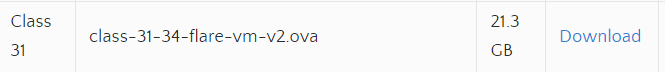
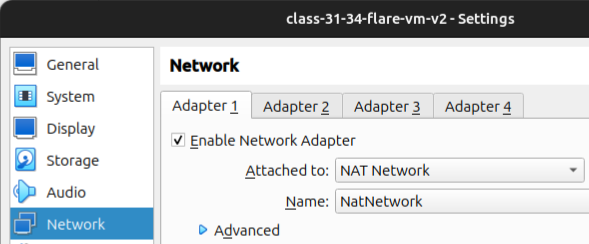
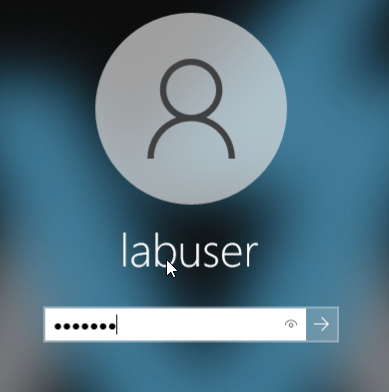
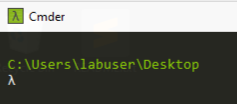
Hélio Ferreira 23/07/2024

# **Lab: Threat Hunting with YARA**

### **Part 1: Staging**

* Download and import the Flare VM OVA.
* Set the network adapter to NAT Network.
* Login as **labuser / labuser**.

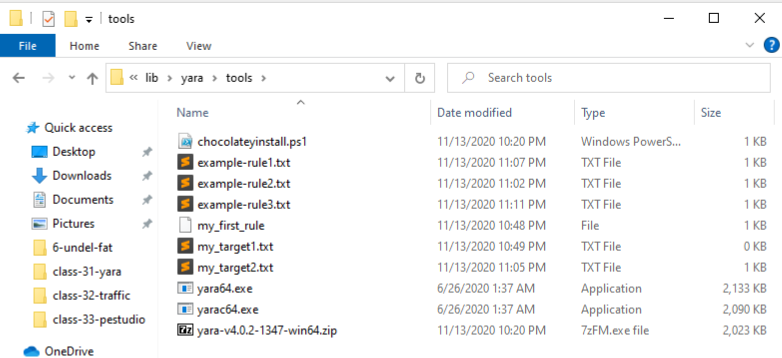
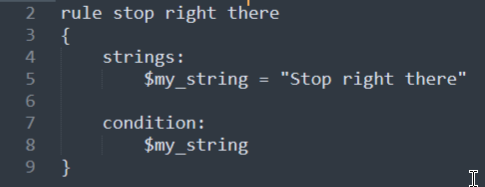
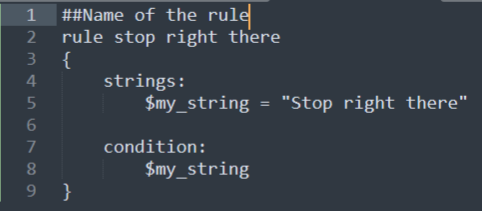
For this lab, try using the **Cmder** console emulator. Similar to GIT Bash, this lets you navigate through Windows using Linux commands.



### **Part 2: My First YARA Rule**

YARA rules can be customized to the needs of your detection systems. Let’s write a simple detection rule that looks for a specific string in a text file, then execute the rule by directly calling YARA from the command line.

*TIP: To succeed with YARA rules, familiarize yourself with YARA rule syntax. What do the components of a rule file mean when executed? Use the included example rules to help you learn this.*

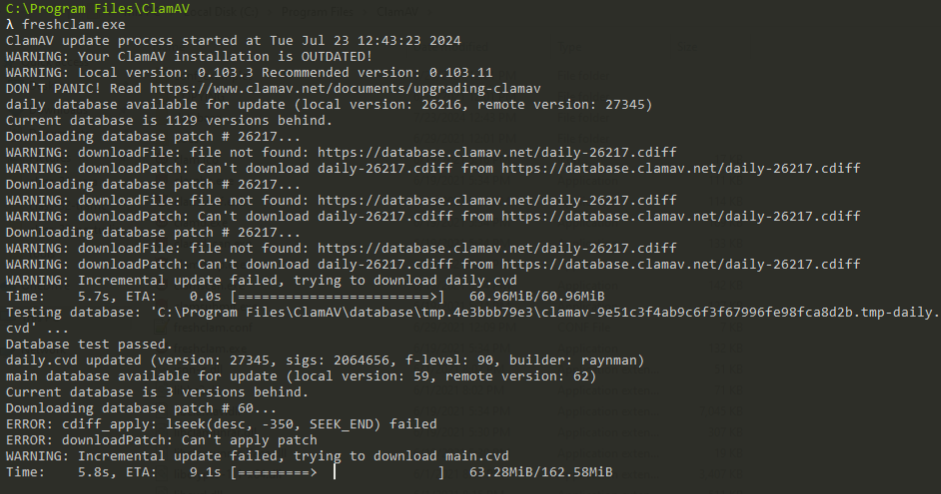
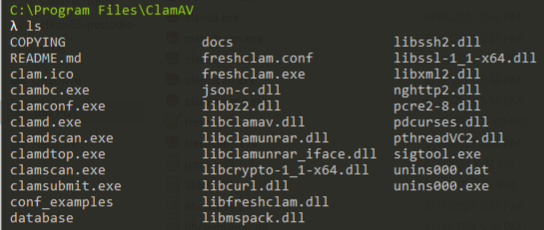
* Access the “yara tools” folder on the desktop. This shortcut will take you to the directory where **yara64.exe** is installed. You’ll need to call **yara64.exe** from command line to use it.
* Create a text file containing a string of your choosing.  
    
    
  The string is “Stop right there”
* Create a YARA rule that tests whether the target file contains the string of your choosing.
* Include comments in your YARA rule.
* Execute the YARA rule against the target file.
* Include a screenshot of the YARA rule and its output.
* Explain whether the rule tested positive or negative. How can you tell?  
    
  I can tell it worked and tested positive because when I asked cmder to find if the document teststring.txt identify any string containing “Stop right there”, it gave me the same output matching the string I mentioned earlier.
* Create a file that does not contain your string. Run your rule against it to generate a negative. Note the difference in output.  
    
  In this case, there was no output, and it means that yara didn’t match the rule with the file.

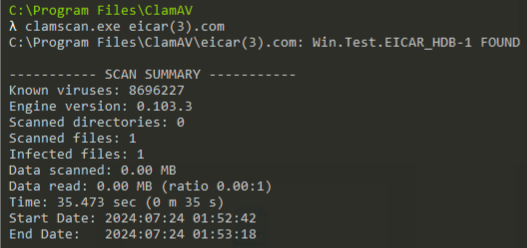
### **Part 3: Customizing ClamAV with YARA Rules**

ClamAV is an open source antivirus that supports YARA rules. Let’s test this out by having ClamAV scan a file using our custom YARA rule.

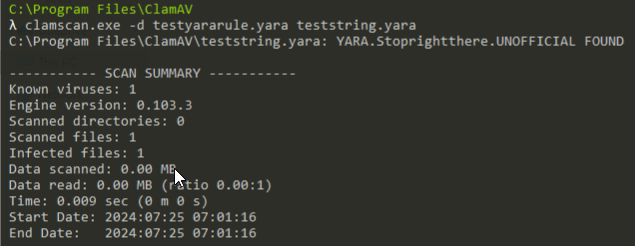
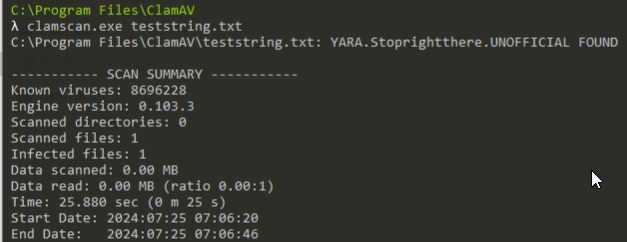
*TIP: Take it slow in this part of the lab and review the provided resources. ClamAV is open source; that means there’s more configuration involved than a commercial software application would require. Remember what configuring Snort was like?*

* Run Fresh Clam to pull the latest virus signatures database into ClamAV’s database directory. You may need to edit the configuration files in ClamAV directory to facilitate the process.



* Practice running ClamAV to perform a scan against the **eicar.com** file in the ClamAV directory.
* Include a screenshot of the positive detection.

Now that we’ve got ClamAV up and running like a normal antivirus solution, let’s get to the fun part: Customizing ClamAV’s detection algorithm by writing our own YARA rule and adding it to ClamAV’s logic.

* Use ClamAV to scan your target text file from Part 2 using the rule you created in Part 2. Do this by instructing ClamAV to use your YARA rule instead of its signature database. Include a screenshot and explanation of how you achieved this.  
  
* Next, instruct ClamAV to scan the target text file from Part 2 using both the rule we created and ClamAV’s signature database. Include a screenshot and explanation of how you achieved this.

PE files contain the string “MZ”. Let’s create a rule that determines whether the target is a PE file.

* Create a new YARA rule named “is\_pe\_file\_clamav\_1.yara”.
* Write the rule to look for the string “MZ” in its target.
* Test the rule using ClamAV against a PE file and a non-PE file. Include screenshots and discussion.

### **Part 4: Reporting**

That was just the beginning of what YARA rules can detect. There’s plenty more you can do with the powerful YARA engine and software that can use it. The one-two combination of YARA and ClamAV means you now have access to an extensible anti-malware toolkit for any environment.

*“Extensibility is a measurement of a piece of technology’s capacity to append additional elements and features to its existing structure. A software program, for example, is considered extensible when its operations may be augmented with add-ons and plugins. Extensible programming languages have the ability to define new features and introduce new functionality within them.” -Techopedia*

Answer the below prompts to the best of your ability:

* What else can YARA detect?  
    
  **Data Patterns:**

- Strings: Exact sequences of characters.

-Regular expressions: Flexible patterns of characters.

-Hexadecimal and binary data: Specific byte sequences.

-Entropy calculations: Measures randomness of data (useful for identifying compressed, encrypted, or malicious content).

**File Characteristics:**

-File metadata: Information like size, creation time, file type, etc.

-File headers and structures: Identifies specific file formats and structures.

-Imports and exports: Analyzes functions imported or exported by a file.

**Code Analysis:**

-Code patterns: Recognizes specific code sequences, function calls, or API usage.

-Opcode patterns: Detects specific instruction sequences in assembly code.

-Control flow graphs: Analyzes the structure of the code.

**Additional Features:**

-Custom modules: Extends YARA's capabilities for specific analysis tasks.

-Rule combination and logic: Creates complex detection logic.

* What if you don’t have a single string to search for in the target?  
    
  **Key Alternatives to String-Based Detection**
* **File Structure and Characteristics:**
  + **File headers:** Identifying specific file formats (PE, ELF, ZIP, etc.) and their structures.
  + **File metadata:** Analyzing attributes like size, creation time, modification time, file type, etc.
  + **Imports and exports:** Examining imported and exported functions to identify potential malicious behavior.
* **Data Patterns and Analysis:**
  + **Hexadecimal and binary data:** Detecting specific byte sequences, often used for identifying code patterns or structures.
  + **Regular expressions:** Matching patterns of characters, providing flexibility in identifying data structures without exact string matches.
  + **Entropy calculations:** Measuring the randomness of data, helpful in detecting compressed, encrypted, or obfuscated content.
* **Code Analysis:**
  + **Opcode patterns:** Detecting specific instruction sequences in assembly code.
  + **Control flow graphs:** Analyzing the structure of the code to identify potential malicious behavior.
* **Behavioral Indicators:**
  + **File modifications:** Detecting files that create, modify, or delete specific files or registry keys.
  + **Network activity:** Identifying files that establish network connections to suspicious IP addresses or ports.
* How would you explain ClamAV’s “file decomposition” feature?  
    
  By decomposing files, ClamAV can:
* **Unpack packed files:** Extract the original code from packed executables.
* **Decompress compressed files:** Uncompress archives to analyze their contents.
* **Decrypt encrypted files:** If possible, decrypt files to reveal their original data.
* **Extract embedded files:** Identify and extract files embedded within other files (e.g., documents with embedded scripts).

This decomposition process allows ClamAV to examine the underlying content of files more effectively, increasing the chances of detecting malicious code. It's particularly useful for identifying complex malware variants that rely on obfuscation techniques.

In essence, ClamAV's file decomposition acts like peeling layers from an onion, revealing the core content for deeper analysis.