Advent of Cyber, Day 21 is LIVE! 🔎🪲 Can you help remove malicious files from a network using Yara?

Grinch Enterprises have been very sneaky this year - using multiple attack vectors (both know and unknown) to wreak havoc across the Best Festival Company. It's 4 days from Christmas and there's still so much work to do! McBlue, the only technical person, has suggested using automation and tooling to detect malicious files across the network. Can you work with him to remove any malicious files from the network?

Learning Objectives:

In this task, we will learn:

* What are Yara rules
* What is the basic structure of Yara rules
* How to write basic Yara rules
* How Yara rules are used

Yara

**Introduction:**

YARA is a multi-platform tool for matching patterns of interest in (malicious) files. It was created by Victor Alvarez from VirusTotal. It is used to perform research on malware families and identify malware with similar patterns. It can help in categorizing malware in different malware families, and can also be used as a detection aid for malware analysis.

Yara rules are very widely used in the industry for the above-mentioned purposes. There are various open-source repositories of Yara rules shared by different organizations and people that can be leveraged by the security community in their fight against malware. [This GitHub](https://github.com/InQuest/awesome-yara)repository contains links to a lot of such open-source repositories.

**Syntax:**

The basic syntax of a Yara rule is as follows:

rule rulename  
  {  
    meta:  
      author = "tryhackme"  
      description = "test rule"  
      created = "11/12/2021 00:00"  
    strings:  
      $textstring = "text"  
      $hexstring = {4D 5A}  
    conditions:  
      $textstring and $hexstring  
  }

We can see that the rule starts with the word rule followed by the name of the rule. This indicates the start of the rule. We discuss the different sections of the rule below. Among these, the metadata section (uses keyword meta ) as discussed at the end, because it is an optional section.

**Strings:**

As the name suggests, this section is all about the strings you want to match in a Yara rule. In this section, we define strings as if we will define variables. A string declaration starts with a $ sign, followed by the name we want to assign to that string. In the above example, we have defined the two strings as $textstring and $hexstring.

As the names of the strings signify, strings can be text strings or hex strings. Text strings are strings found in the legible text portion of a file, however, hex strings are raw sequences of bytes in a file. To define text strings, we use double quotes, and to define hex strings, we use curly brackets. An implementation of this can be seen in the example rule above. Text strings can also use regular expressions or regex, for more complex pattern matching.

**Conditions:**

The condition section defines the conditions that the rule writer wants to meet in order for the rule to hit on a file. Conditions are boolean expressions, and they use the strings defined in the strings section as variables. For example, in the example above, we use the and operator between the two strings. This means that the rule will hit if it identifies a 4D 5A hex string anywhere in the file as well as the text string text. There is no other condition in this rule. The most commonly used operators are and, or and not. The and operator returns true when all the conditions are true. The or operator returns true when any one of the conditions is true. And the not returns true when the given condition is false. In the case of strings, andwill be true when the file contains all the strings that are part of that condition, or will be true when the file contains any of the strings that are part of that condition, and not will be true when the string mentioned in the condition is not a part of the file. The following table illustrates when each of these operators will be true:

|  |  |
| --- | --- |
| **Operator** | **Condition** |
| and | All statements are true |
| or | Any one statement is true |
| not | The statement is false |

Conditions can be much more complex than the ones we mentioned above, but for the sake of brevity, we will not go into further details in this task.

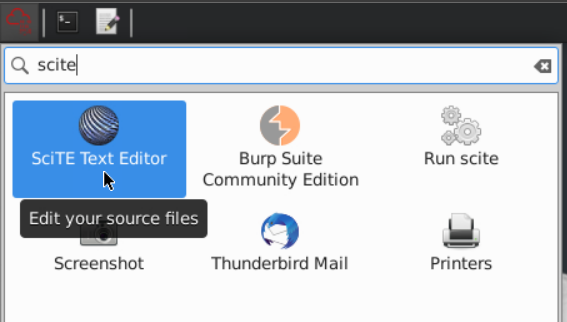
**Metadata:**

Metadata is an optional, but important section in the rules. It starts with the keyword meta. It can be used to add additional information about the rule to help the analyst in their analysis. Generally, it contains arbitrarily defined identifiers, and their values, which are universally understood. For example, in our example rule above, we used the identifiers of author, description, and created. These identifiers are easy to understand in the type of information they are providing. Adding metadata to rules is especially important when contributing to the community, as they provide important contextual information and attribution for the rule.

**Writing your own rules:**

With the basic introduction out of the way, we can try writing some of our own rules. So far, we have understood that the rule basically consists of a strings definition section and a conditions section. The strings definition section will define the strings that we want to use in the rule, and the conditions section will define the conditions that will apply to those strings. Lastly, we can add any additional information in the metadata section.

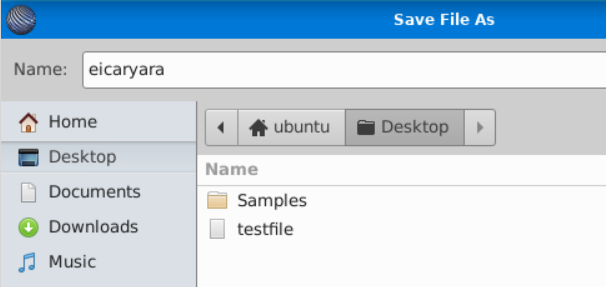
To start writing a rule, start the machine attached to the task. Once the machine has started, click on the tryhackme logo on the top-left corner. In the search bar, write scite and select the relevant icon to open the SciTE text editor. You can open any other text editor of your choice as well.



Now using the knowledge gained from this task, let’s start writing the rule. We will write a rule that matches some strings from the EICAR antivirus test file we learned about in the malware analysis task. This file is placed on the Desktop on the machine with the name testfile. The following screenshot shows our demo rule for detecting a few strings found in the EICAR test file.

rule eicaryara   {  
    meta:  
      author="tryhackme"  
      description="eicar string"  
    strings:  
      $a="X5O"  
      $b="EICAR"  
      $c="ANTIVIRUS"  
      $d="TEST"  
    condition:  
      $a and $b and $c and $d  
  }

Next, we save this file and give it a name of our choice. We have saved this file on the Desktop for easier access when running the Yara rule on our testfile.



**Running the Yara rules:**

So now that we have written a rule, let’s learn how to run it.

The syntax to run a Yara rule can be simply stated as follows:

yara [options] rule\_file [target]

In our example above, our rule\_file will be the file we saved as eicaryara, and the target will be our testfile. Therefore, our command for running the rule will look like this:

yara [options] eicaryara testfile

Giving the options here is optional. If we run this command as it is, it will return us with the rule name and the file name if the rule is hit. If the rule is not hit, it will not return anything.

When the rule is a hit the output will look like this:

Terminal

user@TryHackMe**$** yara eicaryara testfile

eicaryara testfile

user@TryHackMe$

When the rule is a miss, the output will look like this:

Terminal

user@TryHackMe**$** yara eicaryara testfile

user@TryHackMe$

We can use the -m option for Yara to return the metadata when the rule is hit. The output, in that case, will look like this:

Terminal

user@TryHackMe**$** yara -m eicaryara testfile

eicaryara [author="tryhackme",description="eicar string"] testfile

user@TryHackMe$

Similarly, we can use the -s option to print the strings that matched the file. The output in that scenario will be as follows:

Terminal

user@TryHackMe**$** yara -s eicaryara testfile

eicaryara testfile

0x1c:$b: EICAR

0x2b:$c: ANTIVIRUS

0x35:$d: TEST

user@TryHackMe$

The output here shows the rule that matched, the strings that matched, their identifiers, and their location in the file. Notice that the screenshot contains 3 of the 4 strings we wrote in the rule. This is because we modified the rule before running this command such that one of the strings doesn’t appear in the file. Hence we only see the strings that appear in the test file, and we don’t see the ones which did not match.

**Additional resources:**

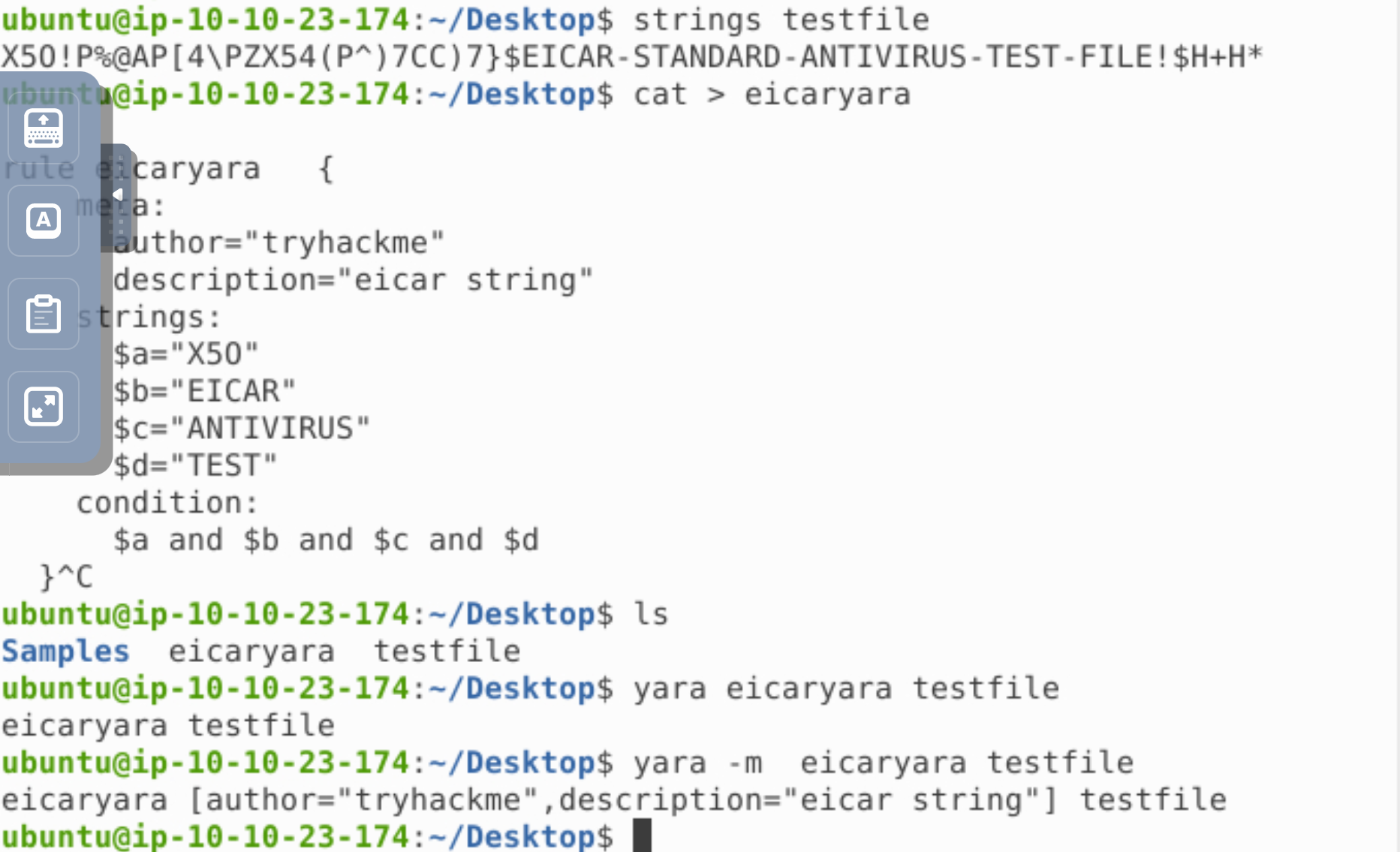
For the sake of brevity, we just touched upon the basics of Yara rules in this task. Yara rules are capable of doing a lot of more complex pattern matching than what we touched here. If you are interested in learning more about Yara rules, you can check the [Yara room on TryHackMe](https://tryhackme.com/jr/yara). You can also check out the [Yara documentation](https://yara.readthedocs.io/en/stable/writingrules.html) or take a look at several [community written rules](https://github.com/Yara-Rules/rules) to get an idea of what type of Yara rules are written by the community.

For now, let’s see if we can answer the following questions to evaluate what we have learned so far.

Deploy the machine attached to the task. It will be visible in the split-screen view when ready. If you don't see the machine in your browser, click the "Show Split View" button.

***Answer the questions below***

We changed the text in the string $a as shown in the *eicaryara* rule we wrote, from X5O to X50, that is, we replaced the letter O with the number 0. The condition for the Yara rule is ***$a and $b and $c and $d***. If we are to only make a change to the first boolean operator in this condition, what boolean operator shall we replace the 'and' with, in order for the rule to still hit the file?

What option is used in the Yara command in order to list down the metadata of the rules that are a hit to a file?



 Submit

What section contains information about the author of the Yara rule?

Tricking on just as section meta is taken as metadata ! lol ! I got confuse couple of time

What option is used to print only rules that did not hit?



 Submit

 Hint

Change the Yara rule value for the $a string to X50. Rerun the command, but this time with the -c option. What is the result?

0 as no matches , -c show the only matches one