the green "Start Machine" button and scroll up to the top of the room and await the timer. The machine will start in a split-screen view. In case the VM is not visible, use the blue "Show Split View" button at the top-right of the page.



Using SSH (TryHackMe VPN required).

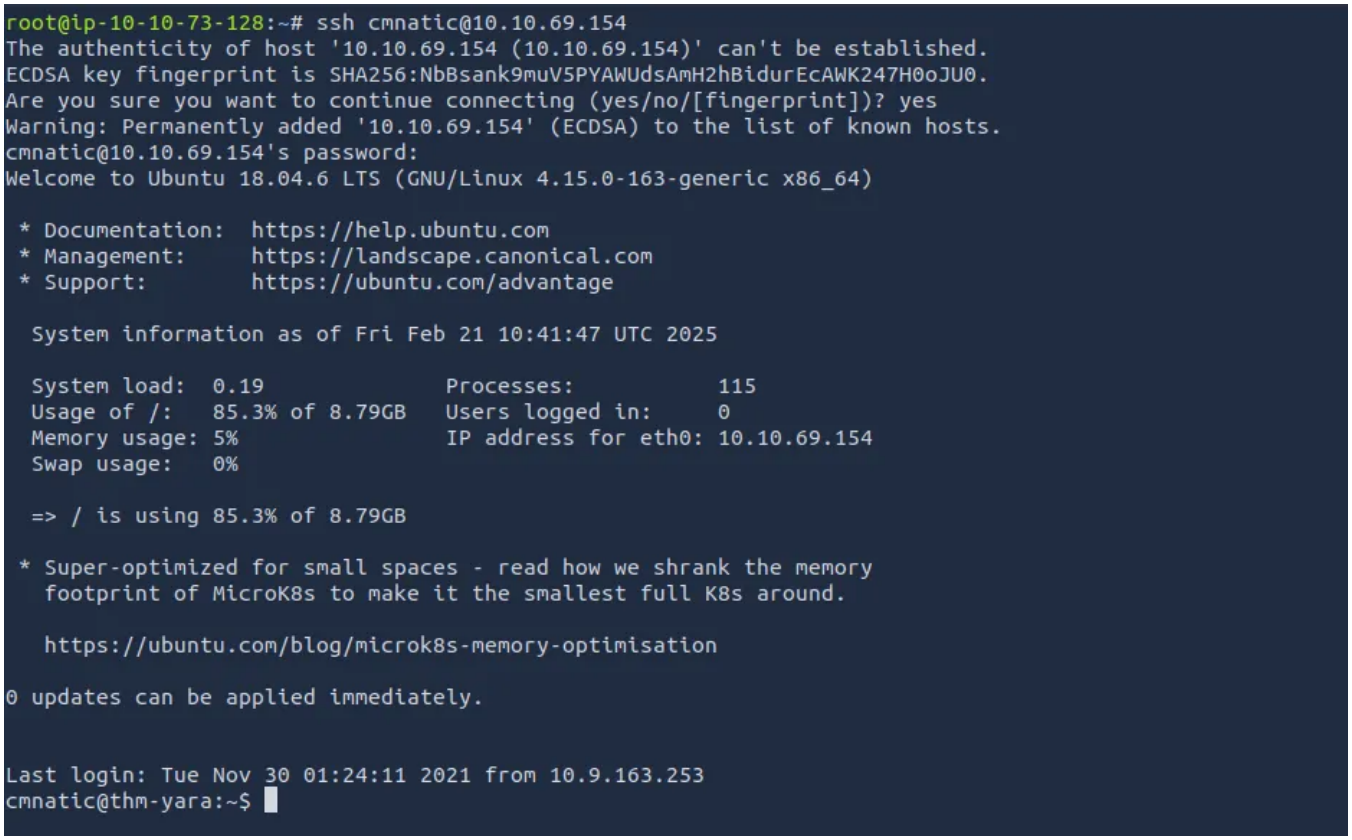
You must be connected to the TryHackMe VPN if you wish to connect your deployed Instance from your own device. If you are unfamiliar with this process, please visit the TryHackMe OpenVPN room to get started. If you have any issues, please read our support articles.

IP Address: **MACHINE_IP**

Username: **cmnatic**

Password: **yararules!**

SSH Port: **22**




```
ssh cmnatic@MACHINE_IP
```

Enter the password when asked

```
yararules!
```

Answer the questions below

I've connected to my instance!

Task 4 Introduction to Yara Rules

Your First Yara Rule

The proprietary language that Yara uses for rules is fairly trivial to pick up, but hard to master. This is because your rule is only as effective as your understanding of the patterns you want to search for.

Using a Yara rule is simple. Every `yara` command requires two arguments to be valid, these are:
1) The rule file we create
2) Name of file, directory, or process ID to use the rule for.

Every rule must have a name and condition.

For example, if we wanted to use "**myrule.yar**" on directory "**some directory**", we would use the following command:

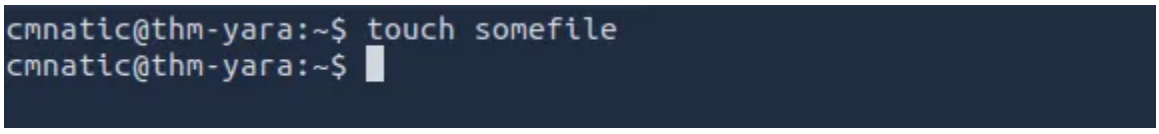
```
yara myrule.yar somedirectory
```

Note that **.yar** is the standard file extension for all Yara rules.

We'll make one of the most basic rules you can make below.

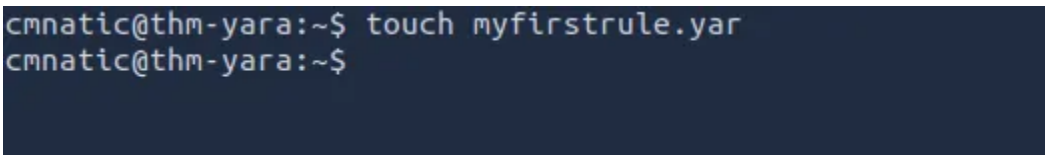
1. Make a file named "**somefile**" via `touch somefile`
2. Create a new file and name it "**myfirstrule.yar**" like below:

Creating a file named somefile



```
cmnatic@thm:~$ touch somefile
```

Creating a file named myfirstrule.yar

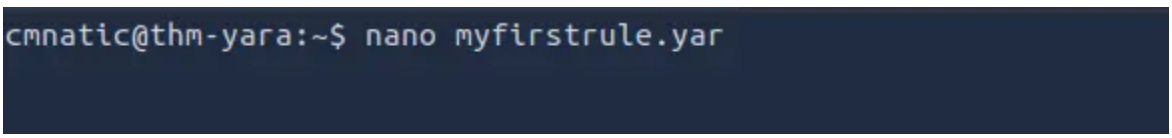



```
cmnatic@thm touch myfirstrule.yar
```

3. Open the "myfirstrule.yar" using a text editor such as `nano` and input the snippet below and save the file:

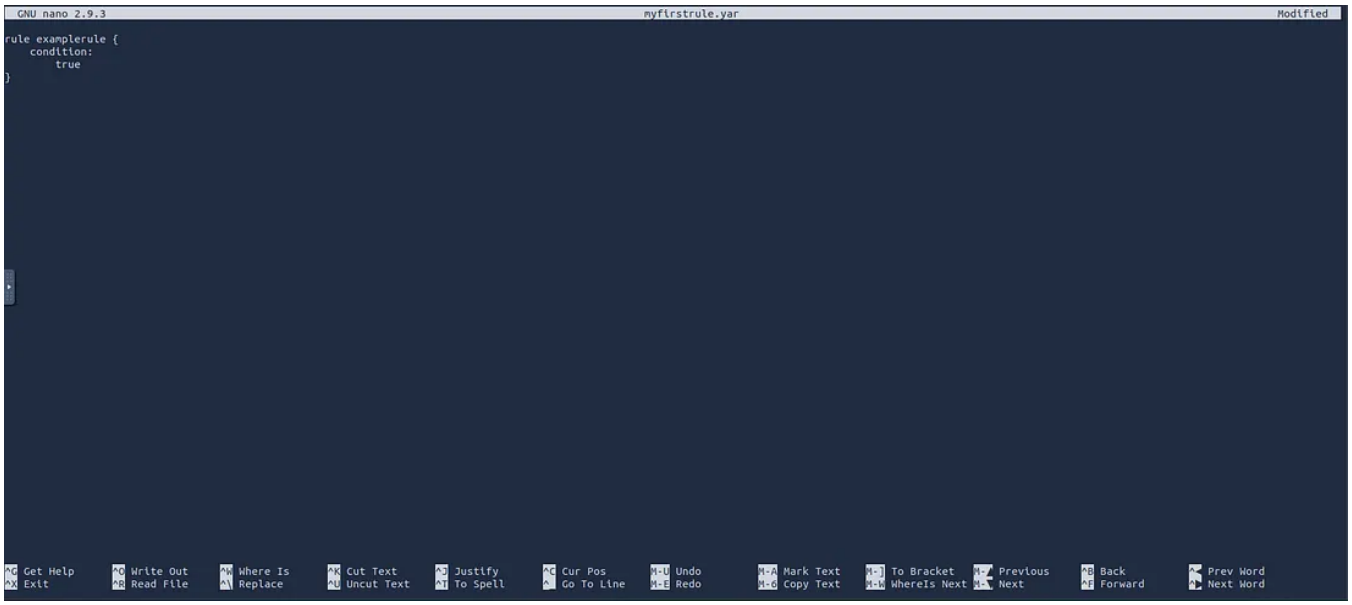
```
rule examplerule {
        condition: true
}
```

Open the **.yar** file using nano:




```
nano myfirstrule.yar
```

Write a valid YARA rule inside the file




```
rule examplerule {
    condition:
        true
}
```

Save the file: Press CTRL + X

Press Y to confirm saving

Press Enter to save and exit

Inputting our first snippet into “myfirstrule.yar” using nano

```
cmnatic@thm nano myfirstrule.yar   GNU nano 4.8 myfirstrule.yar Modified
rule examplerule {
        condition: true
}
```

The **name** of the rule in this snippet is `examplerule`, where we have one condition -

in this case, the **condition** is `condition`. As previously discussed, every rule requires both a name and a condition to be valid. This rule has satisfied those two requirements.

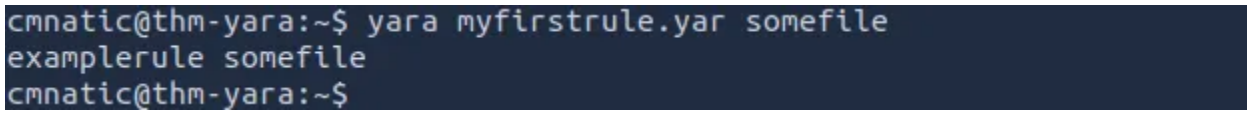
Simply, the rule we have made checks to see if the **file/directory/PID** that we specify exists via `condition: true`. If the file does exist, we are given the output of `examplerule`

Let's give this a try on the file **"somefile"** that we made in step one:

```
yara myfirstrule.yar somefile
```

If "**somefile**" exists, Yara will say `examplerule` because the pattern has been met - as we can see below:

Verifying our the examplerule is correct




```
cmnatic@thm:~$ yara myfirstrule.yar somefile
examplerule somefile
```

- YARA scanned **somefile** using your rule from **myfirstrule.yar.**
- Since **examplerule somefile** appeared in the output, this confirms that the rule conditions were met for **somefile**.
- The rule l has the condition **true**, which means it always matches any file.

If the file does not exist, Yara will output an error such as that below:

Yara complaining that the file does not exist

```
cmnatic@thm:~$ yara myfirstrule.yar sometextfile
error scanning sometextfile: could not open file
```

Congrats! You've made your first rule.

Answer the questions below

One rule to — well — rule them all.

Task 5 Expanding on Yara Rules

Yara Conditions Continued...

Checking whether or not a file exists isn't all that helpful. After all, we can figure that out for ourselves...Using much better tools for the job.

Yara has a few conditions, which I encourage you to read **here** at your own leisure.

The following keywords are reserved and cannot be used as an identifier:

all	and	any	ascii	at	base64	base64wide	condition
contains	endswith	entrypoint	false	filesize	for	fullword	global
import	icontains	iendswith	iequals	in	include	int16	int16be
int32	int32be	int8	int8be	istartswith	matches	meta	nocase
none	not	of	or	private	rule	startswith	strings
them	true	uint16	uint16be	uint32	uint32be	uint8	uint8be
wide	xor	defined					

However, I'll detail a few below and explain their purpose.

Keyword

Desc
Meta
Strings
Conditions
Weight

Meta

This section of a Yara rule is reserved for descriptive information by the author of the rule. For example, you can use `desc`, short for description, to summarise what your rule checks for. Anything within this section does not influence the rule itself. Similar to commenting code, it is useful to summarise your rule.

Strings

Remember our discussion about strings in Task 2? Well, here we go. You can use strings to search for specific text or hexadecimal in files or programs. For example, say we wanted to search a directory for all files containing “Hello World!”, we would create a rule such as below:

```
rule helloworld_checker{
        strings:
                $hello_world = "Hello World!"
}
```

We define the keyword `Strings` where the string that we want to search, i.e., "Hello World!" is stored within the variable `$hello_world`

Of course, we need a condition here to make the rule valid. In this example, to make this string the condition, we need to use the variable's name. In this case, `$hello_world`:

```
rule helloworld_checker{
        strings:
                $hello_world = "Hello World!"
```

```
        condition:
                $hello_world
}
```

Essentially, if any file has the string “Hello World!” then the rule will match. However, this is literally saying that it will only match if “Hello World!” is found and will not match if “*hello world*” or “*HELLO WORLD*.”

To solve this, the condition `any of them` allows multiple strings to be searched for, like below:

```
rule helloworld_checker{
        strings:
                $hello_world = "Hello World!"
                $hello_world_lowercase = "hello world"
                $hello_world_uppercase = "HELLO WORLD"

        condition:
                any of them
}
```

Now, any file with the strings of:

1. Hello World!

2. hello world

3. HELLO WORLD

Will now trigger the rule.

Conditions

We have already used the `true` and `any of them` condition. Much like regular programming, you can use operators such as:

- <= less than or equal to
- >= more than or equal to
- != not equal to

For example, the rule below would do the following:

```
rule helloworld_checker{
        strings:
                $hello_world = "Hello World!"

        condition:
        #hello_world <= 10
}
```

The rule will now:

1. Look for the “Hello World!” string
2. Only say the rule matches if there are less than or equal to ten occurrences of the “Hello World!” string

Combining keywords

Moreover, you can use keywords such as:

- **and**
- **not**
- **or**

To combine multiple conditions. Say if you wanted to check if a file has a string and is of a certain size (in this example, the sample file we are checking is **less than** <10 kb and has “Hello World!” you can use a rule like below:

```
rule helloworld_checker{
        strings:
                $hello_world = "Hello World!"

        condition:
                $hello_world and filesize < 10KB
}
```

The rule will only match if both conditions are true. To illustrate: below, the rule we created, in this case, did not match because although the file has “Hello World!”, it has a file size larger than 10KB:

Yara failing to match the file mytextfile because it is larger than 10kb

```
cmnatic@thm:~$ <output intentionally left blank>
```

However, the rule matched this time because the file has both “Hello World!” and a file size of less than 10KB.

Yara successfully matching the file mytextfile because it has “**Hello World**” and a file size of less than 10KB

```
cmnatic@thm:~$ yara myfirstrule.yar mytextfile.txt
helloworld_textfile_checker mytextfile.txt
```

Remembering that the text within the red box is the name of our rule, and the text within the green is the matched file.

Anatomy of a Yara Rule

Anatomy of a yara Rule

Yara is a tool used to identify file, based on textual or binary pattern.

A rule consists of a set of strings and conditions that determine its logic.

Rules can be compiled with "yarac" to increase the speed of multiple Yara scans.

1 Import Module

Yara modules allow you to extend its functionality. The PE module can be used to match specific data from a PE:

- pe.number_of_exports
- pe.sections[0].name
- pe.imphash()
- pe.imports("kernel32.dll")
- pe.is_dll()

List of modules: pe, elf, hash, math, cuckoo, dotnet, time

2 Rule Name

The rule name identifies your Yara rule. It is recommended to add a meaningful name. There are different types of rules:

- Global rules: applies for all your rules in the file.
- Private rules: can be called in a condition of a rule but not reported.
- Rule tags: used to filter yara's output.

3 Metadata

Rules can also have a metadata section where you can put additional information about your rule.

- Author
- Date
- Description
- Etc..

4 Strings

The field strings is used to define the strings that should match your rule. It exists 3 type of strings:

- Text strings
- Hexadecimal strings
- Regex

5 Condition

Conditions are Boolean expressions used to match the defined pattern.

- Boolean operators:
  - and, or, not
  - <=, >=, ==, <, >, !=
- Arithmetic operators:
  - +, -, *, \, %
- Bitwise operators:
  - &, |, <<, >>, ^, ~
- Counting strings:
  - #string0 == 5
- Strings offset:
  - $string1 at 100

```
import "pe"

rule demo_rule : Tag1 Demo
{
    meta:
        author = "Thomas Roccia"
        description = "demo"
        hash = ""

    strings:
        $string0 = "hello" nocase wide
        $string1 = "world" fullword ascii
        $hex1 = { 01 23 45 ?? 89 ab cd ef }
        $re1 = /md5: [0-9a-zA-Z]{32}/

    condition:
        uint16(0) == 0x5A4D and filesize < 2000KB
        or pe.number_of_sections == 1 and
        any of ($string*) and (not $hex1 or $re1)
}
```

Text Strings

Text strings can be used with modifiers:

- nocase: case insensitive
- wide: encoded strings with 2 bytes per character
- fullword: non alphanumeric
- xor(0x01-0xff): look for xor encryption
- base64: base64 encoding

Hexadecimal

Hex strings can be used to match piece of code:

- Wild-cards: { 00 ?2 A? }
- Jump: { 3B [2-4] B4 }
- Alternatives: { F4 (B4 | 56) }

Regex

Regular expression can also be used and defined as text strings but enclosed in forward slash.

Advanced Condition

- Accessing data at a given position: uint16(0) == 0x5A4D
- Check the size of the file: filesize < 2000KB
- Set of strings: any of ($string0, $hex1)
- Same condition to many strings: for all of them : (# > 3)
- Scan entry point: $value at pe.entry_point
- Match length: !rel[1] == 32
- Search within a range of offsets: $value in (0..100)

@FRØGGER_
Thomas Roccia

Information security researcher “fr0gger_” has recently created a **handy cheatsheet** that breaks down and visualises the elements of a YARA rule (shown above, all image credits go to him). It’s a great reference point for getting started!

Answer the questions below

Upwards and onwards...

Task 6 Yara Modules

Integrating With Other Libraries

Frameworks such as the **Cuckoo Sandbox** or **Python’s PE Module** allow you to improve the technicality of your Yara rules ten-fold.

Cuckoo

Cuckoo Sandbox is an automated malware analysis environment. This module allows you to generate Yara rules based upon the behaviours discovered from Cuckoo Sandbox. As this environment executes malware, you can create rules on specific behaviours such as runtime strings and the like.

Python PE

Python’s PE module allows you to create Yara rules from the various sections and elements of the Windows Portable Executable (PE) structure.

Explaining this structure is out of scope as it is covered in my **malware introductory room**. However, this structure is the standard formatting of all executables and DLL files on windows. Including the programming libraries that are used.

Examining a PE file’s contents is an essential technique in malware analysis; this is because behaviours such as cryptography or worming can be largely identified without reverse engineering or execution of the sample.

Answer the questions below

Sounds pretty cool!

Task 7 Other tools and Yara

Yara Tools

Knowing how to create custom Yara rules is useful, but luckily you don't have to create many rules from scratch to begin using Yara to search for evil. There are plenty of GitHub **resources** and open-source tools (along with commercial products) that can be utilized to leverage Yara in hunt operations and/or incident response engagements.

LOKI (What, not who, is Loki?)

LOKI is a free open-source IOC (*Indicator of Compromise*) scanner created/written by Florian Roth.

Based on the GitHub page, detection is based on 4 methods:

1. File Name IOC Check
2. Yara Rule Check **(we are here)**
3. Hash Check
4. C2 Back Connect Check

There are additional checks that LOKI can be used for. For a full rundown, please reference the **GitHub readme**.

LOKI can be used on both Windows and Linux systems and can be downloaded **here**.

Please note that you are not expected to use this tool in this room.

Displaying Loki's help menu

```
cmnatic@thm:~/Loki$ python3 loki.py -h
usage: loki.py [-h] [-p path] [-s kilobyte] [-l log-file] [-r remote-loghost]
               [-t remote-syslog-port] [-a alert-level] [-w warning-level]
               [-n notice-level] [--allhds] [--alldrives] [--printall]
               [--allreasons] [--noprocscan] [--nofilescan] [--vulnchecks]
               [--nolevcheck] [--scriptanalysis] [--rootkit] [--noindicator]
               [--dontwait] [--intense] [--csv] [--onlyrelevant] [--nolog]
```

```
                [--update] [--debug] [--maxworkingset MAXWORKINGSET]
                [--syslogtcp] [--logfolder log-folder] [--nopesieve]
                [--pesieveshellc] [--python PYTHON] [--nolisten]
                [--excludeprocess EXCLUDEPROCESS] [--force]
```

```
Loki - Simple IOC Scanner

optional arguments:
  -h, --help            show this help message and exit
```

THOR (superhero named programs for a superhero blue teamer)

THOR *Lite* is Florian's newest multi-platform IOC AND YARA scanner. There are precompiled versions for Windows, Linux, and macOS. A nice feature with THOR Lite is its scan throttling to limit exhausting CPU resources. For more information and/or to download the binary, start **here**. You need to subscribe to their mailing list to obtain a copy of the binary. **Note that THOR is geared towards corporate customers**. THOR Lite is the free version.

Please note that you are not expected to use this tool in this room.

Displaying Thor Lite's help menu

```
cmnatic@thm:~$ ./thor-lite-linux-64 -h
Thor Lite
APT Scanner
Version 10.7.3 (2022-07-27 07:33:47)
cc) Nextron Systems GmbH
Lite Version
```

```
> Scan Options
  -t, --template string      Process default scan parameters from
this YAML file
  -p, --path strings         Scan a specific file path. Define
multiple paths by specifying this option multiple times. Append
':NOWALK' to the path for non-recursive scanning (default: only the
system drive) (default [])
      --allhds               (Windows Only) Scan all local hard
drives (default: only the system drive)
```

```
      --max_file_size uint   Max. file size to check (larger files
are ignored). Increasing this limit will also increase memory usage
of THOR. (default 30MB)

> Scan Modes
      --quick     Activate a number of flags to speed up the scan at
cost of some detection.
                  This is equivalent to: --noeventlog --nofirewall
--noprofiles --nowebdirscan --nologscan --noevtx --nohotfixes --
nomft --lookback 3 --lookback-modules filescan
```

FENRIR (naming convention still mythical themed)

This is the 3rd **tool** created by Neo23x0 (Florian Roth). You guessed it; the previous 2 are named above. The updated version was created to address the issue from its predecessors, where requirements must be met for them to function. Fenrir is a bash script; it will run on any system capable of running bash (nowadays even Windows).

Please note that you are not expected to use this tool in this room.

Running Fenrir

```
cmnatic@thm-yara:~/tools$ ./fenrir.sh
#############################################################
    ____             _
   / __/__ ___  ____(_)___
  / _// -_) _ \/ __/ / __/
 /_/  \__/_//_/_/ /_/_/
 v0.9.0-log4shell

 Simple Bash IOC Checker
 Florian Roth, Dec 2021
#############################################################
```

YAYA (Yet Another Yara Automaton)

YAYA was created by the **EFF** (*Electronic Frontier Foundation*) and released in September 2020. Based on their website, "*YAYA is a new open-source tool to help researchers manage multiple YARA rule repositories. YAYA starts by importing a set of*

high-quality YARA rules and then lets researchers add their own rules, disable specific rulesets, and run scans of files."

Note: Currently, YAYA will only run on Linux systems.

Running YAYA

```
cmnatic@thm-yara:~/tools$ yaya
YAYA - Yet Another Yara Automaton
Usage:
yaya [-h]
    -h print this help screen
Commands:
   update - update rulesets
   edit - ban or remove rulesets
   add - add a custom ruleset, located at
   scan - perform a yara scan on the directory at
```

In the next section, we will examine **LOKI** further...

Answer the questions below

Cool tools. I'm ready to use one of them.

Task 8 Using LOKI and its Yara rule set

Using LOKI

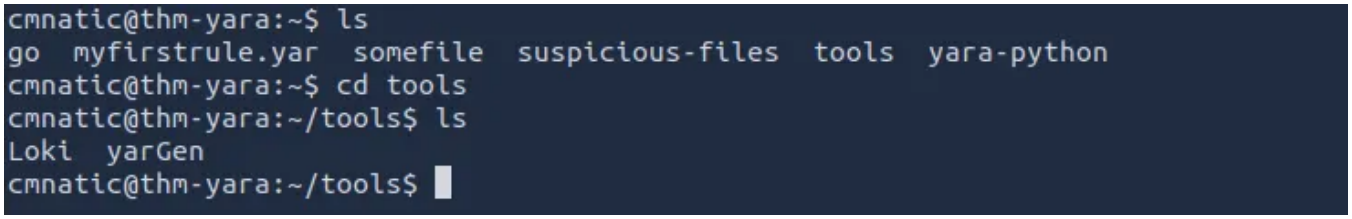
As a security analyst, you may need to research various threat intelligence reports, blog postings, etc. and gather information on the latest tactics and techniques used in the wild, past or present. Typically in these readings, IOCs (hashes, IP addresses, domain names, etc.) will be shared so rules can be created to detect these threats in your environment, along with Yara rules. On the flip side, you might find yourself in a situation where you've encountered something unknown, that your security stack of tools can't/didn't detect. Using tools such as Loki, you will need to add your own rules based on your threat intelligence gathers or findings from an incident response engagement (forensics).

As mentioned before, Loki already has a set of Yara rules that we can benefit from

and start scanning for evil on the endpoint straightaway.

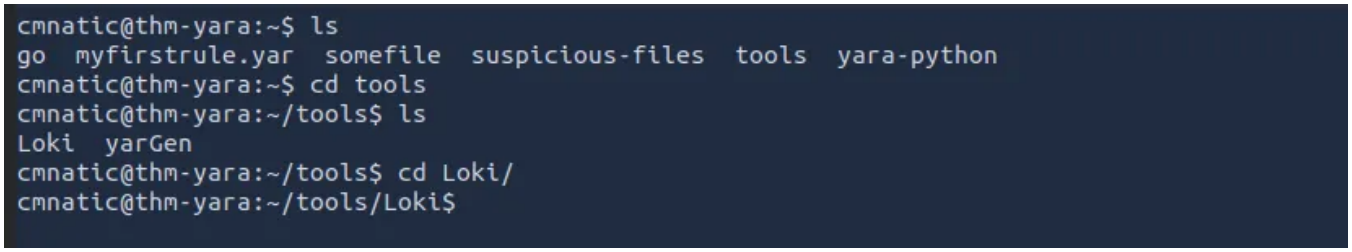
Loki is located in the `tools`.

Listing the tools directory




```
cmnatic@thm-yara:~/tools$ ls
Loki  yarGen
```

Navigate to the `Loki` directory.




```
cd Loki/
```

Run `python loki.py -h` to see what options are available.

```
Loki - Simple IOC Scanner

optional arguments:
  -h, --help            show this help message and exit
  -p path               Path to scan
  -s kilobyte           Maximum file size to check in KB (default 5000 KB)
  -l log-file           Log file
  -r remote-loghost     Remote syslog system
  -t remote-syslog-port
                        Remote syslog port
  -a alert-level        Alert score
  -w warning-level      Warning score
  -n notice-level       Notice score
  --printall            Print all files that are scanned
  --allreasons          Print all reasons that caused the score
  --noprocscan          Skip the process scan
  --nofilescan          Skip the file scan
  --nolevcheck          Skip the Levenshtein distance check
  --scriptanalysis      Activate script analysis (beta)
  --rootkit             Skip the rootkit check
  --noindicator         Do not show a progress indicator
  --reginfs             Do check for Regin virtual file system
  --dontwait            Do not wait on exit
  --intense             Intense scan mode (also scan unknown file types and
                        all extensions)
  --csv                 Write CSV log format to STDOUT (machine processing)
  --onlyrelevant        Only print warnings or alerts
  --nolog               Don't write a local log file
  --update              Update the signatures from the "signature-base" sub
                        repository
  --debug               Debug output
  --maxworkingset MAXWORKINGSET
                        Maximum working set size of processes to scan (in MB,
                        default 100 MB)
  --syslogtcp           Use TCP instead of UDP for syslog logging
  --logfolder log-folder
                        Folder to use for logging when log file is not
                        specified
  --nopesieve           Do not perform pe-sieve scans
  --pesieveshellc       Perform pe-sieve shellcode scan
  --nolisten            Dot not show listening connections
  --excludeprocess EXCLUDEPROCESS
                        Specify an executable name to exclude from scans, can
                        be used multiple times
```

If you are running Loki on your own system, the first command you should run is `--update`. This will add the `signature-base` directory, which Loki uses to scan for known evil. This command was already executed within the attached VM.

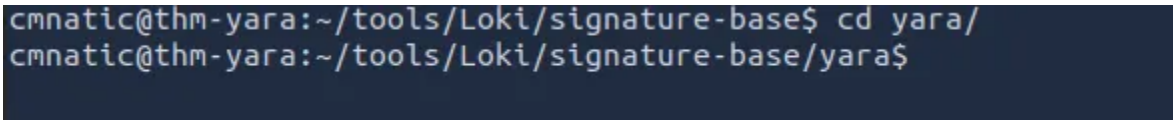
Listing Loki signature-base directory

```
cmnatic@thm-yara:~/tools/Loki$ ls
build.bat     loki-noprivrules.spec             loki-upgrader.py              requirements.txt
config        loki-noprivrules-win2003sup.spec  loki-upgrader.spec            screens
lib           loki-package-builder.py           loki-upgrader-win2003sup.spec signature-base
LICENSE       loki-privrules.spec               plugins                       test
loki.ico      loki-privrules-win2003sup.spec    prepare_push.sh               tools
lokiicon.jpg  loki.py                           README.md
cmnatic@thm-yara:~/tools/Loki$ cd signature-base/
cmnatic@thm-yara:~/tools/Loki/signature-base$ ls
iocs  misc  yara
cmnatic@thm-yara:~/tools/Loki/signature-base$
```

cd

```
cmnatic@thm-yara:~/tools/Loki/signature-base$ ls
iocs  misc  yara
```

Navigate to the `yara` directory.




```
cd yara/
```

Feel free to inspect the different Yara files used by Loki to get an idea of what these rules will hunt for.

To run Loki, you can use the following command (**note that I am calling Loki from within the file 1 directory**)

Instructing Loki to scan the suspicious file

```
cmnatic@thm-yara:~/suspicious-files/file1$ python ../../tools/Loki/loki.py -p .
```

Scenario: You are the security analyst for a mid-size law firm. A co-worker discovered suspicious files on a web server within your organization. These files were discovered while performing updates to the corporate website. The files have been copied to your machine for analysis. The files are located in the `suspicious-files` directory. Use Loki to answer the questions below.

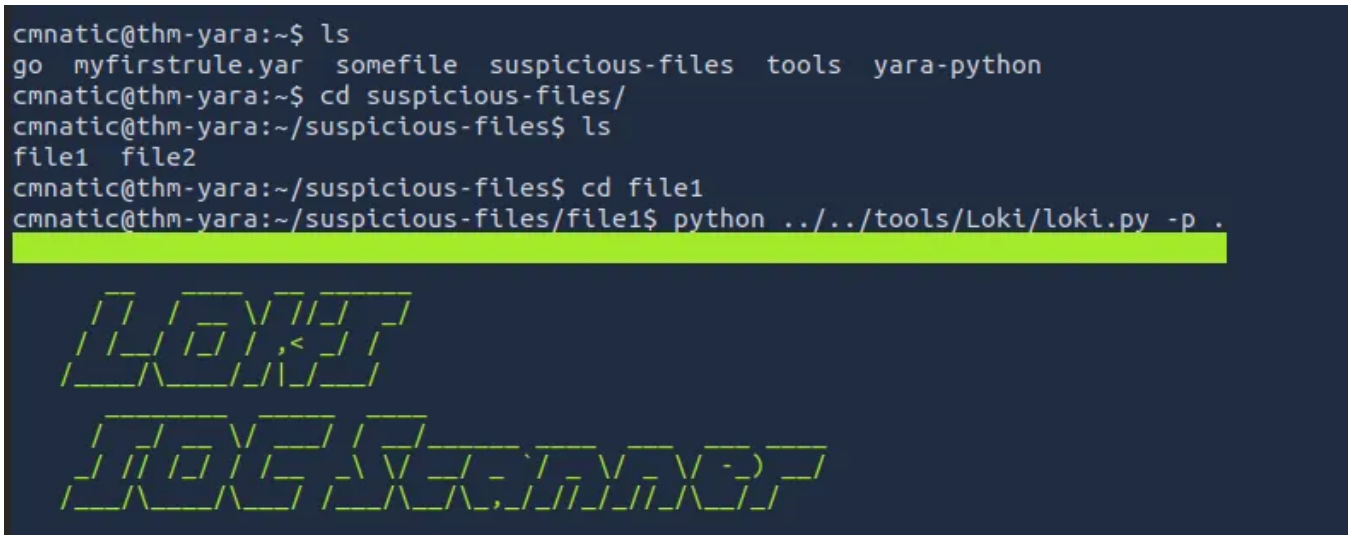
Answer the questions below

8.1 Scan file 1. Does Loki detect this file as suspicious/malicious or benign?

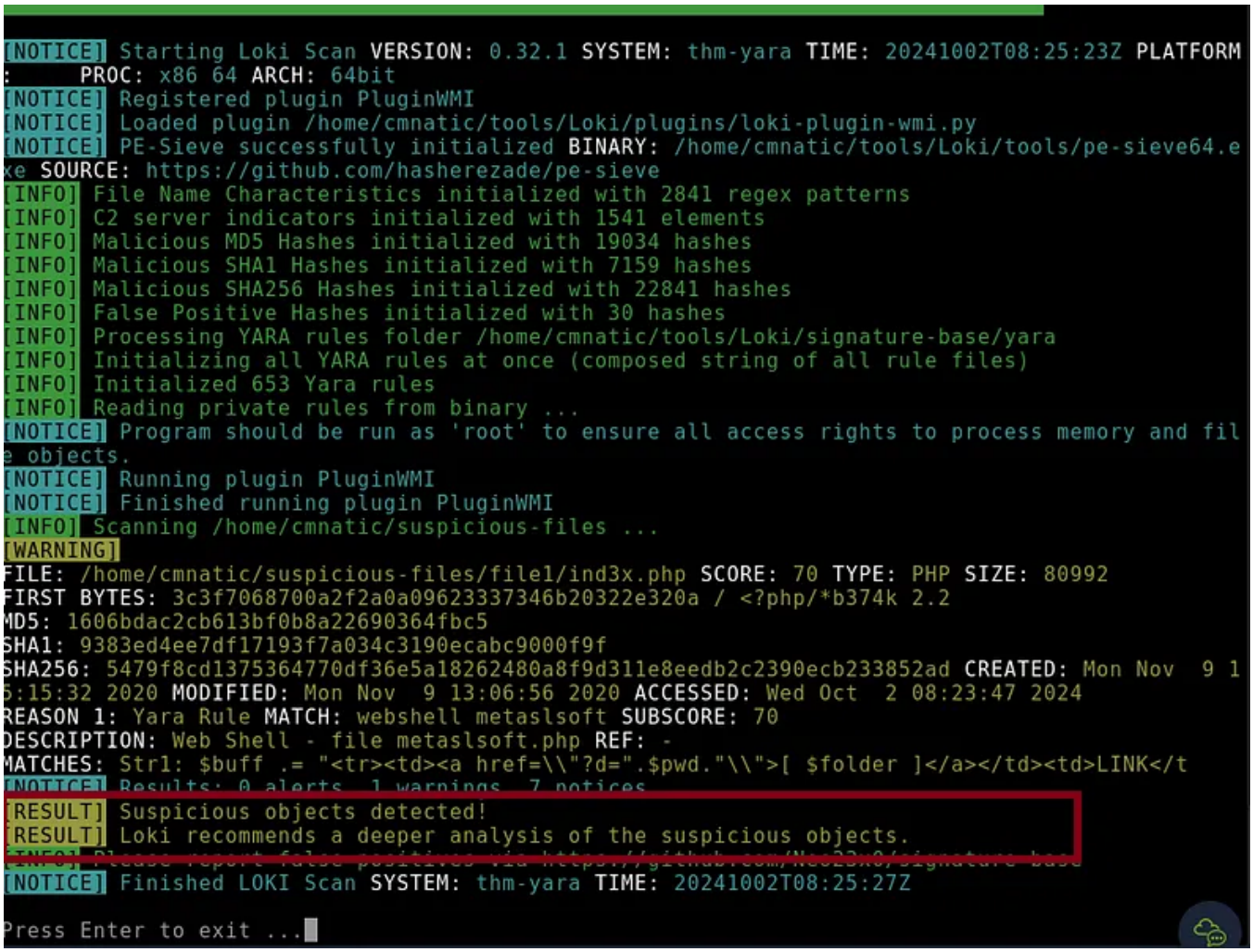
Listed the contents of the home directory

Navigate to suspicious-files and listed the Contents

Navigate to file1 and run the **loki.py** Script

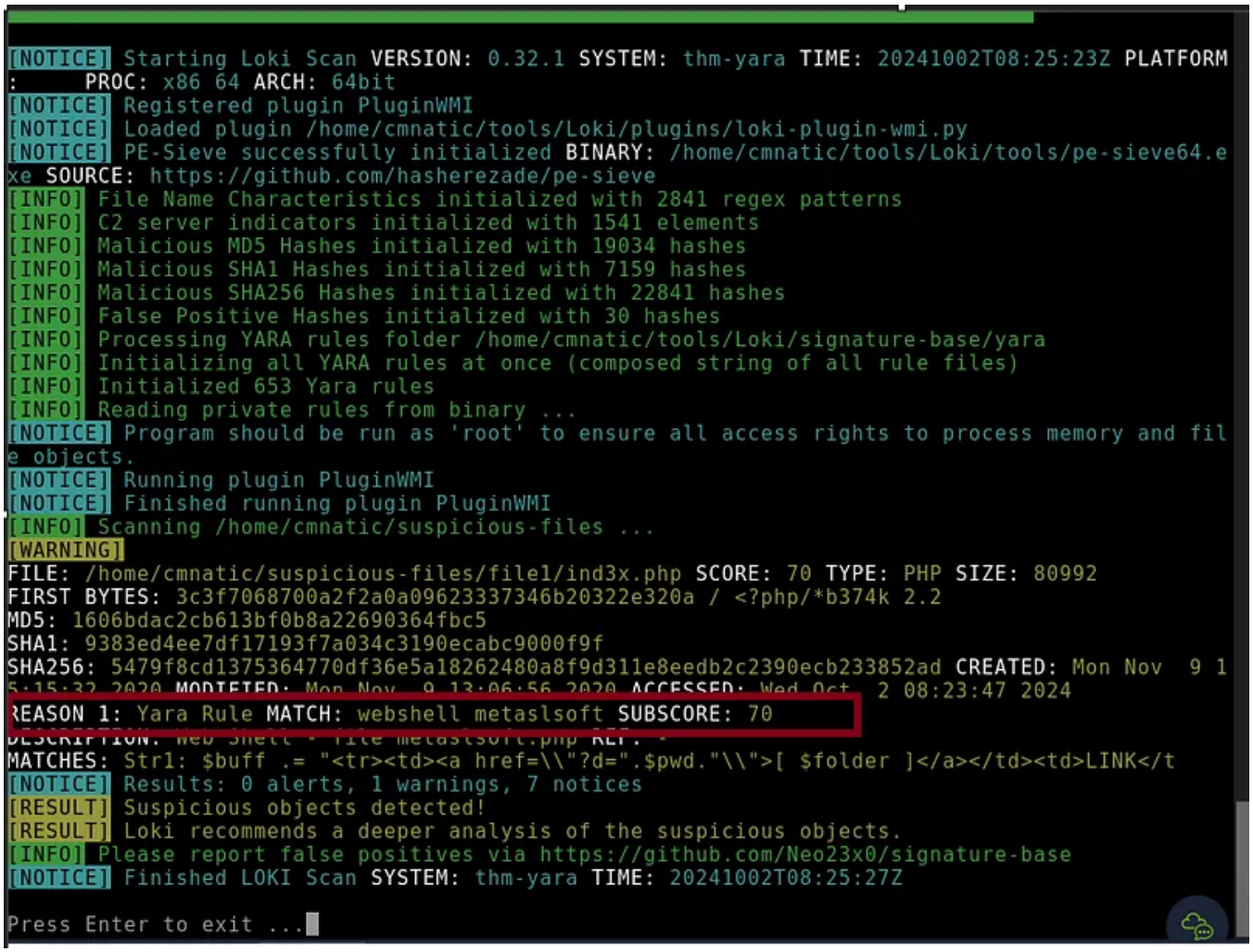



```
cd ~/suspicious-files
ls
cd file1
python ../../tools/Loki/loki.py -p .
```



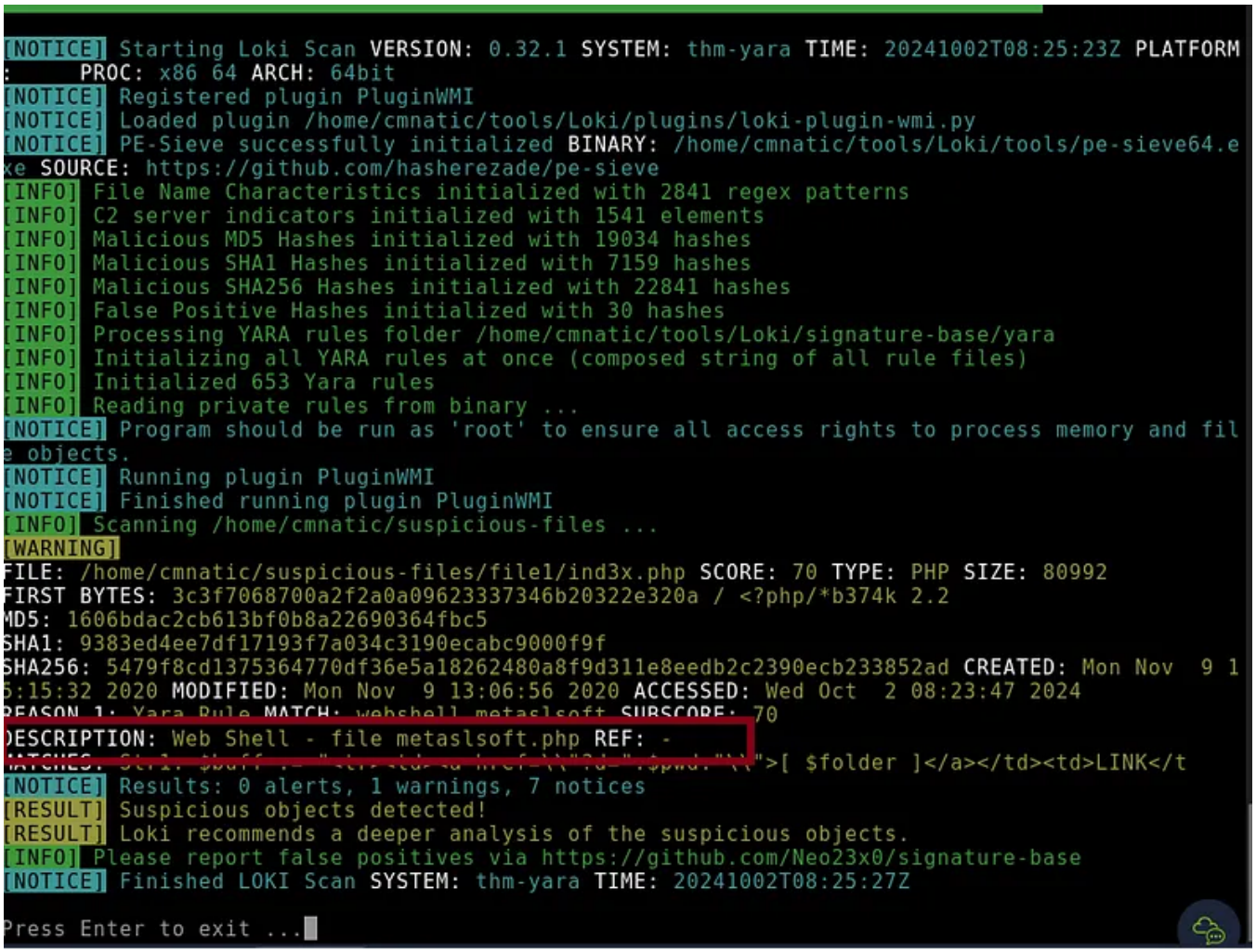
Answer: **Suspicious**

8.2 What Yara rule did it match on?



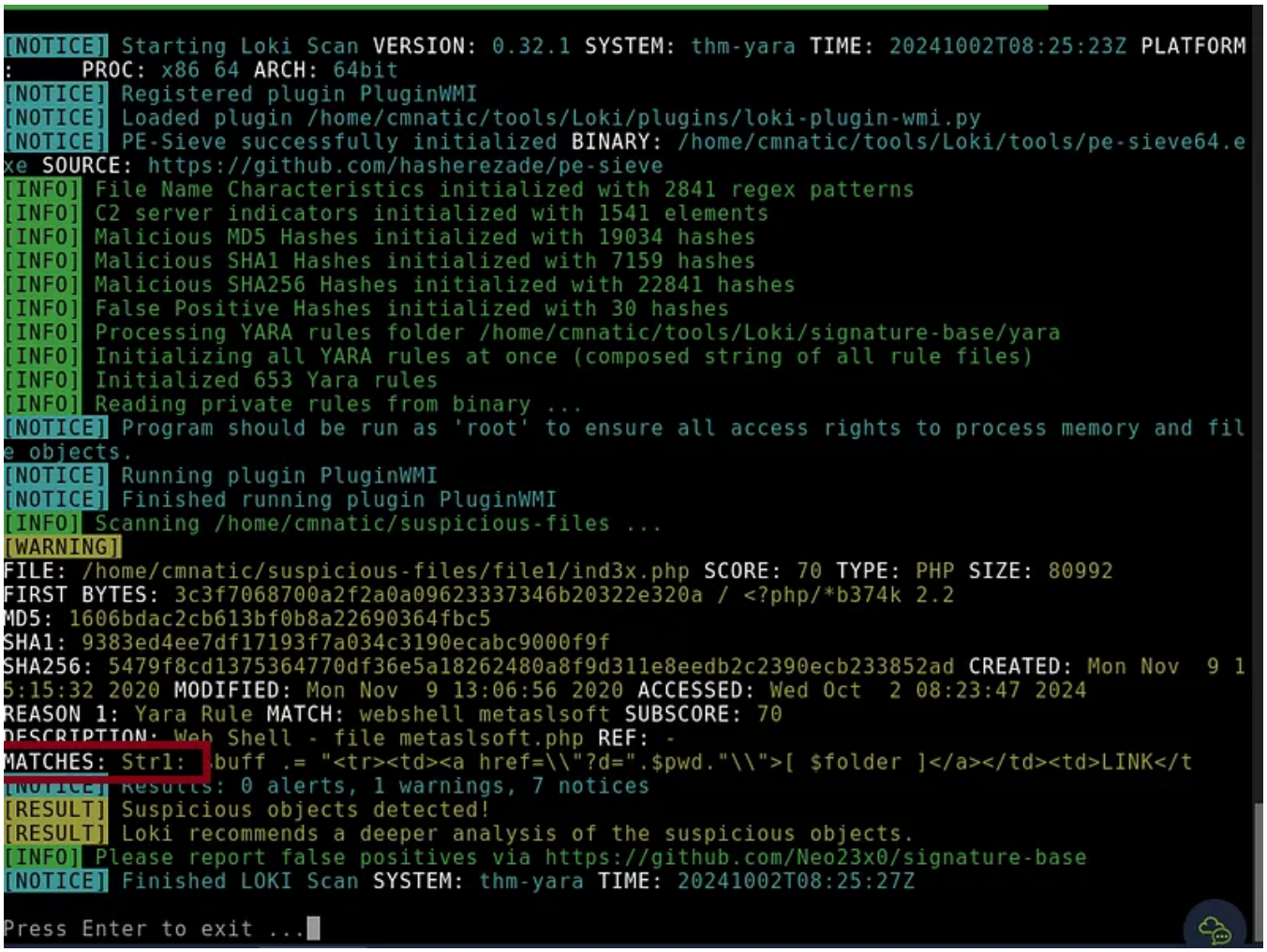
Answer: **webshell_metaslsoft**

8.3 What does Loki classify this file as? (Question Hint Check description)



Answer: **Web shell**

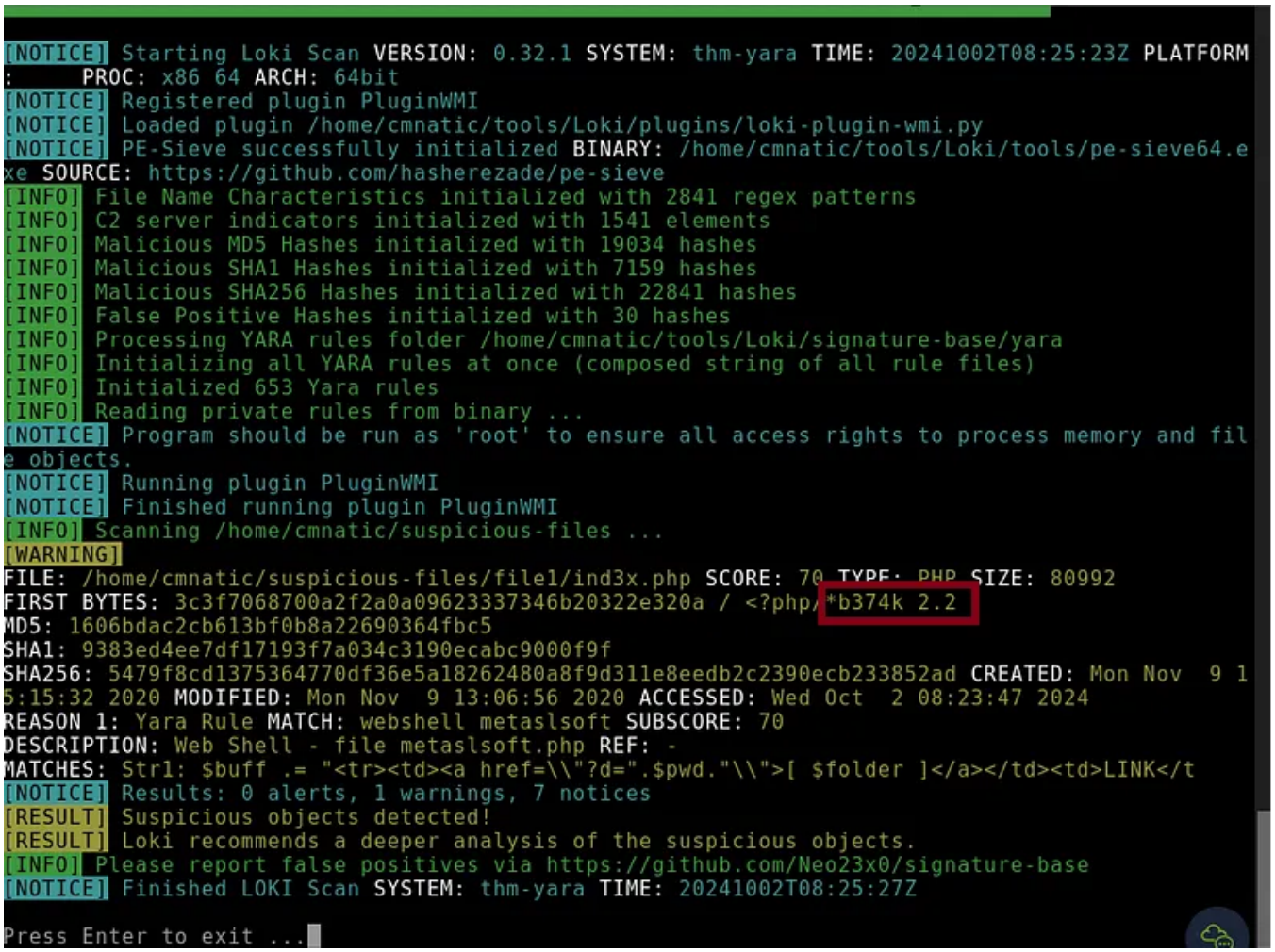
8.4 Based on the output, what string within the Yara rule did it match on?



Answer: **str1**

8.5 What is the name and version of this hack tool?

The first bytes show that the file starts with a PHP tag (<?php), followed by the string /*b374k 2.2, which reveals that the file contains a b374k web shell tool, version 2.2.



Answer: **b374k 2.2**

8.6 Inspect the actual Yara file that flagged file 1. Within this rule, how many strings are there to flag this file? (Question Hint yara/thor-webshells.yar)

Navigate to the **signature-base/yara** directory

Navigating to signature-basels

```
cmnatic@thm-yara:~/suspicious-files/file1$ cd ~/tools/Loki/signature-base/yara
cmnatic@thm-yara:~/tools/Loki/signature-base/yara$
```

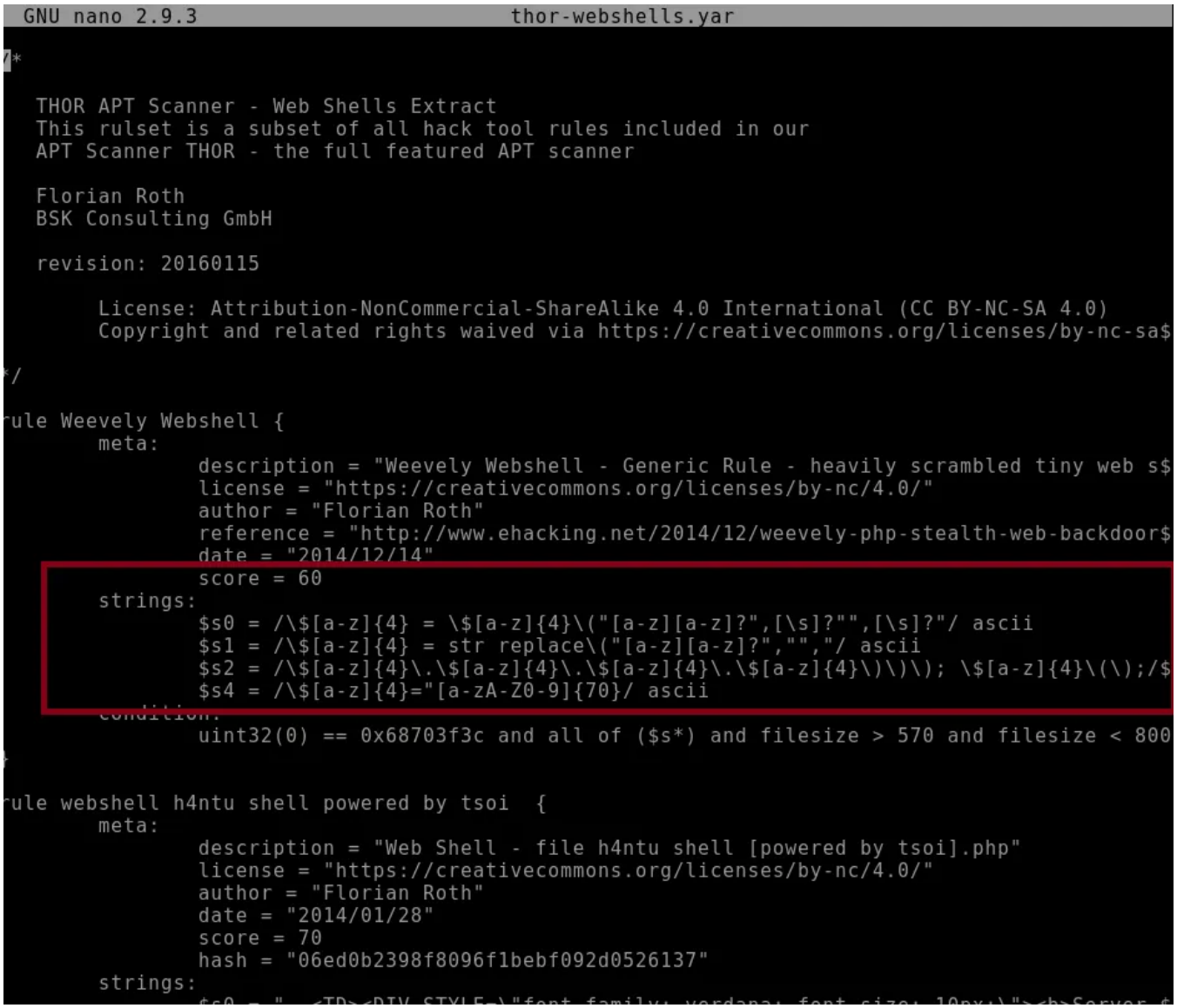
```
cd ~/tools/Loki/signature-base/yara
```

List the Yara files in the directory


```
ls
```

Inspect Yara rule thor-webshells.yar using nano text editor to see how many strings are used for flagging files


```
nano thor-webshells.yar
```



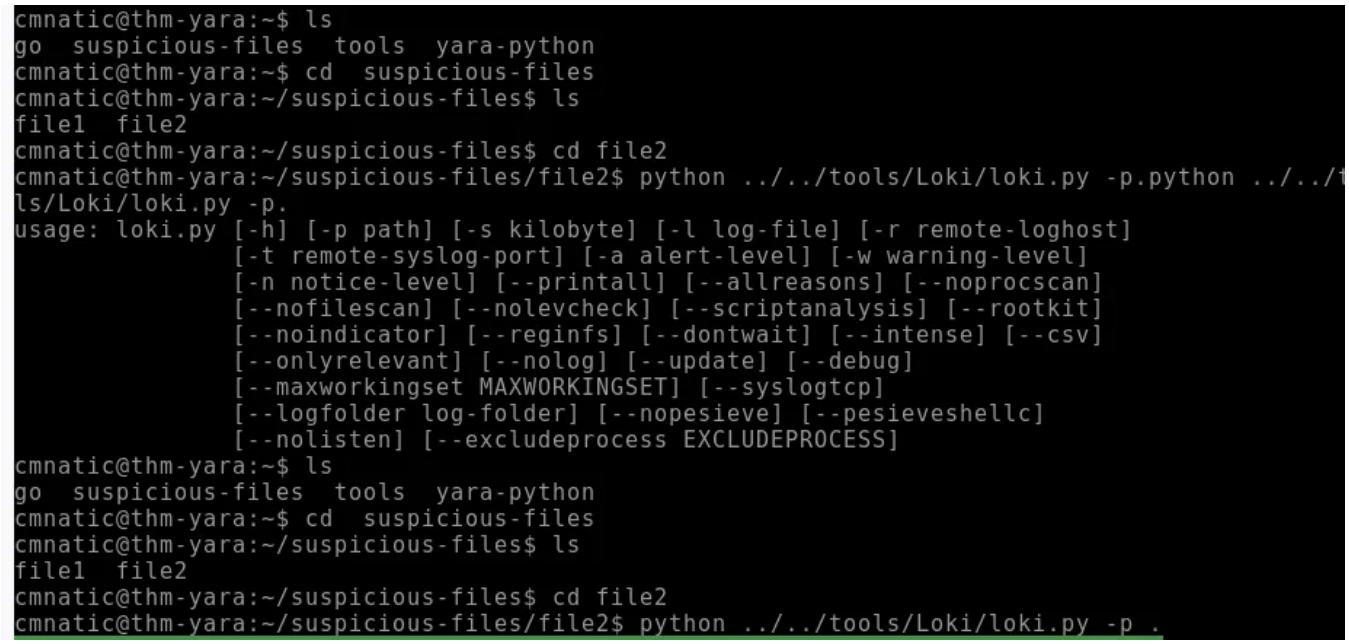
Any one of the strings that match a part of the file will trigger the rule. This means if LOKI finds just one of the patterns (“eval(base64_decode(“), it will flag the file.

The Yara rule you are inspecting has multiple strings, but **only one** string needs to match in the file for it to be flagged as suspicious or malicious.

Answer: **1**

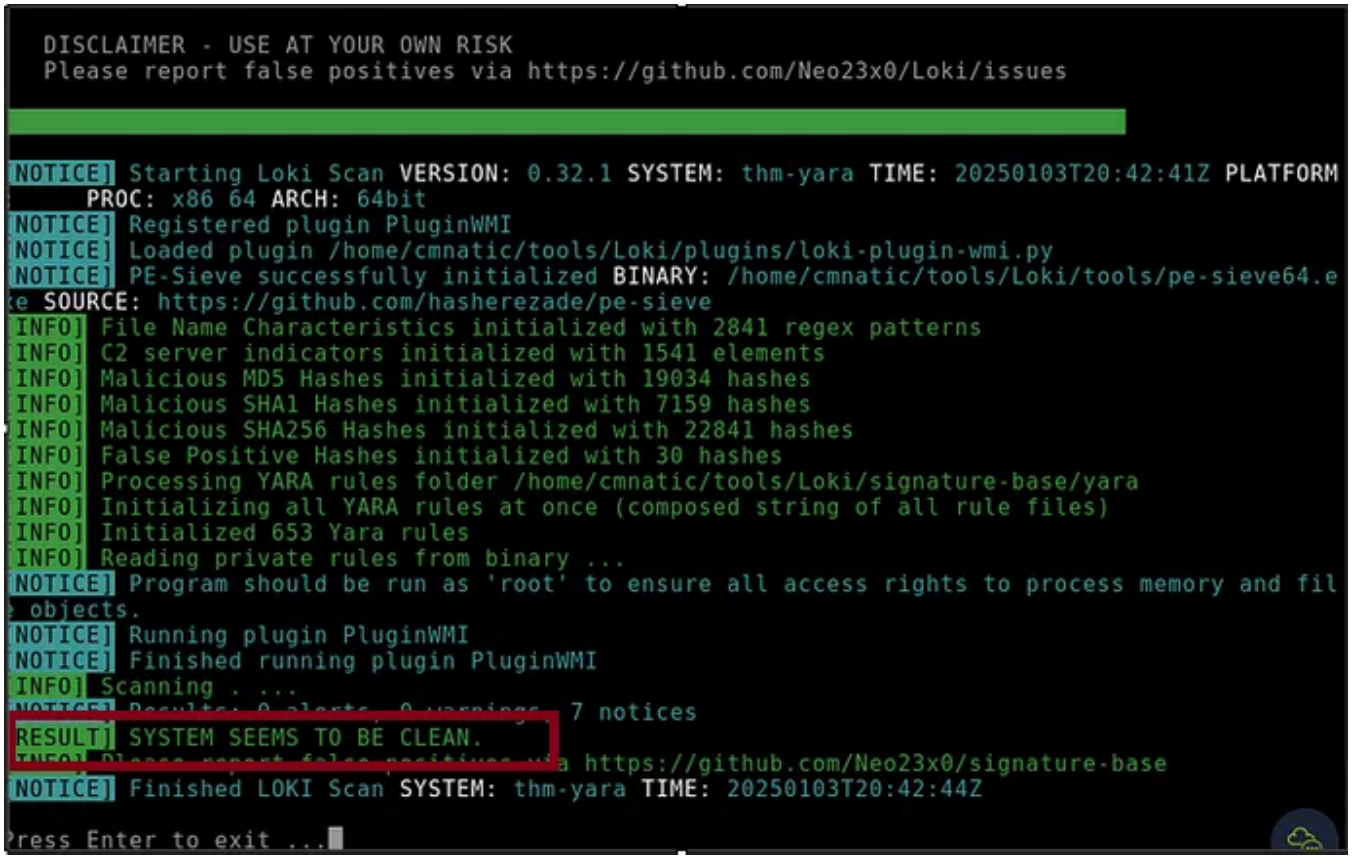
8.7 Scan file 2. Does Loki detect this file as suspicious/malicious or benign?

Navigate to file 2 and Run LOKI to Scan




```
ls
cd ~/suspicious-files
ls
cd file2
Python ../../tools/Loki/loki.py -p .
```

The scan results indicate that **LOKI** did not detect anything suspicious in file 2.

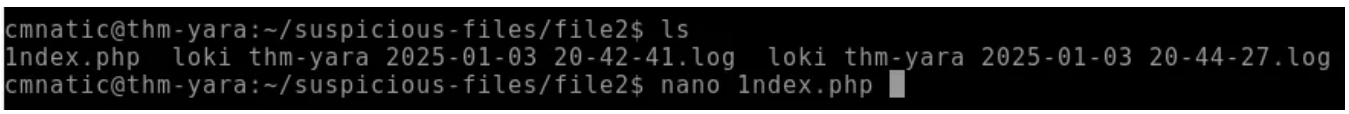


This suggests that **file 2** is **benign** and does not pose any security threat based on the scan.

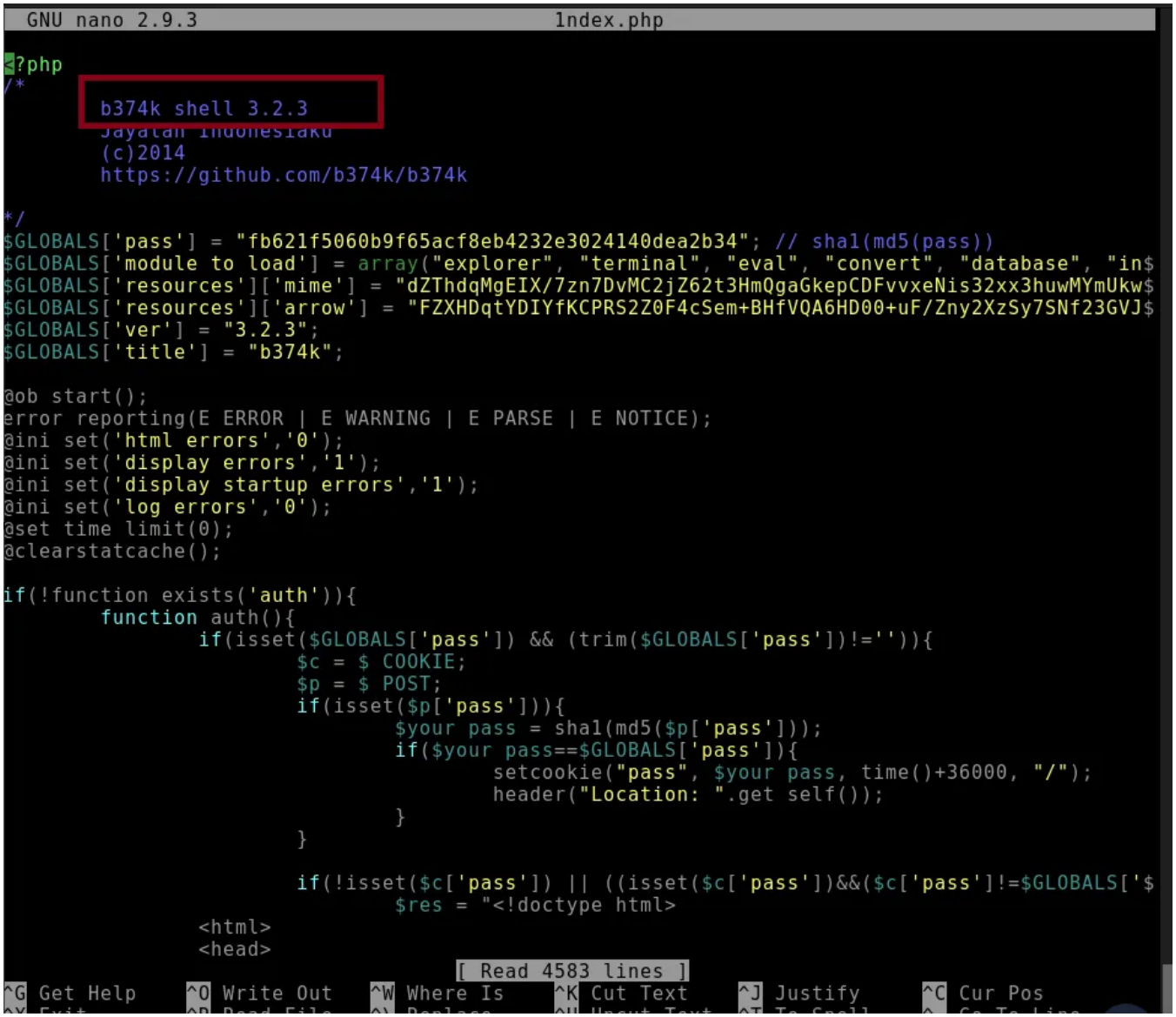
Answer: **Benign**

8.8 Inspect file 2. What is the name and version of this web shell? (Question Hint Read the comments in the file)

Display the contents of **1ndex.php**




```
ls
nano 1ndex.php
```



b374k is a **web shell** that allows attackers to control a compromised web server. It is a popular web shell used in **penetration testing** and **web attacks.**

The version of this **b374k shell** is **3.2.3.**

Answer: **b374k 3.2.3**

Task 9 Creating Yara rules with yarGen

Creating Yara rules with yarGen

From the previous section, we realized that we have a file that Loki didn't flag on. At this point, we are unable to run Loki on other web servers because if **file 2** exists in

any of the webs servers, it will go undetected.

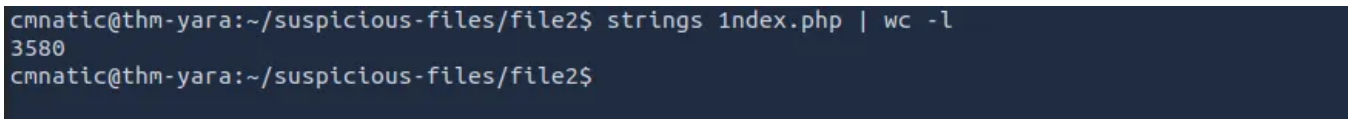
We need to create a Yara rule to detect this specific web shell in our environment. Typically this is what is done in the case of an incident, which is an event that affects/impacts the organization in a negative fashion.

We can manually open the file and attempt to sift through lines upon lines of code to find possible strings that can be used in our newly created Yara rule.

Let's check how many lines this particular file has. You can run the following:

```
strings <file name> | wc -l .
```

Using wc to count the amount of lines in the file




```
cmnatic@thm-yara:~/suspicious-files/file2$ strings 1ndex.php | wc -l
3580
```

If you try to go through each string, line by line manually, you should quickly realize that this can be a daunting task.

Catting the output of 1ndex.php

```
if(res=='error'){
$('.ulProgress'+ulType+i).html('( failed )');
}
else{
$('.ulRes'+ulType+i).html(res);
}
loading_stop();
},
error: function(){
loading_stop();
```

```
$('.ulProgress'+ulType+i).html('( failed )');
$('.ulProgress'+ulType+i).removeClass('ulProgress'+ulType+i);
$('.ulFilename'+ulType+i).removeClass('ulFilename'+ulType+i);
}
});
}
```

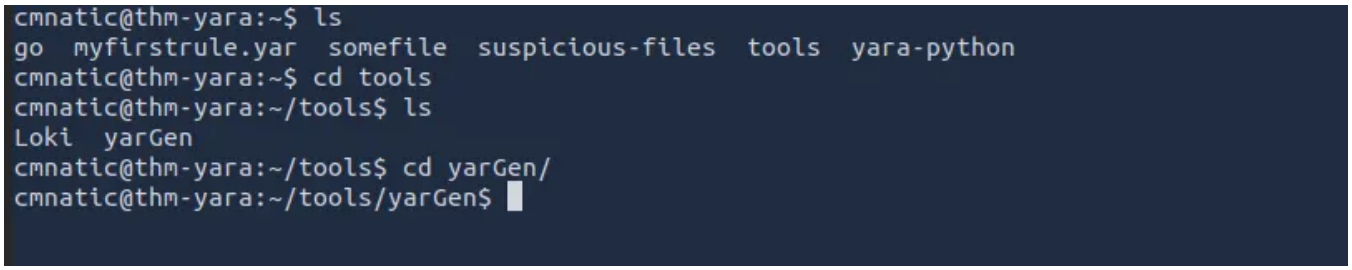
```
function ul_go(ulType){
ulFile = (ulType=='comp')? $('.ulFileComp'):$('.ulFileUrl');
ulResult = (ulType=='comp')? $('.ulCompResult'):$('.ulUrlResult');
ulResult.html('');

ulFile.each(function(i){
if(((ulType=='comp')&&this.files[0])||((ulType=='url')&&(this.value!
=''))){
file = (ulType=='comp')? this.files[0]: this.value;
filename = (ulType=='comp')? file.name:
file.substring(file.lastIndexOf('/')+1);

ulSaveTo = (ulType=='comp')? $('.ulSaveToComp')[i].value:
$('.ulSaveToUrl')[i].value;
ulFilename = (ulType=='comp')? $('.ulFilenameComp')[i].value:
$('.ulFilenameUrl')[i].value;

--snippet cropped for brevity--
```

Luckily, we can use **yarGen** (yes, another tool created by Florian Roth) to aid us with this task.

What is yarGen? yarGen is a generator for YARA rules.

From the README — *"The main principle is the creation of yara rules from strings found in malware files while removing all strings that also appear in goodware files. Therefore yarGen includes a big goodware strings and opcode database as ZIP archives that have to be extracted before the first use."*

Navigate to the `yarGen` directory, which is within `tools`.




```
cd tools
ls
cd yarGen/
```

If you are running yarGen on your own system, you need to update it first by running the following command: `python3 yarGen.py --update`.

This will update the good-opcodes and good-strings DB's from the online repository. This update will take a few minutes.

Once it has been updated successfully, you'll see the following message at the end of the output.

Updating yarGen

```
cmnatic@thm-yara:~/tools/yarGen$ python3 yarGen.py --update
--------------------------------------------------------------------------------
                    _____
    __ _____ _____/ ___/__ ___
   / // / _ `/ __/ (_ / -_) _ \
   \_, /\_,_/_/  \___/\__/_//_/
  /___/  Yara Rule Generator
         Florian Roth, July 2020, Version 0.23.3

   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 ...
```

To use yarGen to generate a Yara rule for file 2, you can run the following command:

```
cmnatic@thm-yara:~/tools/yarGen$ python3 yarGen.py -m /home/cmnatic/suspicious-files/file2 --excludegood -
o /home/cmnatic/suspicious-files/file2.yar
------------------------------------------------------------------------
                    _____
   __ _____ _____/ ___/__ ___
  / // / _ `/ __/ (_ / -_) _ \
  \_, /\_,_/_/  \___/\__/_//_/
 /___/  Yara Rule Generator
        Florian Roth, July 2020, Version 0.23.3

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

```
python3 yarGen.py -m /home/cmnatic/suspicious-files/file2 --excludegood -o /hom
```

A brief explanation of the parameters above:

- `-m` is the path to the files you want to generate rules for
- `--excludegood` force to exclude all goodware strings (*these are strings found in legitimate software and can increase false positives*)
- `-o` location & name you want to output the Yara rule

If all is well, you should see the following output.

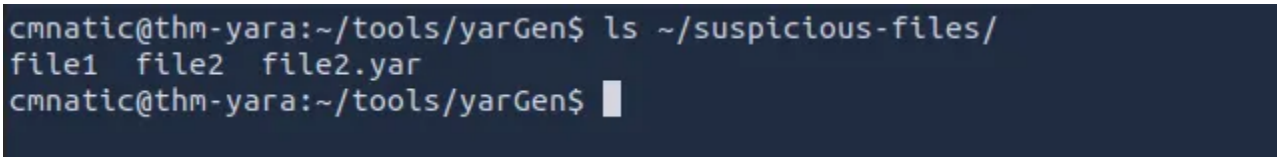
Using yarGen to generate a rule for file2

```
[=] Generated 1 SIMPLE rules.
[=] All rules written to /home/cmnatic/suspicious-files/file2.yar
[+] yarGen run finished
cmnatic@thm-yara:~/tools/yarGen$
```

```
[=] Generated 1 SIMPLE rules.
            [=] All rules written to /home/cmnatic/suspicious-files/file2.yar
            [+] yarGen run finished
```

Check if the YARA rule was created

If **file2.yar** appears, the rule was successfully generated.




```
ls ~/suspicious-files/
```

Generally, you would examine the Yara rule and remove any strings that you feel might generate false positives. For this exercise, we will leave the generated Yara rule as is and test to see if Yara will flag file 2 or no.

Note: Another tool created to assist with this is called **yarAnalyzer** (you guessed it — created by Florian Roth). We will not examine that tool in this room, but you should read up on it, especially if you decide to start creating your own Yara rules.

Further Reading on creating Yara rules and using yarGen:

- https://www.bsk-consulting.de/2015/02/16/write-simple-sound-yara-rules/
- https://www.bsk-consulting.de/2015/10/17/how-to-write-simple-but-sound-yara-rules-part-2/
- https://www.bsk-consulting.de/2016/04/15/how-to-write-simple-but-sound-yara-rules-part-3/

Answer the questions below

9.1 From within the root of the suspicious files directory, what command would you run to test Yara and your Yara rule against file 2? (Question Hint Use the same name I called the Yara file to answer this question)

```
cmnatic@thm-yara:~/suspicious-files$ yara file2.yar file2/1ndex.php
_home_cmnatic_suspicious_files_file2_1ndex file2/1ndex.php
cmnatic@thm-yara:~/suspicious-files$
```

```
yara file2.yar file2/1ndex.php
```

We are inside the suspicious-files directory, so we do not need the full path (~/suspicious-files/).
file2.yar > This is the YARA rule generated earlier.
file2/1ndex.php > This is the actual suspicious web shell file inside the file2 directory.
Running this command applies the YARA rule to **1ndex.php**, checking for any matches.

Answer: **yara file2.yar file2/1ndex.php**

9.2 Did Yara rule flag file 2? (Yay/Nay)

YARA flagged **1ndex.php** as suspicious.
The rule worked, and the file matches the patterns in **file2.yar**.

Answer: **Yay**

9.3 Copy the Yara rule you created into the Loki signatures directory.

```
cmnatic@thm-yara:~/suspicious-files$ cp ~/suspicious-files/file2.yar /home/cmnatic/tools/Loki/signature-ba
se/
```

```
cp ~/suspicious-files/file2.yar /home/cmnatic/tools/Loki/signature-base/ls
```

Verify that the file was copied successfully

```
cmnatic@thm-yara:~/suspicious-files$ ls /home/cmnatic/tools/Loki/signature-base/
file2.yar  iocs  misc  yara
cmnatic@thm-yara:~/suspicious-files$
```

```
ls /home/cmnatic/tools/Loki/signature-base/
```

Answer: **No answer nedeed**

9.4 Test the Yara rule with Loki, does it flag file 2? (Yay/Nay)

Answer: **Yay**

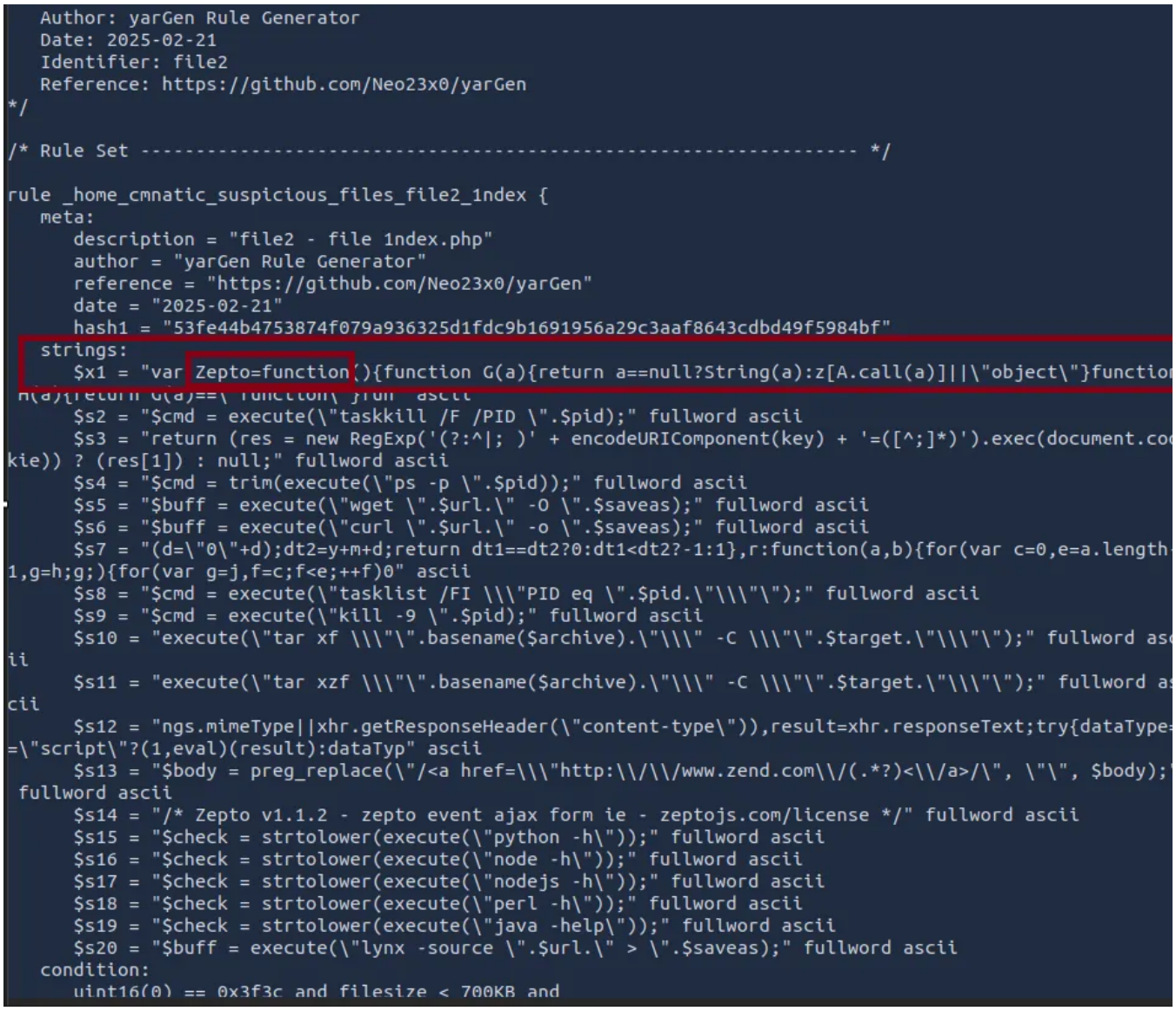
9.5 What is the name of the variable for the string that it matched on? (Question Hint Look at $x1)

Checked the contents of **file2.yar** to identify the specific string that was flagged by your YARA rule.

```
cmnatic@thm-yara:~/tools/Loki/signature-base$ ls
file2.yar  iocs  misc  yara
cmnatic@thm-yara:~/tools/Loki/signature-base$ cat file2.yar
/*
```

```
ls
cat file2.yar
```

The actual content of **$x1** is the string that YARA matched.



```
   Author: yarGen Rule Generator
   Date: 2025-02-21
   Identifier: file2
   Reference: https://github.com/Neo23x0/yarGen
*/

/* Rule Set ----------------------------------------------------------------- */

rule _home_cmnatic_suspicious_files_file2_1ndex {
   meta:
      description = "file2 - file 1ndex.php"
      author = "yarGen Rule Generator"
      reference = "https://github.com/Neo23x0/yarGen"
      date = "2025-02-21"
      hash1 = "53fe44b4753874f079a936325d1fdc9b1691956a29c3aaf8643cdbd49f5984bf"
   strings:
      $x1 = "var Zepto=function(){function G(a){return a==null?String(a):z[A.call(a)]||\"object\"}functio
 n(a){return G(a)==\"function\"}fun" ascii
      $s2 = "$cmd = execute(\"taskkill /F /PID \".$pid);" fullword ascii
      $s3 = "return (res = new RegExp('(?:^|; )' + encodeURIComponent(key) + '=([^;]*)').exec(document.co
kie)) ? (res[1]) : null;" fullword ascii
      $s4 = "$cmd = trim(execute(\"ps -p \".$pid));" fullword ascii
      $s5 = "$buff = execute(\"wget \".$url.\" -O \".$saveas);" fullword ascii
      $s6 = "$buff = execute(\"curl \".$url.\" -o \".$saveas);" fullword ascii
      $s7 = "(d=\"0\"+d);dt2=y+m+d;return dt1==dt2?0:dt1<dt2?-1:1},r:function(a,b){for(var c=0,e=a.length-
1,g=h;g;){for(var g=j,f=c;f<e;++f)0" ascii
      $s8 = "$cmd = execute(\"tasklist /FI \\\"PID eq \".$pid.\"\\\"\");" fullword ascii
      $s9 = "$cmd = execute(\"kill -9 \".$pid);" fullword ascii
      $s10 = "execute(\"tar xf \\\"\".basename($archive).\"\\\" -C \\\"\".$target.\"\\\"\");" fullword asc
ii
      $s11 = "execute(\"tar xzf \\\"\".basename($archive).\"\\\" -C \\\"\".$target.\"\\\"\");" fullword as
cii
      $s12 = "ngs.mimeType||xhr.getResponseHeader(\"content-type\")),result=xhr.responseText;try{dataType=
=\"script\"?(1,eval)(result):dataTyp" ascii
      $s13 = "$body = preg_replace(\"/<a href=\\\"http:\\/\\/www.zend.com\\/(.*?)<\\/a>/\", \"\", $body);"
 fullword ascii
      $s14 = "/* Zepto v1.1.2 - zepto event ajax form ie - zeptojs.com/license */" fullword ascii
      $s15 = "$check = strtolower(execute(\"python -h\"));" fullword ascii
      $s16 = "$check = strtolower(execute(\"node -h\"));" fullword ascii
      $s17 = "$check = strtolower(execute(\"nodejs -h\"));" fullword ascii
      $s18 = "$check = strtolower(execute(\"perl -h\"));" fullword ascii
      $s19 = "$check = strtolower(execute(\"java -help\"));" fullword ascii
      $s20 = "$buff = execute(\"lynx -source \".$url.\" > \".$saveas);" fullword ascii
   condition:
      uint16(0) == 0x3f3c and filesize < 700KB and
```

This means the rule flagged file2 because it detected the presence of the **Zepto.js** function, which is often associated with certain web shell scripts or JavaScript-based exploits.

Answer: **Zepto**

9.6 Inspect the Yara rule, how many strings were generated?

```
/* Rule Set ----------------------------------------------------------------- */

rule _home_cmnatic_suspicious_files_file2_1ndex {
   meta:
      description = "file2 - file 1ndex.php"
      author = "yarGen Rule Generator"
      reference = "https://github.com/Neo23x0/yarGen"
      date = "2025-02-21"
      hash1 = "53fe44b4753874f079a936325d1fdc9b1691956a29c3aaf8643cdbd49f5984bf"
   strings:
      $x1 = "var Zepto=function(){function G(a){return a==null?String(a):z[A.call(a)]||\"object\"}function
 H(a){retur n G(a)==\"function\"}fun" ascii
      $s2 = "$cmd = execute(\"taskkill /F /PID \".$pid);" fullword ascii
      $s3 = "return (res = new RegExp('(?:^|; )' + encodeURIComponent(key) + '=([^;]*)').exec(document.coo
kie)) ? (re;[1]) : null;" fullword ascii
      $s4 = "$cmd = trim(execute(\"ps -p \".$pid));" fullword ascii
      $s5 = "$buff = execute(\"wget \".$url.\" -O \".$saveas);" fullword ascii
      $s6 = "$buff = execute(\"curl \".$url.\" -o \".$saveas);" fullword ascii
      $s7 = "(d=\"0\"+d);dt2=y+m+d;return dt1==dt2?0:dt1<dt2?-1:1},r:function(a,b){for(var c=0,e=a.length-
1,g=h;g;){f>r(var g=j,f=c;f<e;++f)0" ascii
      $s8 = "$cmd = execute(\"tasklist /FI \\\"PID eq \".$pid.\"\\\"\");" fullword ascii
      $s9 = "$cmd = execute(\"kill -9 \".$pid);" fullword ascii
      $s10 = "execute(\"tar xf \\\"\".basename($archive).\"\\\" -C \\\"\".$target.\"\\\"\");" fullword asc
ii
      $s11 = "execute(\"tar xzf \\\"\".basename($archive).\"\\\" -C \\\"\".$target.\"\\\"\");" fullword as
cii
      $s12 = "ngs.mimeType||xhr.getResponseHeader(\"content-type\")),result=xhr.responseText;try{dataType=
=\"script\"?(1,eval)(result):dataTyp" ascii
      $s13 = "$body = preg_replace(\"/<a href=\\\"http:\\/\\/www.zend.com\\/(.*?)<\\/a>/\", \"\", $body);"
 fullword ascii
      $s14 = "/* Zepto v1.1.2 - zepto event ajax form ie - zeptojs.com/license */" fullword ascii
      $s15 = "$check = strtolower(execute(\"python -h\"));" fullword ascii
      $s16 = "$check = strtolower(execute(\"node -h\"));" fullword ascii
      $s17 = "$check = strtolower(execute(\"nodejs -h\"));" fullword ascii
      $s18 = "$check = strtolower(execute(\"perl -h\"));" fullword ascii
      $s19 = "$check = strtolower(execute(\"java -help\"));" fullword ascii
      $s20 = "$buff = execute(\"lynx -source \".$url.\" > \".$saveas);" fullword ascii
   condition:
      uint16(0) == 0x3f3c and filesize < 700KB and
      1 of ($x*) and 4 of them
}

cmnatic@thm-yara:~/tools/Loki/signature-base$ grep -o '\$x[0-9]\+' ~/tools/Loki/signature-base/file2.yar |
 wc -l
1
```

Number of strings in the Yara rule are 20

Answer: **20**

9.7 One of the conditions to match on the Yara rule specifies file size. The file has to be less than what amount?

```
/* Rule Set ----------------------------------------------------------------- */

rule _home_cmnatic_suspicious_files_file2_1ndex {
   meta:
      description = "file2 - file 1ndex.php"
      author = "yarGen Rule Generator"
      reference = "https://github.com/Neo23x0/yarGen"
      date = "2025-02-21"
      hash1 = "53fe44b4753874f079a936325d1fdc9b1691956a29c3aaf8643cdbd49f5984bf"
   strings:
      $x1 = "var Zepto=function(){function G(a){return a==null?String(a):z[A.call(a)]||\"object\"}function
 H(a){return G(a)==\"function\"}fun" ascii
      $s2 = "$cmd = execute(\"taskkill /F /PID \".$pid);" fullword ascii
      $s3 = "return (res = new RegExp('(?:^|; )' + encodeURIComponent(key) + '=([^;]*)').exec(document.coo
kie)) ? (res[1]) : null;" fullword ascii
      $s4 = "$cmd = trim(execute(\"ps -p \".$pid));" fullword ascii
      $s5 = "$buff = execute(\"wget \".$url.\" -O \".$saveas);" fullword ascii
      $s6 = "$buff = execute(\"curl \".$url.\" -o \".$saveas);" fullword ascii
      $s7 = "(d=\"0\"+d);dt2=y+m+d;return dt1==dt2?0:dt1<dt2?-1:1},r:function(a,b){for(var c=0,e=a.length-
1,g=h;g;){for(var g=j,f=c;f<e;++f)0" ascii
      $s8 = "$cmd = execute(\"tasklist /FI \\\"PID eq \".$pid.\"\\\"\");" fullword ascii
      $s9 = "$cmd = execute(\"kill -9 \".$pid);" fullword ascii
      $s10 = "execute(\"tar xf \\\"\".basename($archive).\"\\\" -C \\\"\".$target.\"\\\"\");" fullword asc
ii
      $s11 = "execute(\"tar xzf \\\"\".basename($archive).\"\\\" -C \\\"\".$target.\"\\\"\");" fullword as
cii
      $s12 = "ngs.mimeType||xhr.getResponseHeader(\"content-type\")),result=xhr.responseText;try{dataType=
=\"script\"?(1,eval)(result):dataTyp" ascii
      $s13 = "$body = preg_replace(\"/<a href=\\\"http:\\/\\/www.zend.com\\/(.*?)<\\/a>/\", \"\", $body);"
 fullword ascii
      $s14 = "/* Zepto v1.1.2 - zepto event ajax form ie - zeptojs.com/license */" fullword ascii
      $s15 = "$check = strtolower(execute(\"python -h\"));" fullword ascii
      $s16 = "$check = strtolower(execute(\"node -h\"));" fullword ascii
      $s17 = "$check = strtolower(execute(\"nodejs -h\"));" fullword ascii
      $s18 = "$check = strtolower(execute(\"perl -h\"));" fullword ascii
      $s19 = "$check = strtolower(execute(\"java -help\"));" fullword ascii
      $s20 = "$buff = execute(\"lynx -source \".$url.\" > \".$saveas);" fullword ascii
   condition:
      uint16(0) == 0x3f3c and filesize < 700KB and
      1 of ($x*) and 4 of them
}

cmnatic@thm-yara:~/tools/Loki/signature-base$ grep -o '\$x[0-9]\+' ~/tools/Loki/signature-base/file2.yar |
 wc -l
1
```

The condition states that the file must be **smaller** than **700KB** for the rule to trigger.

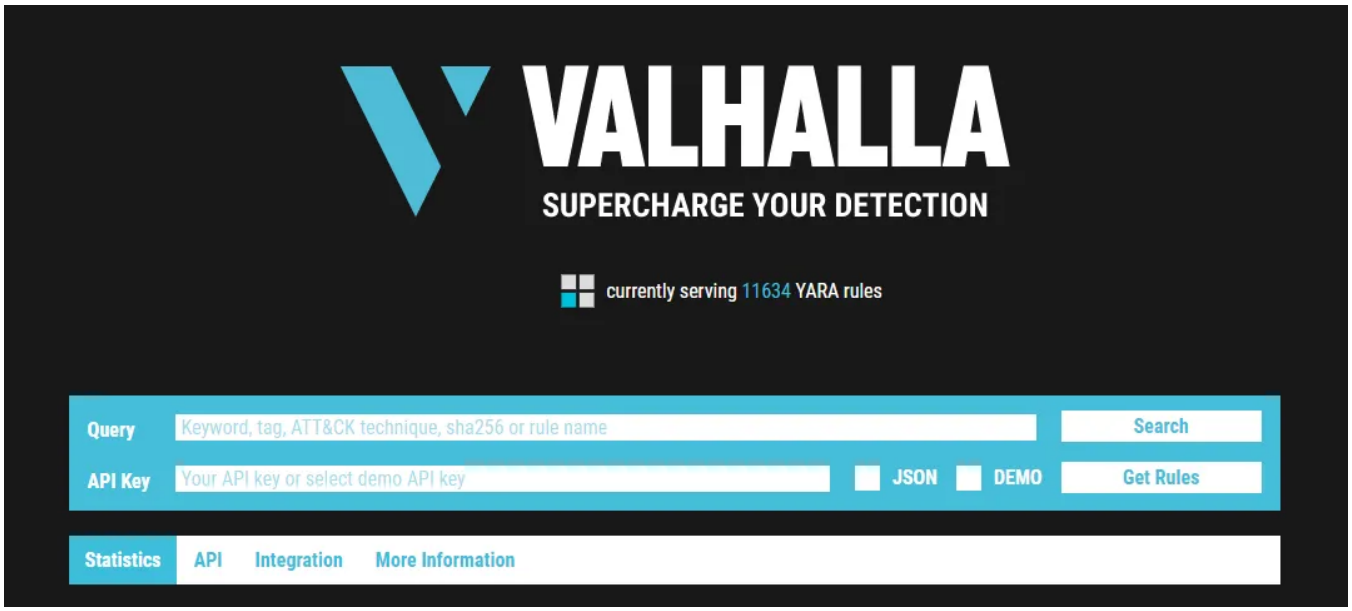
Answer: **700KB**

Task 10 Valhalla

Valhalla

Valhalla is an online Yara feed created and hosted by **Nextron-Systems** (erm, Florian Roth). By now, you should be aware of the ridiculous amount of time and energy Florian has dedicated to creating these tools for the community. Maybe we should have just called this the Florian Roth room. (lol)

Per the website, "*Valhalla boosts your detection capabilities with the power of thousands of hand-crafted high-quality YARA rules.*"



From the image above, we should denote that we can conduct searches based on a keyword, tag, ATT&CK technique, sha256, or rule name.

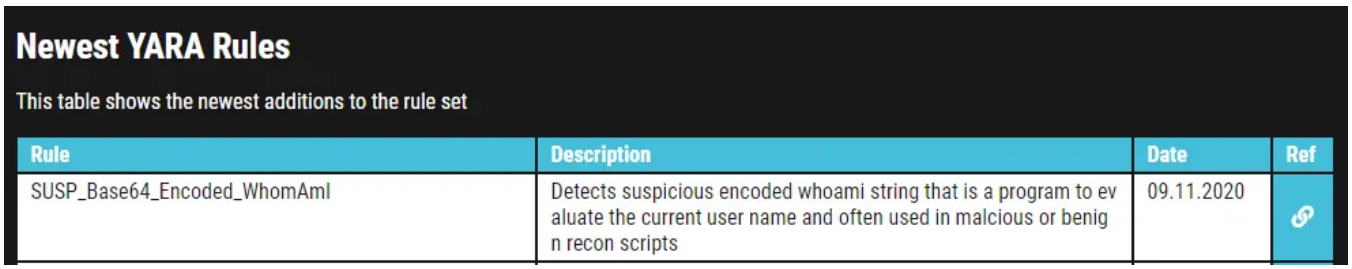
Note: For more information on ATT&CK, please visit the **MITRE** room.

MITRE— SOC Level 1 -Cyber Defence Frameworks — TryHackMe Walkthrough

This room will discuss the various resources MITRE has made available for the cybersecurity community.

iritt.medium.com

Taking a look at the data provided to us, let's examine the rule in the screenshot below:



We are provided with the name of the rule, a brief description, a reference link for

more information about the rule, along with the rule date.

Feel free to look at some rules to become familiar with the usefulness of Valhalla. The best way to learn the product is by just jumping right in.

Picking up from our scenario, at this point, you know that the 2 files are related. Even though Loki classified the files are suspicious, you know in your gut that they are malicious. Hence the reason you created a Yara rule using yarGen to detect it on other web servers. But let's further pretend that you are not code-savvy (FYI — not all security professionals know how to code/script or read it). You need to conduct further research regarding these files to receive approval to eradicate these files from the network.

Time to use Valhalla for some threat intelligence gathering...

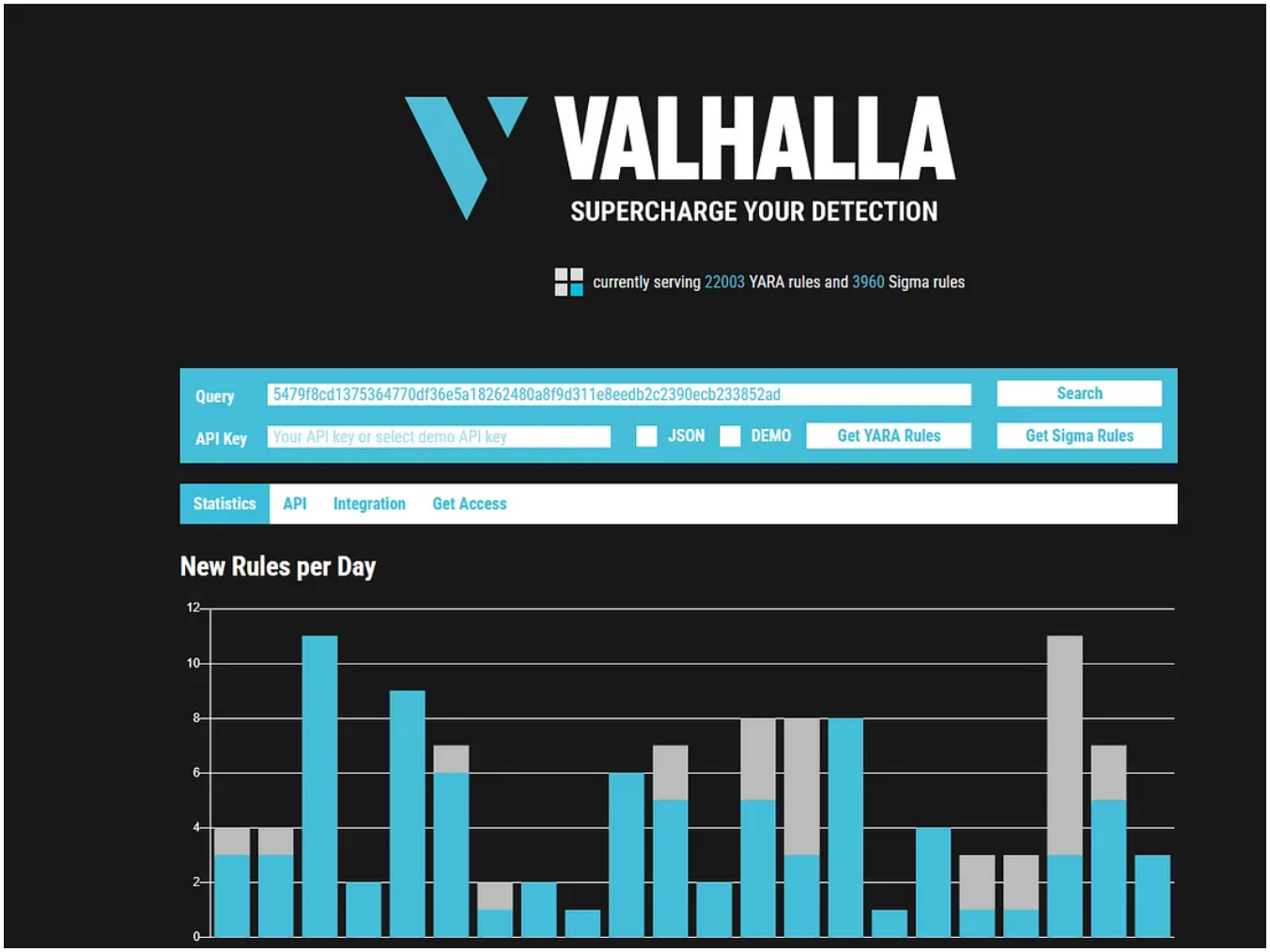
Answer the questions below

10.1 Enter the SHA256 hash of file 1 into Valhalla. Is this file attributed to an APT group? (Yay/Nay)

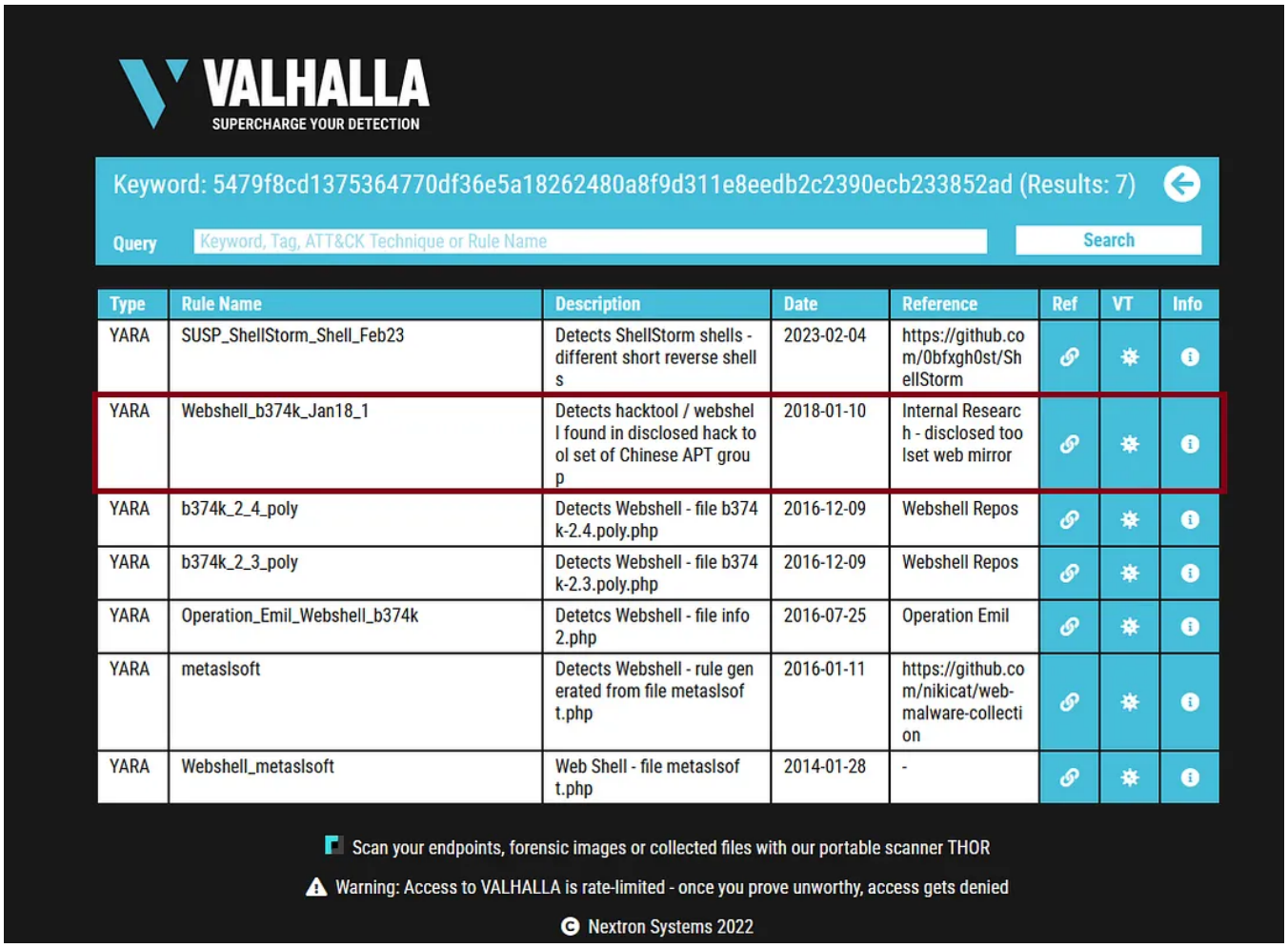
```
cmnatic@thm-yara:~/suspicious-files/file1$ ls
ind3x.php  loki_thm-yara_2025-02-21_12-35-28.log
cmnatic@thm-yara:~/suspicious-files/file1$ sha256sum ind3x.php
5479f8cd1375364770df36e5a18262480a8f9d311e8eedb2c2390ecb233852ad  ind3x.php
cmnatic@thm-yara:~/suspicious-files/file1$
```

```
sha256sum ind3x.php
```

Go to https://valhalla.nextron-systems.com/ and paste the hash




```
5479f8cd1375364770df36e5a18262480a8f9d311e8eedb2c2390ecb233852ad
```



Results

The file was detected by several YARA rules.

One rule, “**Webshell_b374k_Jan18_1**”, has a description that says:

“**Detects hacktool / webshell I found in disclosed toolset of Chinese APT group.**”

This means the file was found in a hacking toolset used by a Chinese APT group.

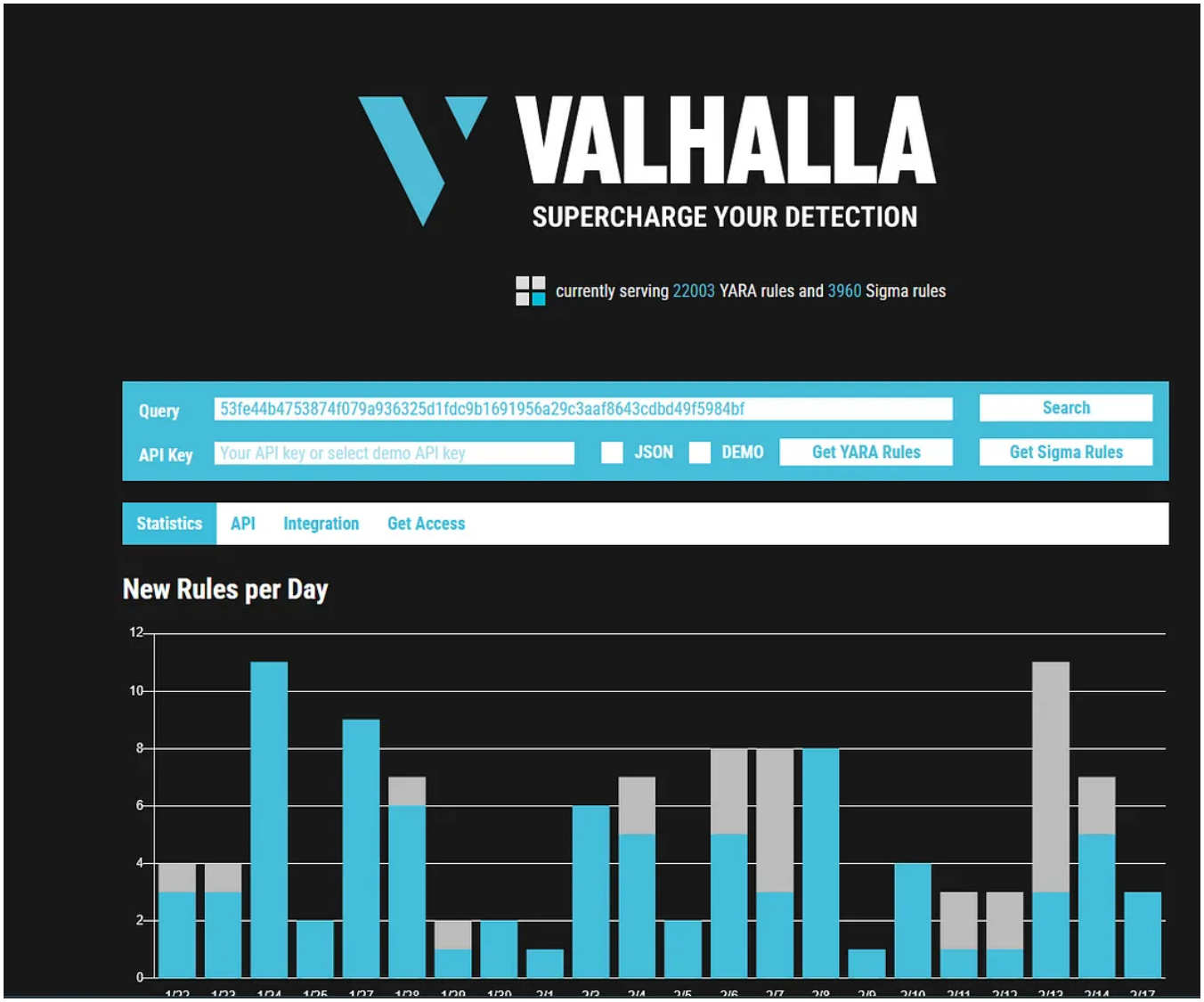
Answer: **Yay**

10.2 Do the same for file 2. What is the name of the first Yara rule to detect file 2?

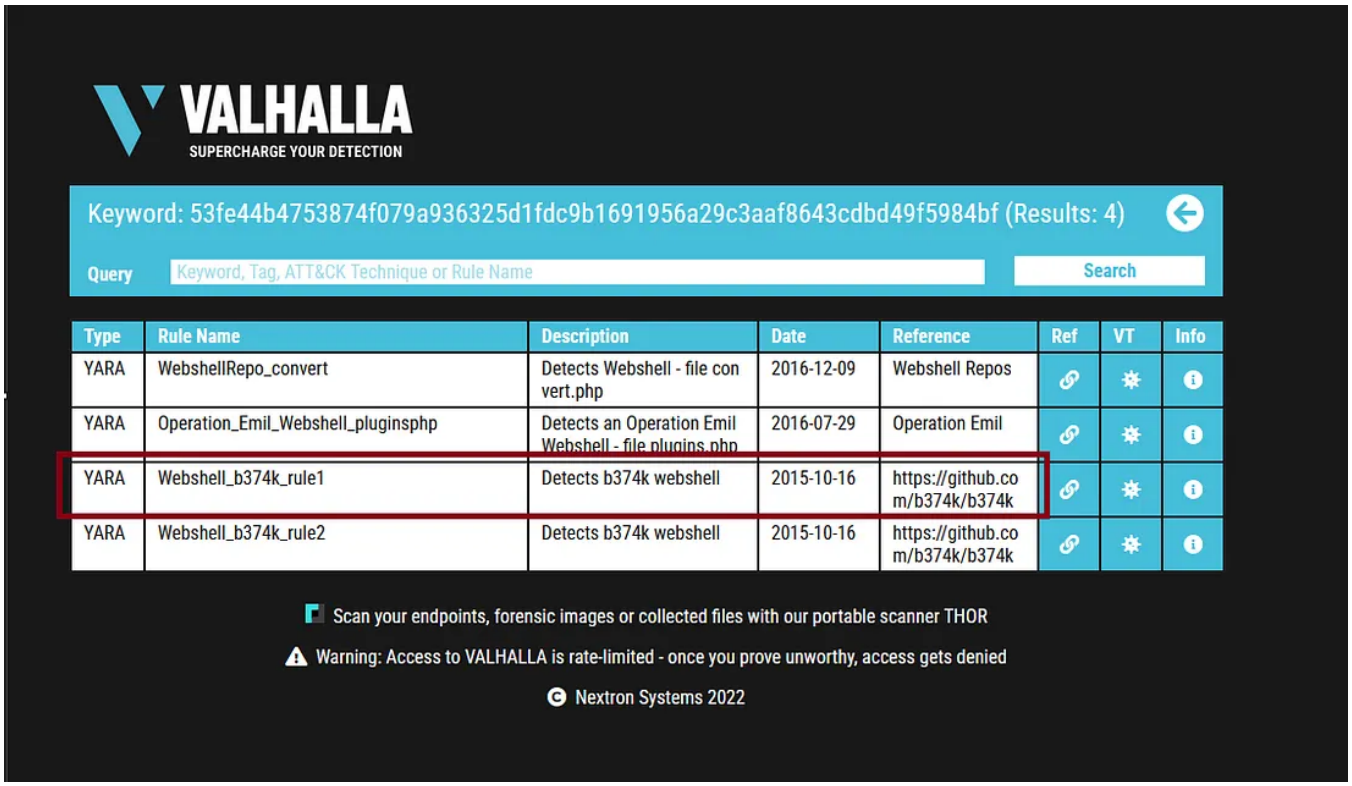
```
cmnatic@thm-yara:~/suspicious-files/file2$ ls
1ndex.php  loki_thm-yara_2025-02-21_14-11-49.log  loki_thm-yara_2025-02-21_14-45-20.log
cmnatic@thm-yara:~/suspicious-files/file2$ sha256sum 1ndex.php
53fe44b4753874f079a936325d1fdc9b1691956a29c3aaf8643cdbd49f5984bf  1ndex.php
cmnatic@thm-yara:~/suspicious-files/file2$
```

```
sha256sum 1ndex.php
```

Go to https://valhalla.nextron-systems.com/ and paste the hash



```
53fe44b4753874f079a936325d1fdc9b1691956a29c3aaf8643cdbd49f5984bf
```



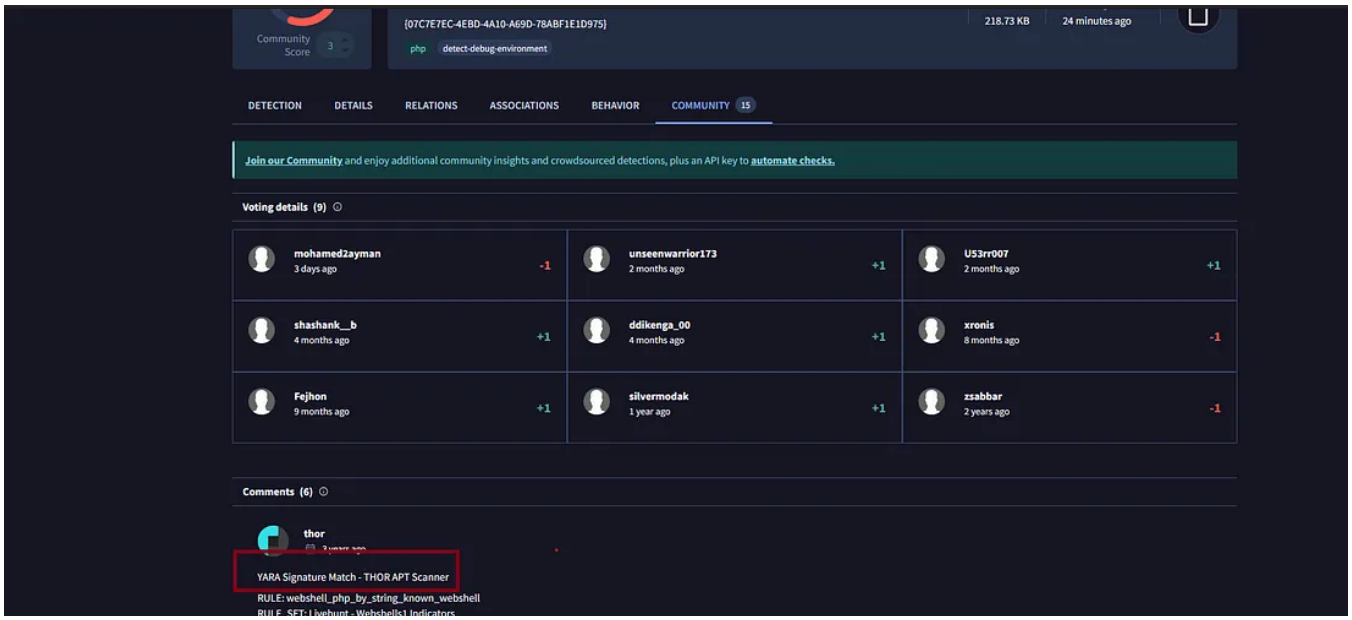
Results

The file was detected 4 YARA rules.

The **first YARA rule** that detected File 2, based on the **oldest date (2015–10–16)** in the results.

Answer: **Webshell_b374k_rule1**

10.3 Examine the information for file 2 from Virus Total (VT). The Yara Signature Match is from what scanner? (Question Hint This information is on the Community tab of the VirusTotal page, and not on the Detection tab)

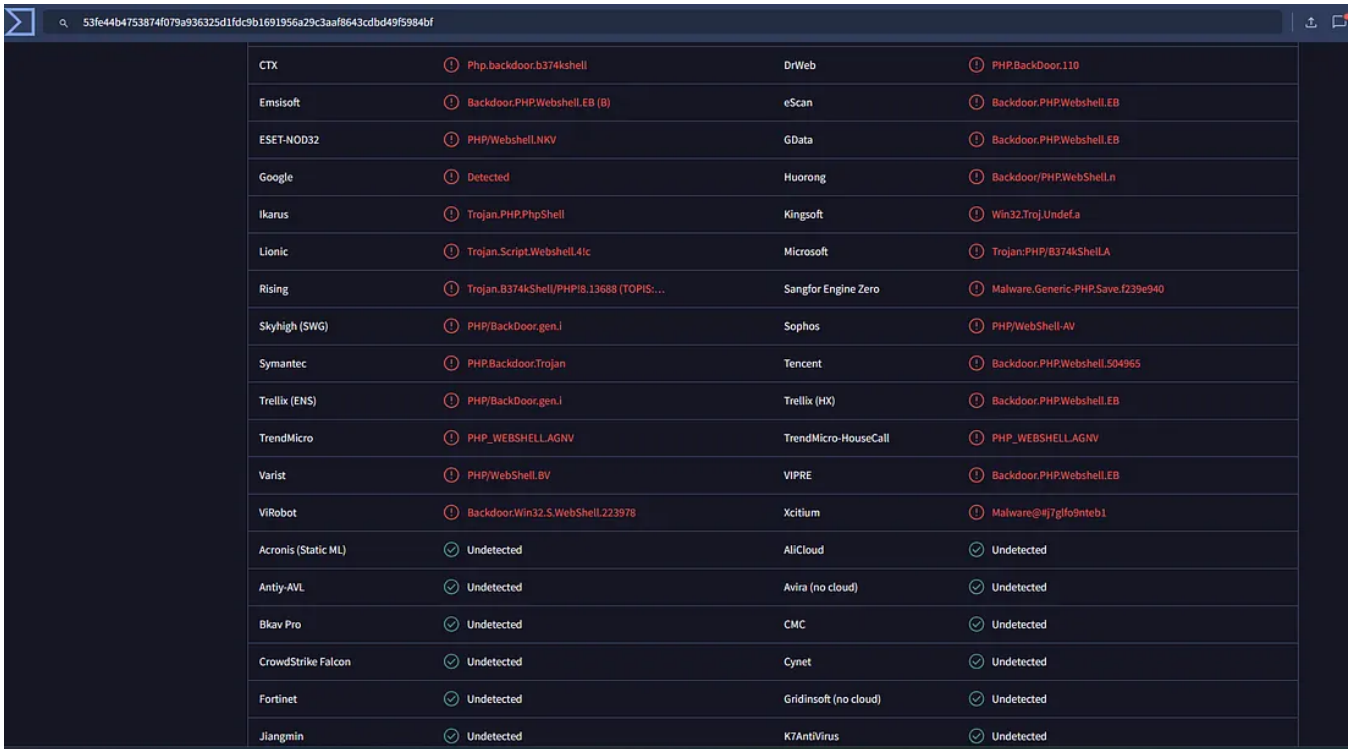


From the **Community tab** in **VirusTotal**, the **YARA Signature Match** is attributed to:

This means the **THOR APT Scanner** identified the file as suspicious based on its YARA rules.

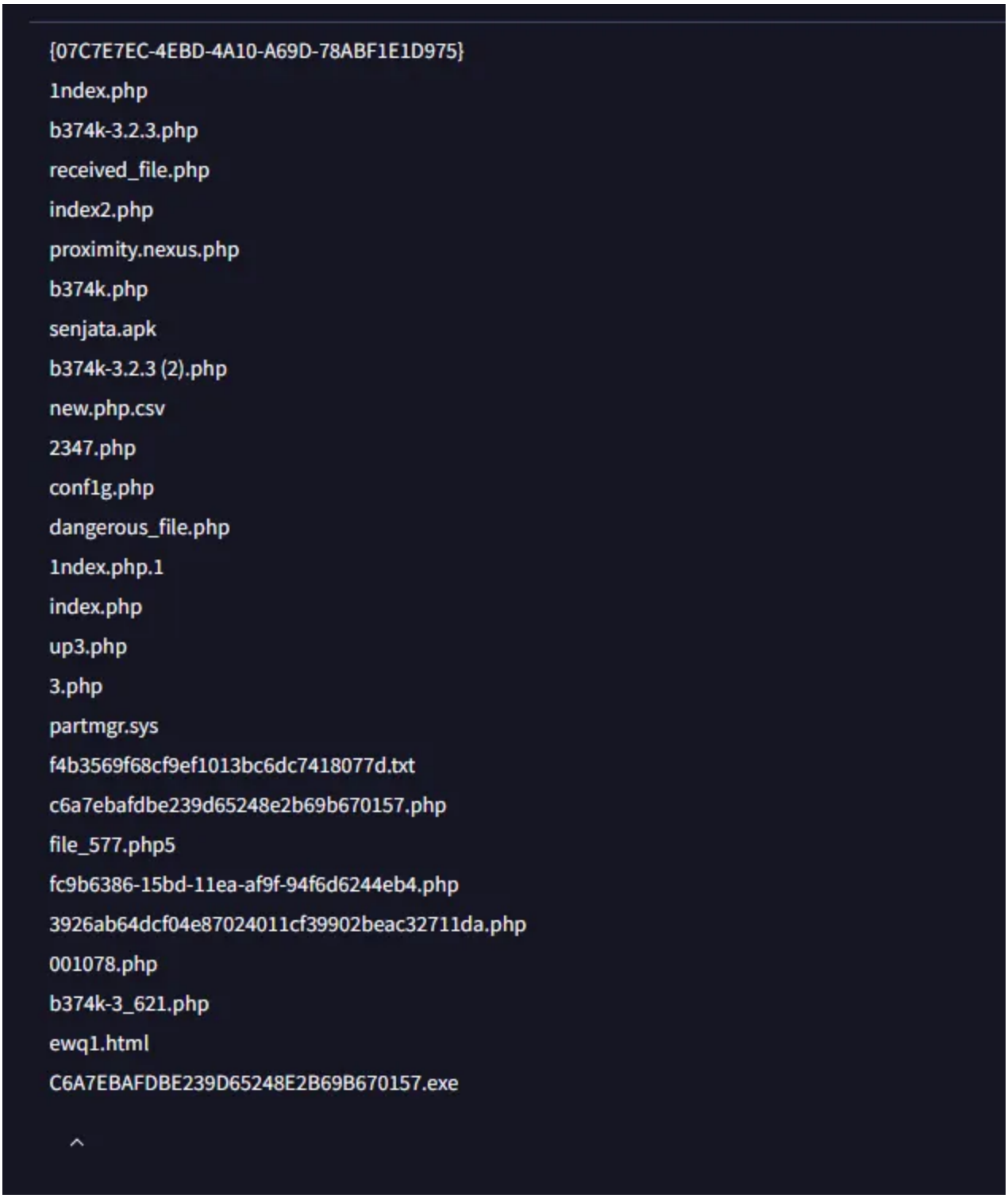
Answer: **THOR APT Scanner**

10.4 Enter the SHA256 hash of file 2 into Virus Total. Did every AV detect this as malicious? (Yay/Nay)



Answer: **Nay**

10.5 Besides .PHP, what other extension is recorded for this file? (Question Hint Look under the “details” tab in Virustotal to find out the extensions for this submission)



The file list shows multiple extensions, including **.PHP, .APK, .TXT, .CSV, .SYS, .HTML**, and **.EXE.**

.APK (Android application) — Android application package
.CSV (Comma-separated values file) — A data storage format

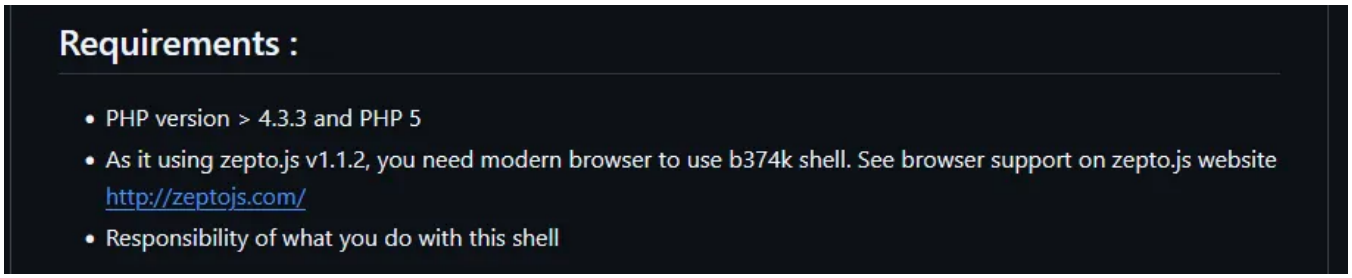
.SYS (System file) — A system file used by Windows
.TXT (Text file) — A plain text file
.HTML (Web page file) — A web page file
.EXE (Executable file) — A Windows executable

.EXE is important because it indicates that the same malware might also be distributed as an **executable** for **Windows systems.**
This means the threat isn't just limited to **web-based attacks** (**PHP files** on web servers) but could also be used for **Windows-based infections.**

Answer: **.EXE**

10.6 What JavaScript library is used by file 2? (Question Hint Go to the Github page and search inside the index.php file)

Go to the Github pag: https://github.com/b374k/b374k



- The **requirements section** clearly states that **zepto.js v1.1.2** is needed for the b374k shell to function properly.
- It also mentions that a **modern browser** is required to support **Zepto.js.**
- The link to the **official Zepto.js website** further confirms its usage in the script.

Also

Inside the **index.php** file

```
26    /* JAVASCRIPT AND CSS FILES START */
27    $zepto_code = packer_read_file($GLOBALS['packer']['base_dir']."zepto.js");
28    $js_main_code = "\n\n".packer_read_file($GLOBALS['packer']['base_dir']."main.js");
29
30    $js_code = "\n\n".packer_read_file($GLOBALS['packer']['base_dir']."sortable.js").$js_main_code;
31    $js_code .= "\n\n".packer_read_file($GLOBALS['packer']['base_dir']."base.js");
32
33
34    if(isset($_COOKIE['packer_theme']))     $theme = $_COOKIE['packer_theme'];
35    else $theme ="default";
36    $css_code = packer_read_file($GLOBALS['packer']['theme_dir'].$theme.".css");
37
38    /* JAVASCRIPT AND CSS FILES END */
39
```

Line 27: The script loads **zepto.js** using:

$zepto_code = packer_read_file($GLOBALS[‘packer’][‘base_dir’].”zepto.js”);

This means that **zepto.js** is being read and used in the script.

Line 34–35: The script dynamically sets a theme using cookies, which suggests that **Zepto.js** is likely involved in UI interactions.

Line 30–31: Other JavaScript files (**sortable.js, base.js, main.js**) are also included, but Zepto.js is explicitly mentioned.

It confirms that **Zepto.js** is being used in **file 2**

Answer: **Zepto**

10.7 Is this Yara rule in the default Yara file Loki uses to detect these type of hack tools? (Yay/Nay) (Question Hint Examine thor-webshell.yar and search for the rule name)

The default Loki rules are stored in **/home/cmnatic/tools/Loki/signature-base/yara/**

We need to search for the rule in Loki’s YARA rule repository

```
cmnatic@thm-yara:~$ ls /home/cmnatic/tools/Loki/signature-base/yara/ | grep "Webshell_b374k_rule1"
cmnatic@thm-yara:~$
```

```
ls /home/cnmatic/tools/Loki/signature-base/yara/ | grep "Webshell_b374k_rule1"
```

Since the command returned **no output**, this means that the rule is **not present** in Loki's default YARA rules.

Answer: **Nay**

Task 11 Conclusion

In this room, we explored Yara, how to use Yara, and manually created basic Yara rules. We also explored various open-source tools to hit the ground running that utilizes Yara rules to detect evil on endpoints.

By going through the room scenario, you should understand the need (*as a blue teamer*) to know how to create Yara rules effectively if we rely on such tools. Commercial products, even though not perfect, will have a much richer Yara ruleset than an open-source product. Both commercial and open-source will allow you to add Yara rules to expand its capabilities further to detect threats.

If it is not clear, the reason why **file 2** was not detected is that the Yara rule was not in the Yara file used by Loki to detect the hack tool (web shell) even though its the hack tool has been around for years and has even been attributed to at least 1 nation-state. The Yara rule is present in the commercial variant of Loki, which is Thor.

There is more that can be done with Yara and Yara rules. We encourage you to explore this tool further at your own leisure.

Answer the questions below

No answer needed.

Follow

Written by IritT

3.7K followers · 6 following

In the world of cybersecurity, the strongest defense is knowledge. Hack the mind, secure the future.

No responses yet

Itsjustme

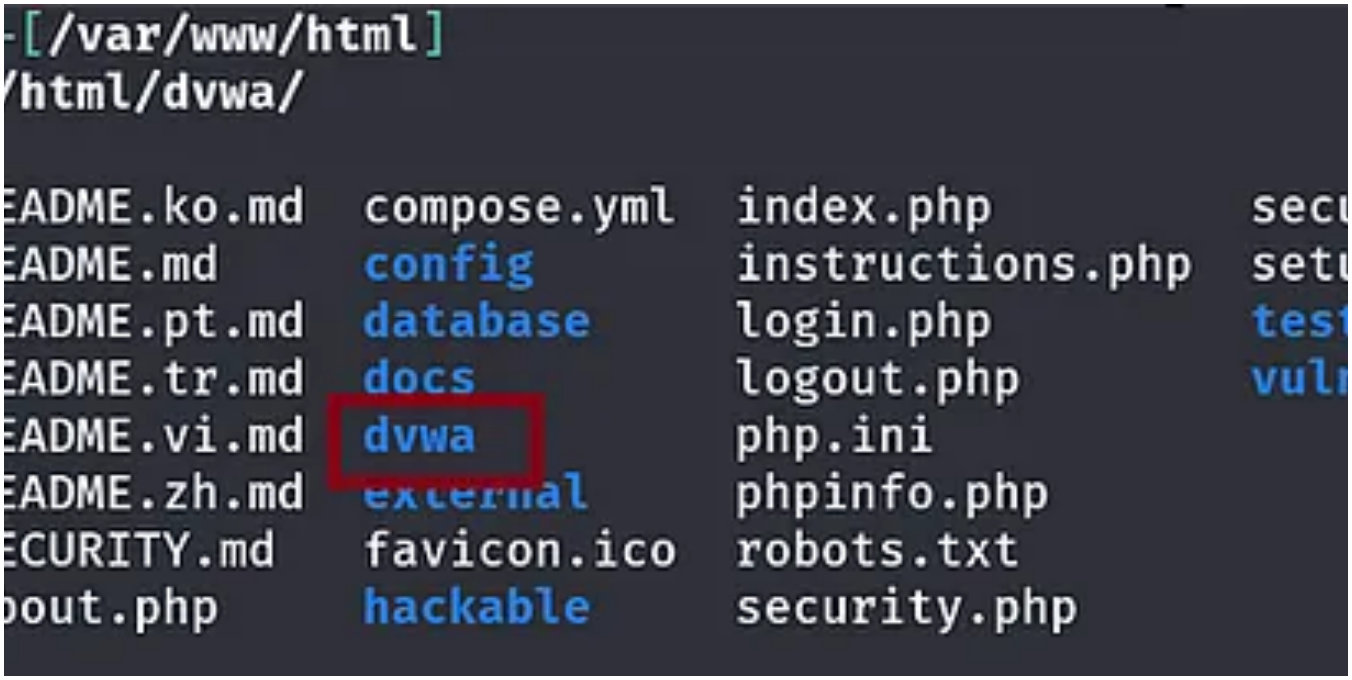
More from IritT

IritT

Wireshark: The Basics — Wireshark— TryHackMe Walkthrough

Learn the basics of Wireshark and how to analyse protocols and PCAPs.

Oct 26, 2024 7 2



IritT

Setting Up DVWA Training Platform on Kali Linux in VMware | A Simple Guide

What is DVWA?

Sep 29, 2024 22 6

IritT

Burp Suite: The Basics — TryHackMe Walkthrough

An introduction to using Burp Suite for web application pentesting.

Sep 19, 2024 2

IritT

OWASP Top 10 — TryHackMe Walkthrough

https://www.youtube.com/watch?v=xC8I9HuvHuI

Oct 9, 2024 15 1

See all from IritT

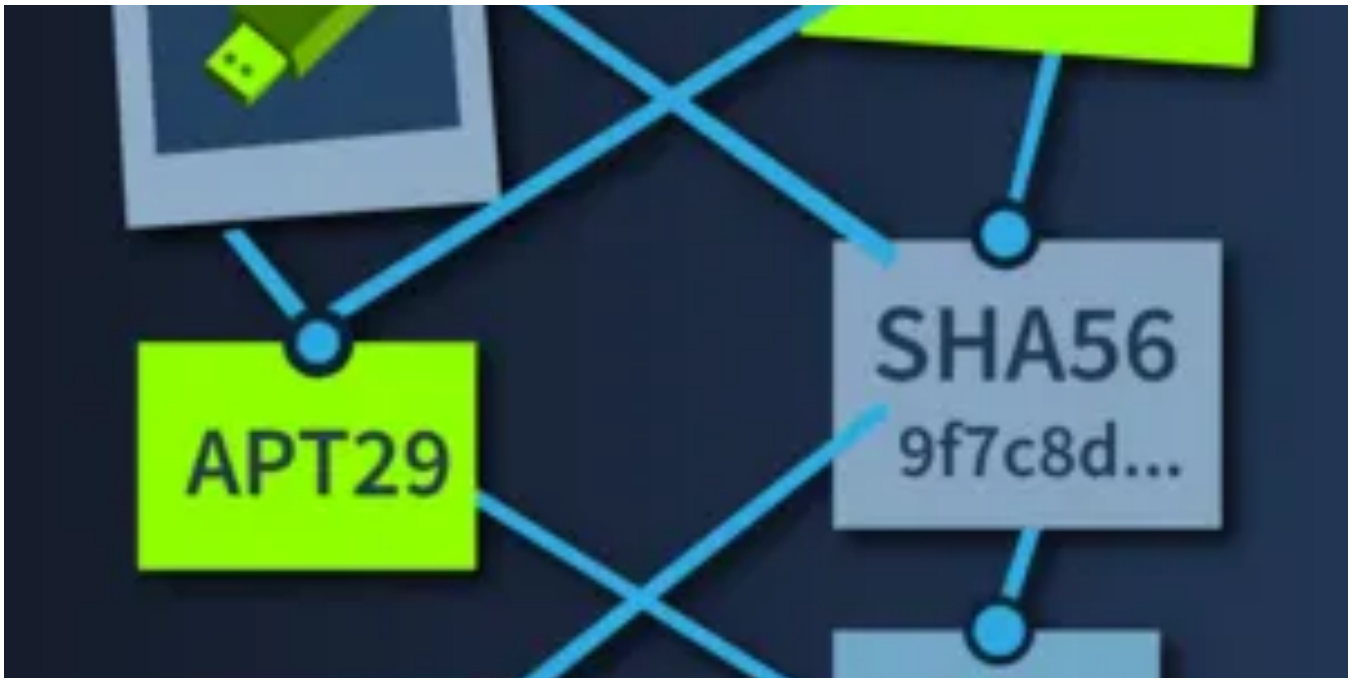
Recommended from Medium

In T3CH by Ansul Kotadia

Web Security Essentials: TryHackMe Answers

Learn how the web works, common website security risks, and protections for a safer internet.

Aug 30 51

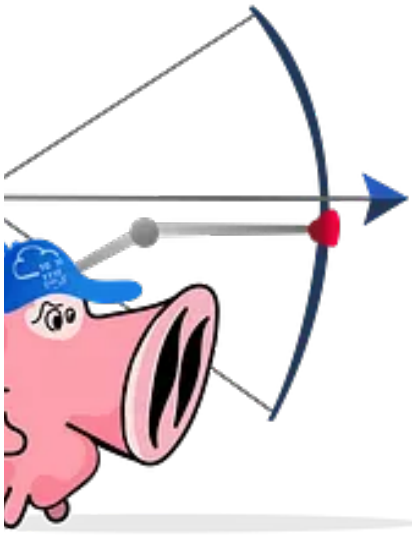


Francesco Pastore

THM - Invite Only

A writeup for “Invite Only” on TryHackMe

3



the Journal of a Cyber Security Enthusiast

SNORT Challenge — The Basics | TryHackMe — Network Security & Traffic Analysis

“The room invites you a challenge to investigate a series of traffic data and stop malicious activity under two different scenarios. Let’s...

Jul 27 2 1

MAGESH

Linux Threat Detection 1 -Tryhackme

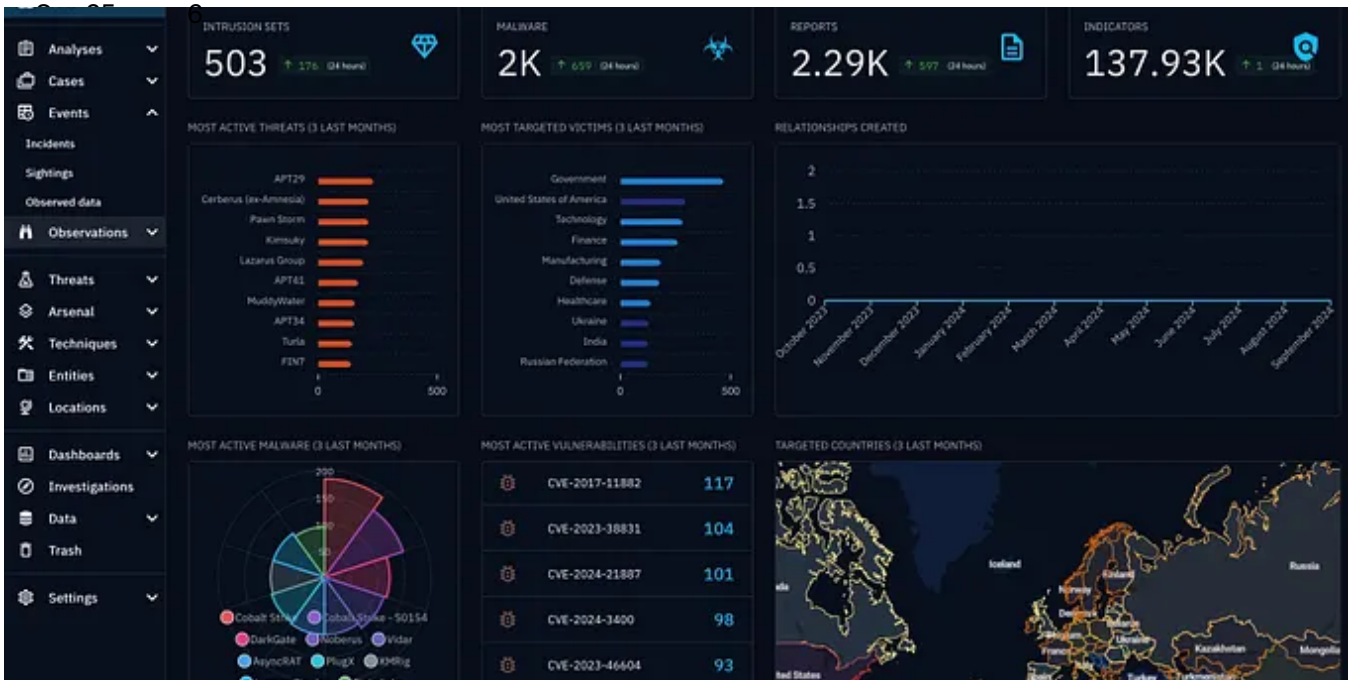
Explore how attackers break into Linux systems and how you can detect this in logs

Sep 26

Sle3pyHead

Linux Threat Detection 1 Walkthrough Notes | TryHackMe

Analyze logs, detect breach, trace processes, and recover compromised systems.



In InfoSec Write-ups by Vito Rallo (CRIMSON7)

TI for fun, or more: a more serious OpenCTI 💡

Build your own CTI by deploying OpenCTI—seriously, and on a budget

Jun 21 25

See more recommendations