

LAB TITLE: INTRODUCTION TO YARA 101 LAB

STUDENT NAME: BRIAN NJIRU.

STUDENT ID: 2025/ACTI/6177.

COURSE NAME: ADVANCED THREAT INTELLIGENCE.

INSTRUCTOR NAME: AMINU IDRIS.

DATE OF SUBMISSION: 27TH FEBRUARY 2025.

VERSION: 1.0

Contents

1. INTRODUCTION TO YARA	4
1.1 YARA rules.	4
1.2 Installation and setup.....	4
1.3 Basic YARA rule structure	5
1.4 YARA rules command structure.	6
2. YARA RULES CREATED.....	7
2.1 Detecting a string in a file.	7
2.2 Detecting Malware via Strings and Byte Patterns.....	8
2.3 Detecting Malware with Regular Expressions.	9
2.4 Detecting Malware while avoiding false positives.	10
2.5 Detecting Malware based on its hash.....	11
2.6 Detecting malware using a comprehensive rule.....	12
3. TESTING THE RULES.	14
3.1 Testing detect_string_rule.yara.....	14
3.2 Testing evil_software_rule.yara	16
3.3 Testing regex_rule.yara	18
3.4 Testing optimized_rule.yara	21
3.5 Testing hash_rule.yara	24
4. CHALLENGES AND LEARNINGS.....	27
4.1 Unsolved issue using Regular Expressions in YARA.....	27
5. CONCLUSION	28

Figure 1: Installing YARA in kali Linux virtual machine.....	4
Figure 2: YARA rule basic syntax.	5
Figure 3: YARA rule basic syntax - strings section and condition sections.....	5
Figure 4: YARA rules basic syntax - the meta section.....	6
Figure 5:YARA rule command structure.....	6
Figure 6:YARA rules command structure - available options.....	6
Figure 7: YARA rule to detect the string "malicious_string" in a file.	7
Figure 8: YARA rules to detect malware via strings and byte patterns.	8
Figure 9:YARA rule to detect malware version with regular expressions.	9
Figure 10:YARA rule to detect malware while avoiding false positives.	10
Figure 11:YARA rule to identify a file based on its hash.	11
Figure 12: YARA rule to detect piece of malware comprehensively.....	12
Figure 13: Python script to check for the date of modification.	13
Figure 14: malware_file.exe.....	14
Figure 15: benign_file.txt.....	15
Figure 16: Running the YARA rule detect_string_rule.yara.....	15
Figure 17: evil_sample.exe.....	16
Figure 18: benign_file.txt.....	17
Figure 19:Running the YARA rule, evil_software_rule.yara	17
Figure 20: malware_v1.exe	18
Figure 21: malware_v2.exe	19
Figure 22: random_file.txt	19
Figure 23: Running regex_rule.yara	20
Figure 24: benign_file.txt	21
Figure 25: malware_1.exe.....	22
Figure 26: malware_2.exe.....	22
Figure 27: Running optimized_rule.yara	23
Figure 28: known_malware.exe.....	24
Figure 29: benign_file.txt.....	25
Figure 30: Running hash_rule.yara	25

1. INTRODUCTION TO YARA

1.1 YARA rules.

YARA is a tool aimed at helping malware researchers to identify and classify malware samples. With YARA you can create descriptions of malware families, among other descriptions, based on textual or binary patterns.

1.2 Installation and setup

YARA is a multiplatform program running on Windows, Linux and Mac OS X. To install YARA, I had to follow the steps outlined in the YARA documentation depending on the operating system. Personally, I used a Kali Linux virtual machine so I followed the necessary steps to install YARA on my system.

I ran the command `sudo apt install YARA.`

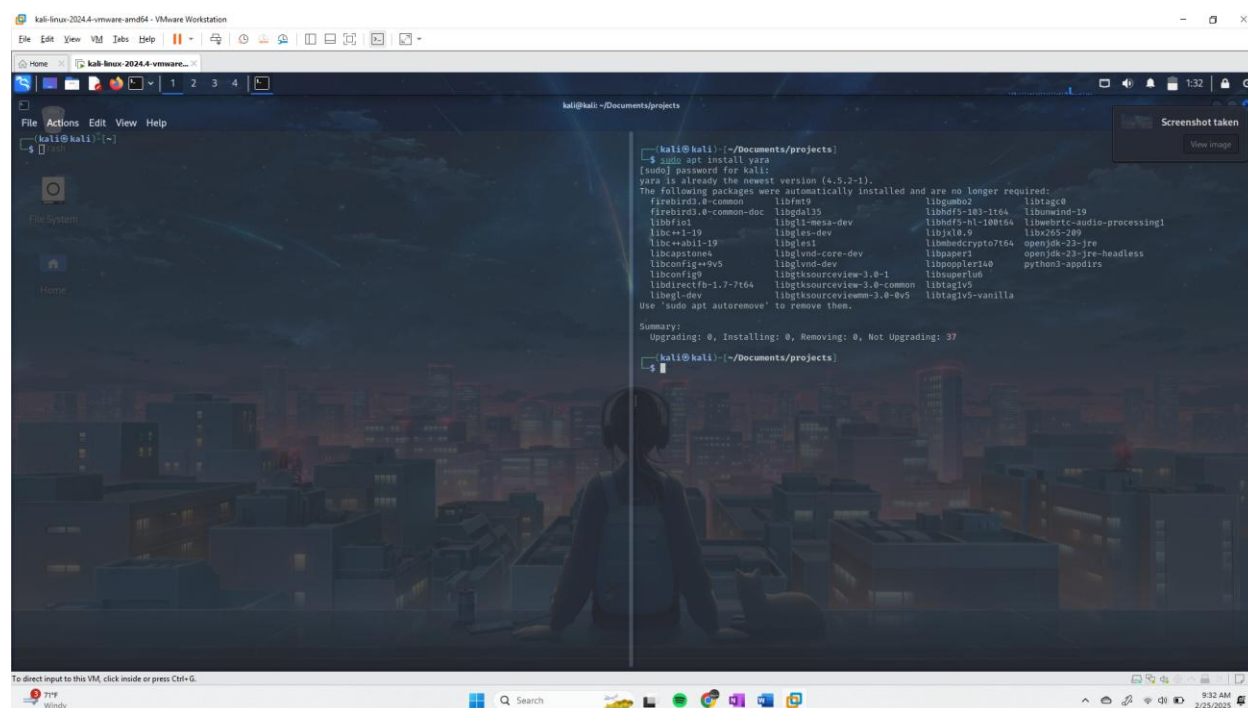


Figure 1: Installing YARA in kali Linux virtual machine

1.3 Basic YARA rule structure

YARA rules have a syntax that resembles the C language.

```
rule dummy
{
    condition:
        false
}
```

Figure 2: YARA rule basic syntax.

Each rule in YARA starts with the keyword **rule** followed by a rule identifier. Identifiers must follow the same lexical conventions of the C programming language; they can contain any alphanumeric character and the underscore character, but the first character cannot be a digit. Rule identifiers are case sensitive and cannot exceed 128 characters.

Rules are generally composed of two sections: strings definitions and conditions. The strings definition section can be omitted if the rule doesn't rely on any string, but the condition section is always required. Each string has an identifier consisting of a \$ character followed by a sequence of alphanumeric characters and underscores, these identifiers can be used in the condition section to refer to the corresponding string.

```
rule ExampleRule
{
    strings:
        $my_text_string = "text here"
        $my_hex_string = { E2 34 A1 C8 23 FB }

    condition:
        $my_text_string or $my_hex_string
}
```

Figure 3: YARA rule basic syntax - strings section and condition sections

YARA rules may include a meta section which includes metadata about the rule and the author. The meta section usually includes: description, author and date. The description tag describes what the rule does. The author tag includes the name of the person who wrote the YARA rule. The date tag contains the date on which the YARA rule was written.

```
rule SampleRule
{
  meta:
    description = "Detects 'malicious_string' or a byte pattern"
    author = "Instructor"
    date = "2025-01-23"

  strings:
    $text_string = "malicious_string"
    $byte_pattern = { E8 34 12 56 }

  condition:
    $text_string or $byte_pattern
}
```

Figure 4: YARA rules basic syntax - the meta section.

1.4 YARA rules command structure.

YARA rules are executed from the command line. To run written YARA rules, two things are needed: a file with the rules you want to use and the target to be scanned. The target can be a file, a folder or a process.

```
yara [OPTIONS] RULES_FILE TARGET
```

Figure 5: YARA rule command structure.

Some options available include:

```
-C --compiled-rules
```

RULES_FILE contains rules already compiled with yarac.

```
-c --count
```

Print only number of matches.

```
-d <identifier>=<value> --define=identifier=value
```

Define **external** variable. This option can be used multiple times.

```
-q --disable-console-logs
```

Disable printing console log messages.

Figure 6: YARA rules command structure - available options

2. YARA RULES CREATED.

The objective of the lab was to equip us with a good understanding of YARA rule creation, file analysis and how to use YARA for malware detection. The lab required us to create and run a number of YARA rules in order to successfully identify patterns in sample files. These rules are described below.

2.1 Detecting a string in a file.

I was tasked with creating a YARA rule to detect the string “**malicious_string**” in any file. I approached the task as documented in the YARA rules documentation by specifying a “**text_string**” variable that contained the string that I had to detect; “**malicious_string**”. I included the relevant information in the meta section and then finally specified that the string must be contained in the file for accurate detection in the conditions section. I saved the YARA rule as “**detect_string_rule.yara**”.

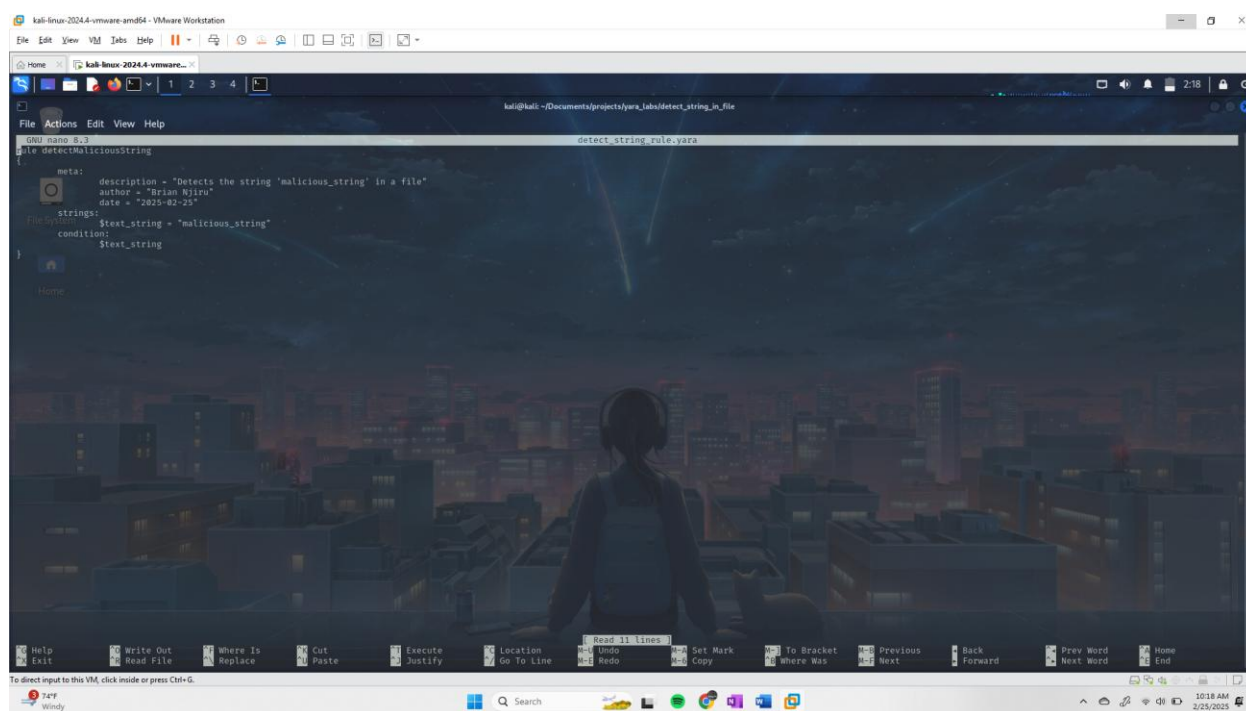


Figure 7: YARA rule to detect the string "malicious_string" in a file.

2.2 Detecting Malware via Strings and Byte Patterns

I was tasked with detecting a piece of malware called **EvilSoftware**. The malware contained the string “**EvilSoftware**”, and its byte signature was **{E8 34 12 56}**. I wrote a YARA rule with two variables in the strings section: **\$evil_string** and **\$evil_bytes**. These two variables contained the respective string and byte signature that needed to be detected. I then specified in the condition section that either the string or byte signature had to be present for the YARA rule to detect the malware. I saved the yara rule as “**evil_software_rule.yara**”

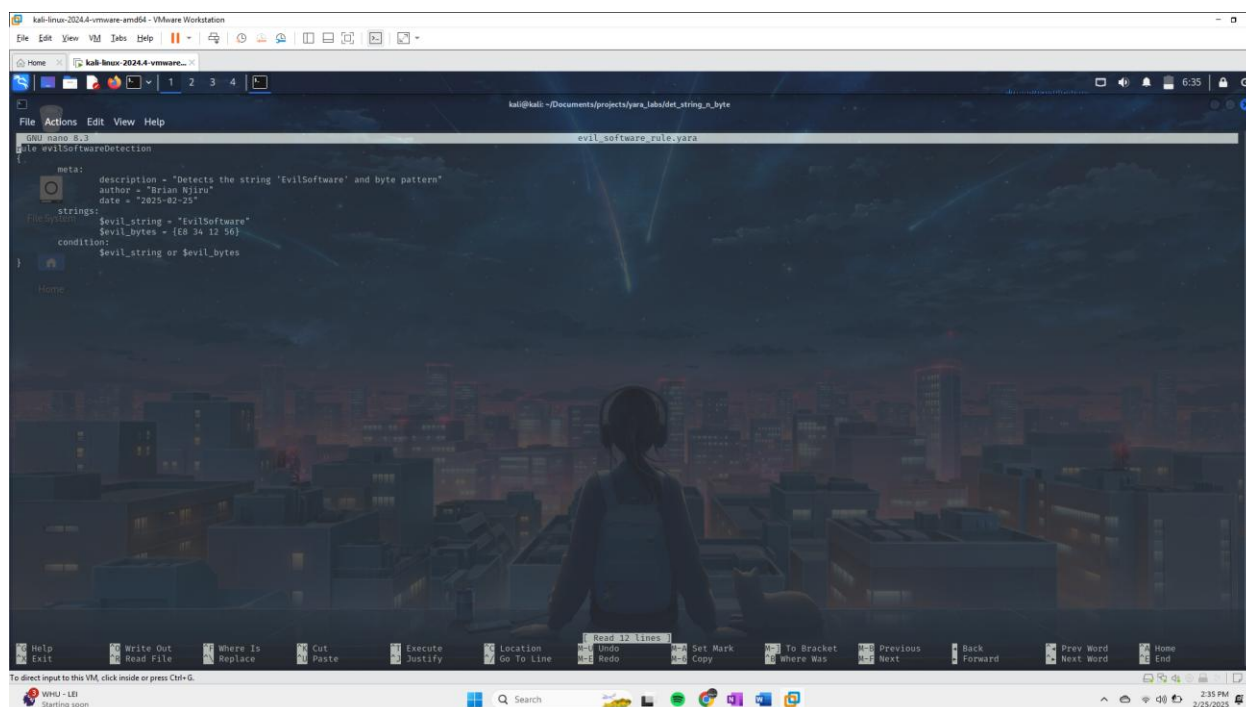


Figure 8: YARA rules to detect malware via strings and byte patterns.

2.3 Detecting Malware with Regular Expressions.

I was tasked with detecting various versions of a malware family in a folder. The filenames of the malware files from the malware family had the following pattern: “**malware_v1**”, “**malware_v2**” and so on. I wrote a YARA rule that detected files whose names matched the pattern “**malware_vX**” where x is any number. I wrote a variable called **\$filename_pattern** which the regular expression **/malware_v\d+/. I saved the rule as **regex_rule.yara**.**

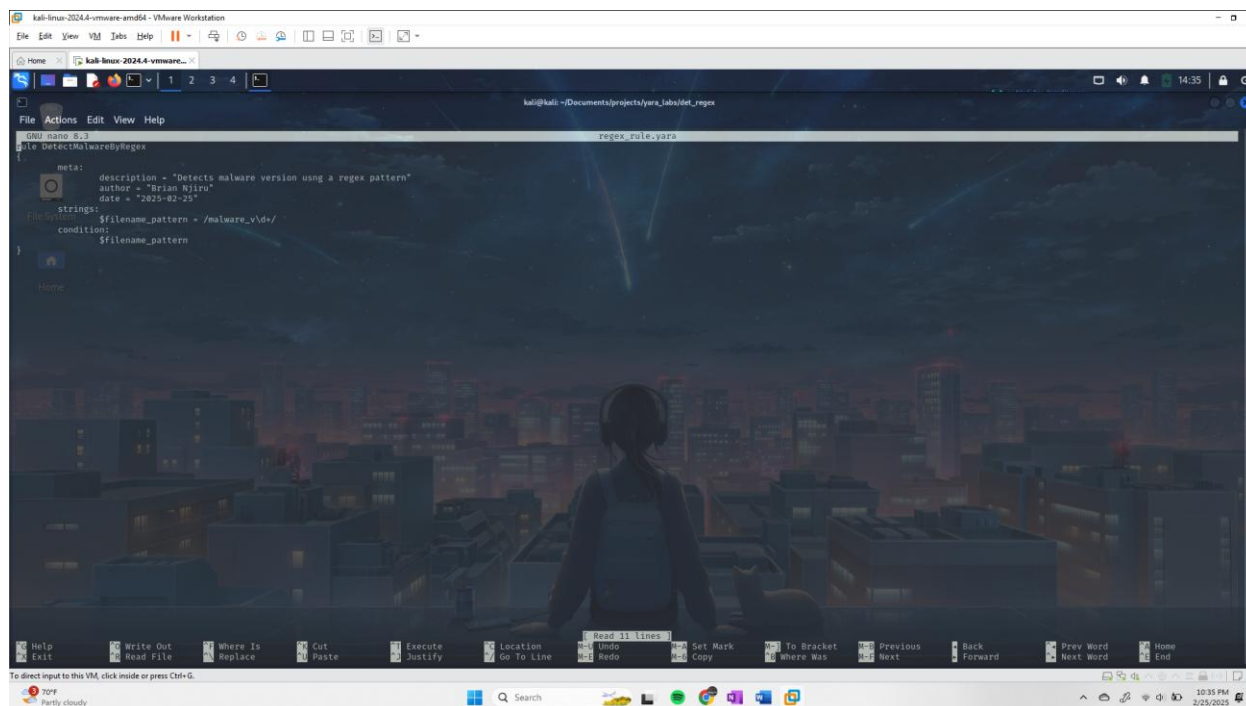


Figure 9:YARA rule to detect malware version with regular expressions.

2.4 Detecting Malware while avoiding false positives.

I was tasked with detecting files with the string “malicious_behaviour” but excludes any files that contain the string “success”. I wrote a YARA rule with 2 variables in the string section: “\$malicious_behaviour” and “\$success”. The variables contain the strings “malicious_behaviour” and “success” respectively. I specified that the rule should detect files that included the string “malicious_behaviour” and not the string “success”. I named the rule “optimized_rule.yara”

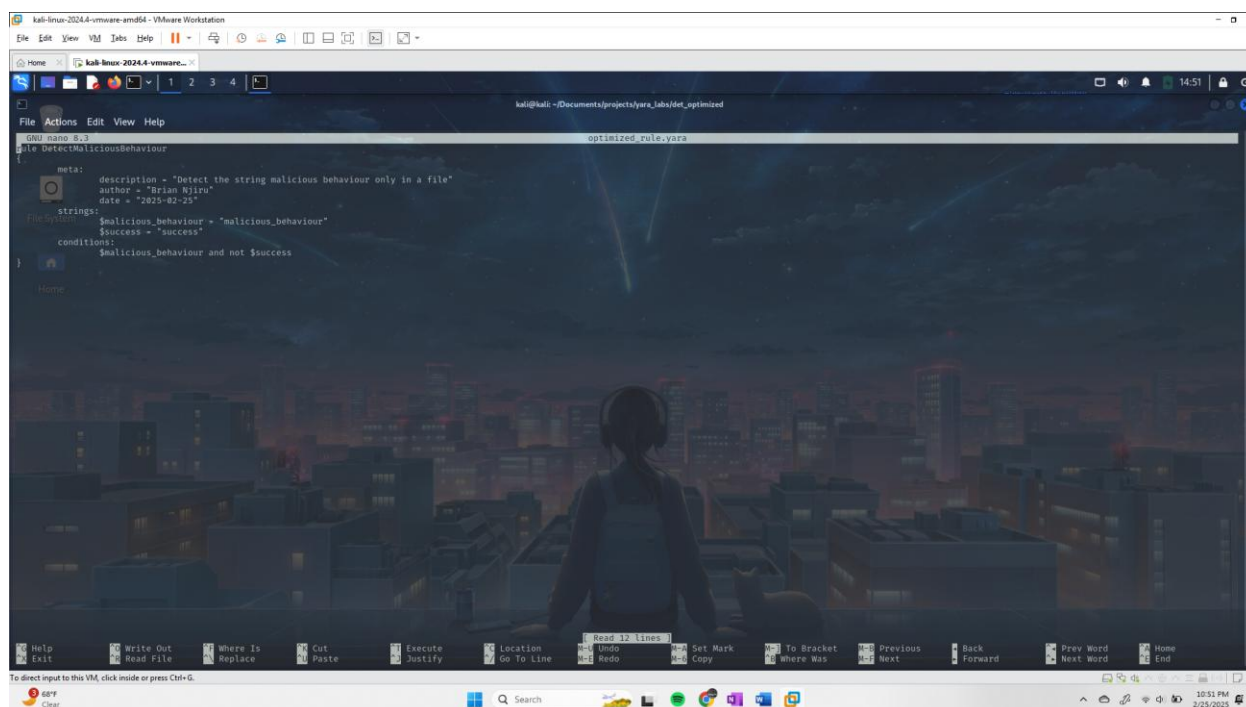
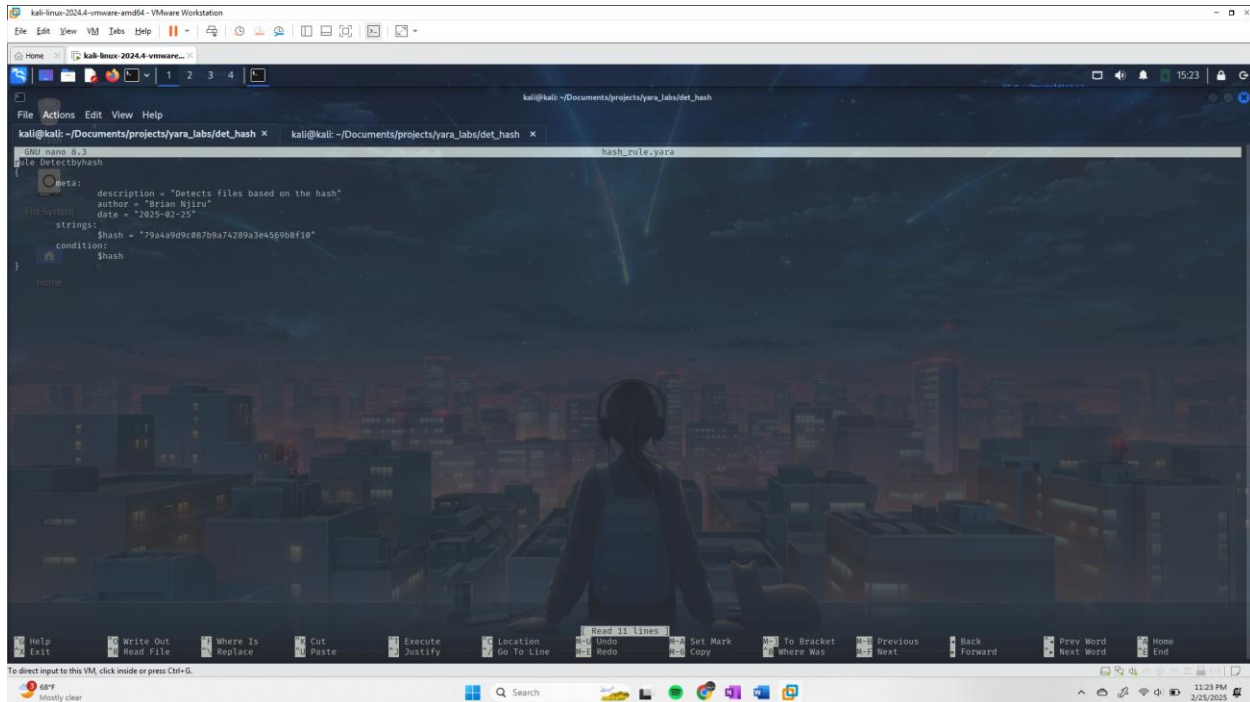


Figure 10:YARA rule to detect malware while avoiding false positives.

2.5 Detecting Malware based on its hash.

I was tasked with detecting a malware based on its hash. I wrote a rule with a variable **\$hash** which contained the hash for the file. I then gave the rule the condition that it should only detect files that matched that hash. I saved the file as **"hash_rule.yara"**.



The screenshot shows a Kali Linux virtual machine environment. A terminal window is open, displaying the contents of a file named `hash_rule.yara`. The YARA rule is as follows:

```
meta:
  description = "Detects files based on the hash"
  author = "Brian Hjiru"
  date = "2025-02-25"
strings:
  $hash = "79a49d9c087b9a74289a3e456908f18"
condition:
  $hash
```

The terminal window has a dark theme with a cityscape background. The top of the window shows the file editor interface with tabs for `hash_rule.yara` and `hash_rule.yara`. The bottom of the window shows the Windows taskbar with various application icons and the system clock indicating 11:23 PM on 2/25/2025.

Figure 11: YARA rule to identify a file based on its hash.

2.6 Detecting malware using a comprehensive rule.

I was tasked with detecting a piece of malware based on the following:

- The string “infected_file”
- The byte sequence {12 AB 34 CD}.
- The file was modified in the last 30 days.
- Exclude files from trusted sources based on hash.

I wrote a YARA rule with 2 variables in the strings section:

- **\$infected_file** variable contained the string “infected_file”.
- **\$byte_sequence** variable contained the byte sequence {12 AB 35 CD}

In the condition section I specified a set of conditions that would accurately detect the piece of malware hence completing the given objective for this section of the lab. I first checked for the presence of the string and the byte sequence using the condition “**any of (\$infected_file*) or any of (\$bytes)**”. I then ensured that the rule did not detect files from trusted sources using the condition that it doesn’t detect files with the known hash value, “**not (hash.sha256(0, filesize) == “TRUSTED_HASH”)**”. I replaced the value “**TRUSTED_HASH**” with the actual trusted hash of the file from the trusted source. I then saved the YARA rule as **comprehensive_rule.yara**.

The screenshot shows a Kali Linux virtual machine environment. A terminal window is open, displaying the contents of a YARA rule file named `comprehensive_rule.yara`. The rule is titled `rule ComprehensiveMalwareDetection` and includes a meta-section with a description, author, and date. The strings section defines `$infected_file` as "infected_file" and `$byte_sequence` as {12 AB 34 CD}. The condition section checks for the presence of either string or byte sequence, and excludes files with a specific SHA256 hash.

```

rule ComprehensiveMalwareDetection
{
  meta:
    description = "Detects piece of malware using a comprehensive set of conditions"
    author = "Brian Kijiru"
    date = "2023-02-27"

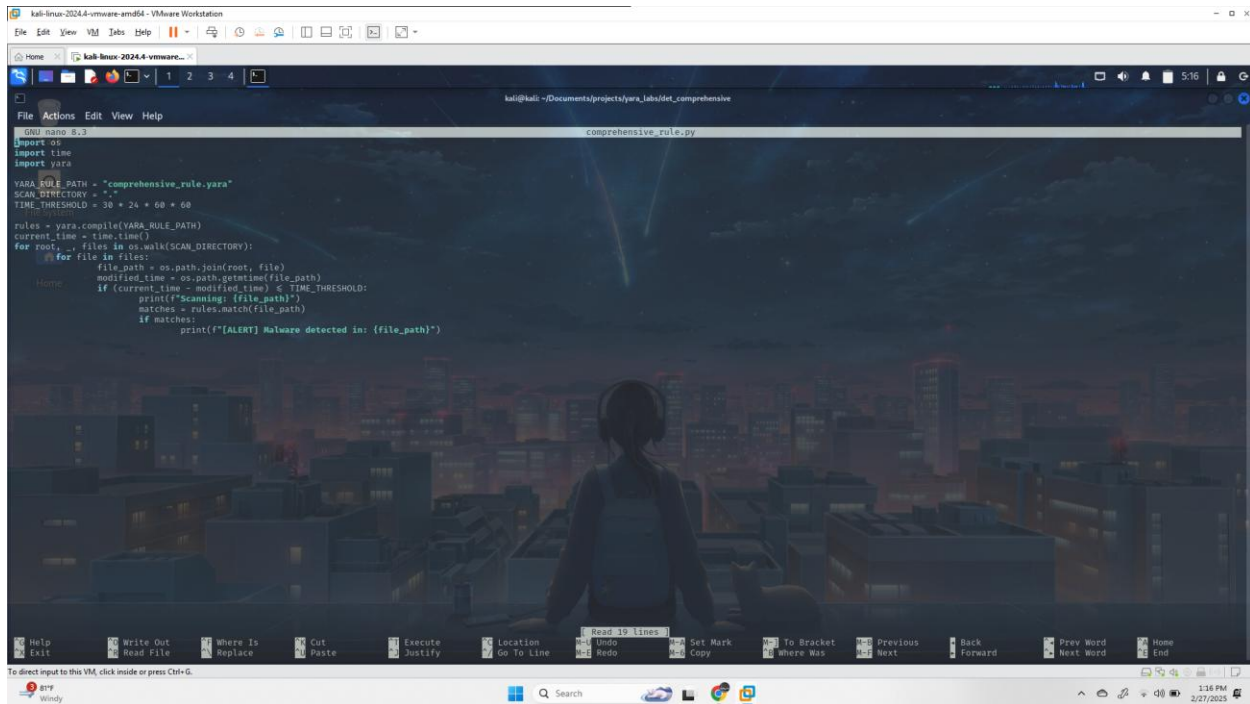
  strings:
    $infected_file = "infected_file"
    $byte_sequence = {12 AB 34 CD}

  condition:
    $infected_file or $byte_sequence and
    not (
      hash.sha256(0, filesize) == "e136958c2a7ad938a8312fb99058ac733ef5d77e347edda98936c6e9123f9ba8"
    )
}

```

Figure 12: YARA rule to detect piece of malware comprehensively

I then proceeded to write a python script to check the date of modification to ensure that the rule only detects files that have been modified in the last 30 days.



The screenshot shows a Kali Linux virtual machine running in VMware Workstation. The terminal window displays the nano text editor editing a file named 'comprehensive_rule.py'. The script is a Python program designed to scan a directory for files modified within a specific time threshold. The code includes imports for 'os' and 'yara', defines a YARA rule path and scan directory, sets a time threshold of 30 days, and iterates through files in the directory to check their modification times against the threshold. If a match is found, it prints an alert message. The background of the terminal window features a cityscape at night with a person wearing headphones.

```
#!/usr/bin/env python3
import os
import time
import yara

YARA_RULE_PATH = "comprehensive_rule.yara"
SCAN_DIRECTORY = "."
TIME_THRESHOLD = 30 * 24 * 60 * 60

rules = yara.compile(YARA_RULE_PATH)
current_time = time.time()
for root, _, files in os.walk(SCAN_DIRECTORY):
    for file in files:
        file_path = os.path.join(root, file)
        modified_time = os.path.getmtime(file_path)
        if (current_time - modified_time) < TIME_THRESHOLD:
            print("Scanning: {}".format(file_path))
            matches = rules.match(file_path)
            if matches:
                print("[ALERT] Malware detected in: {}".format(file_path))
```

Figure 13: Python script to check for the date of modification.

3. TESTING THE RULES.

In this section I tested the rules that I was tasked with creating in the previous section.

3.1 Testing detect_string_rule.yara

I tested the YARA rule, “**detect_string_rule.yara**” on some sample files provided for this lab.
The sample files are:

- **malware_file.exe**, this file contains the string “**This file contains malicious_string**”

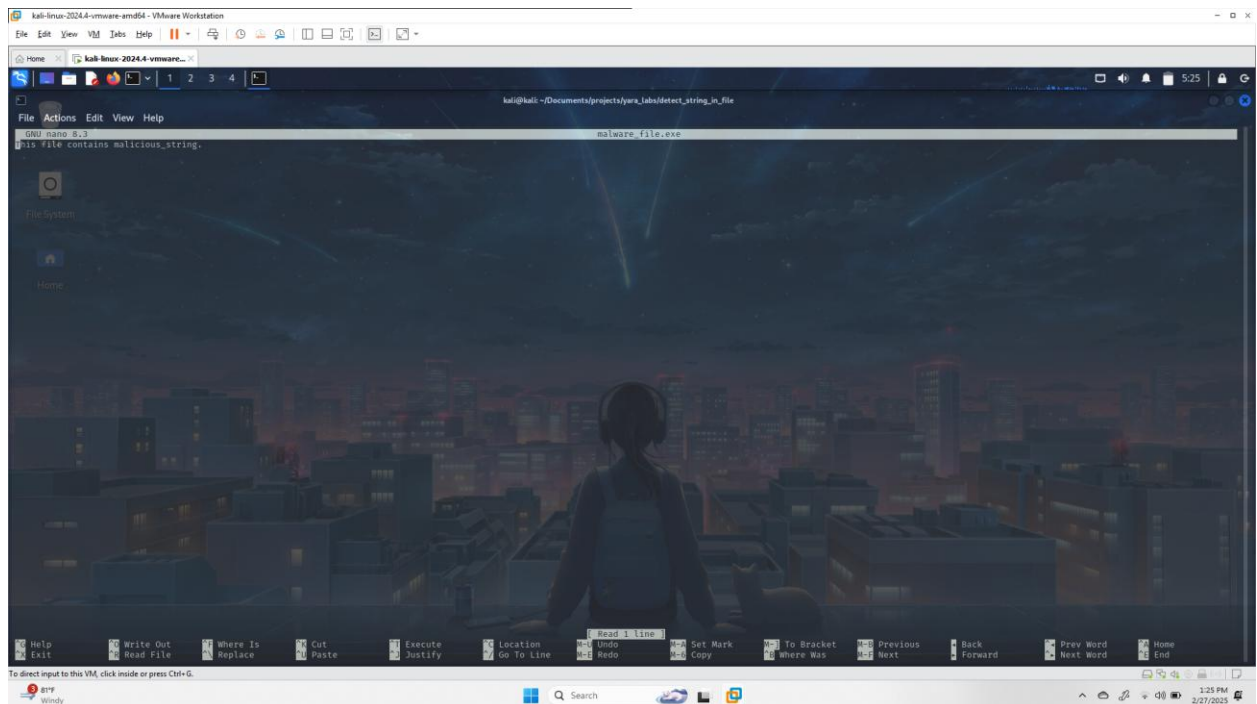


Figure 14: malware_file.exe

- **benign_file.txt**, this file contains the string “This is a safe document”.

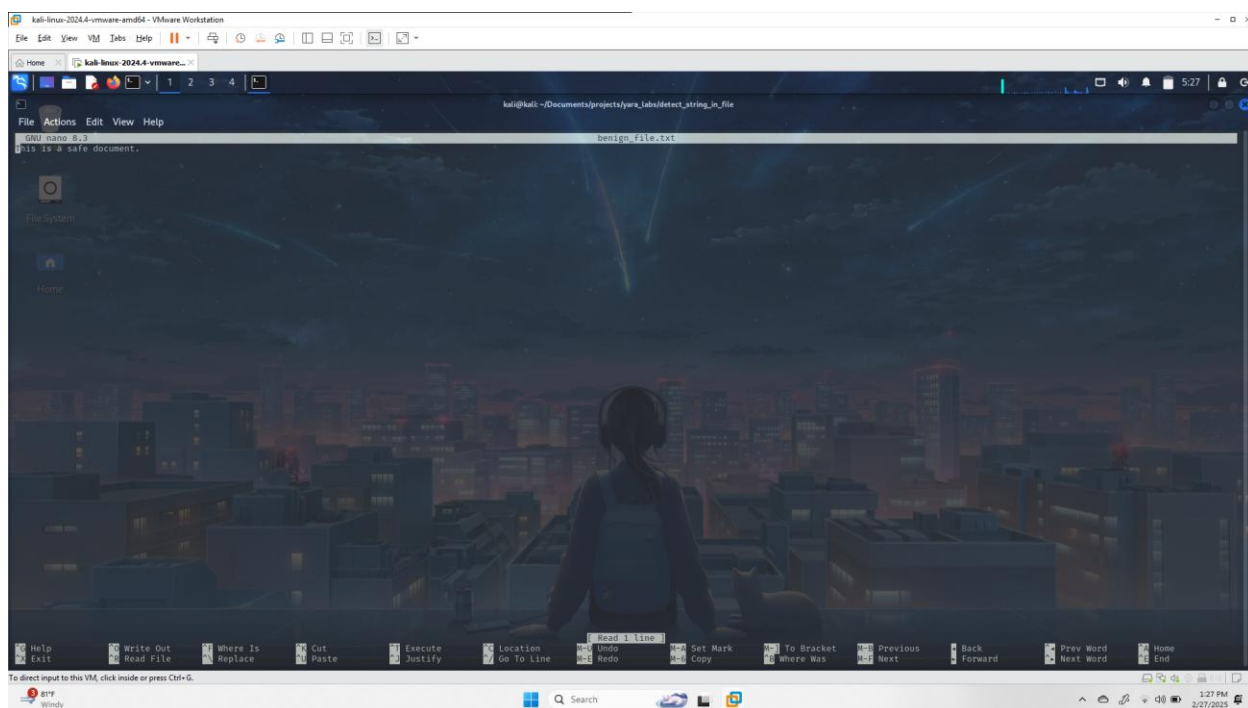


Figure 15: benign_file.txt

I ran the YARA rule using the command “**yara -r detect_string_rule.yara samples**” where samples is the name of the folder where the sample files are stored.

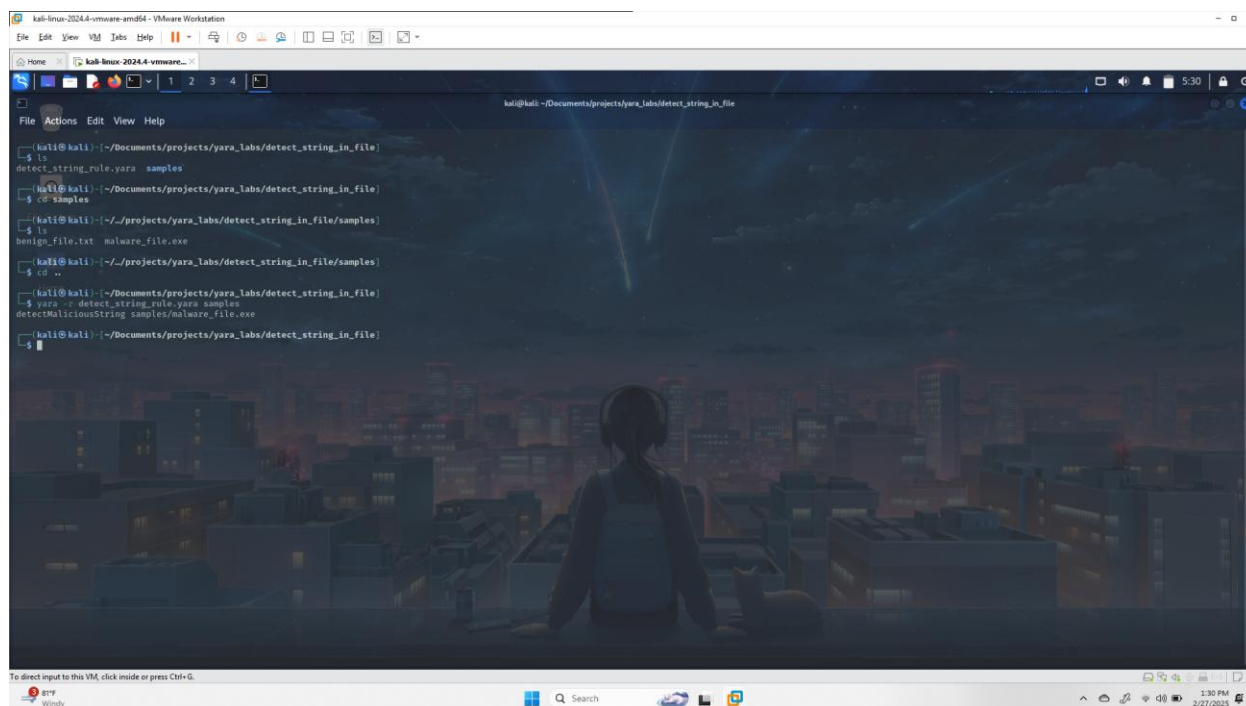


Figure 16: Running the YARA rule detect_string_rule.yara

The rule accurately identified the file and behaved according to the expected outcomes.

The rule did not match **benign_file.txt** but matched the file **malware_file.exe**.

3.2 Testing evil_software_rule.yara

I tested the YARA rule “**evil_software_rule.yara**” on some sample files provided for this activity. The sample files are:

- **evil_sample.exe**, this file contains the string “**This file contains EvilSoftware code**” and the byte sequence **{E8 34 12 56}**

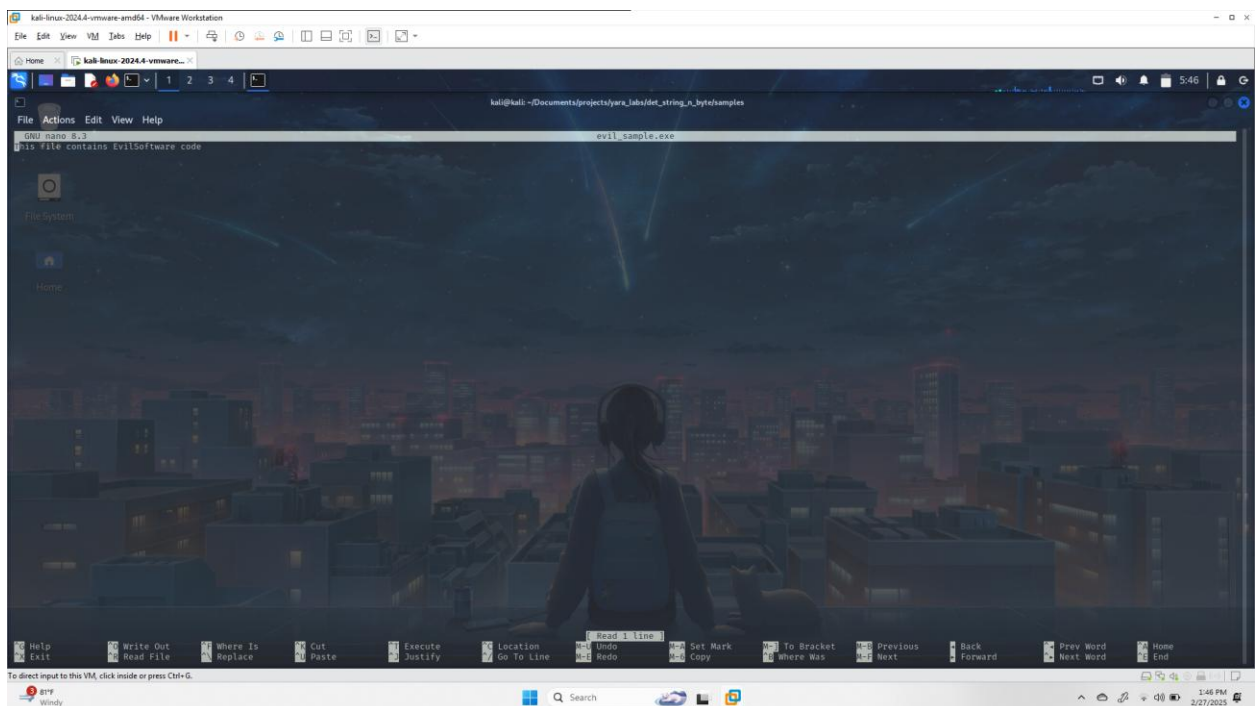


Figure 17: evil_sample.exe

- **benign_file.txt**, this file contains the string “This is a safe document”.

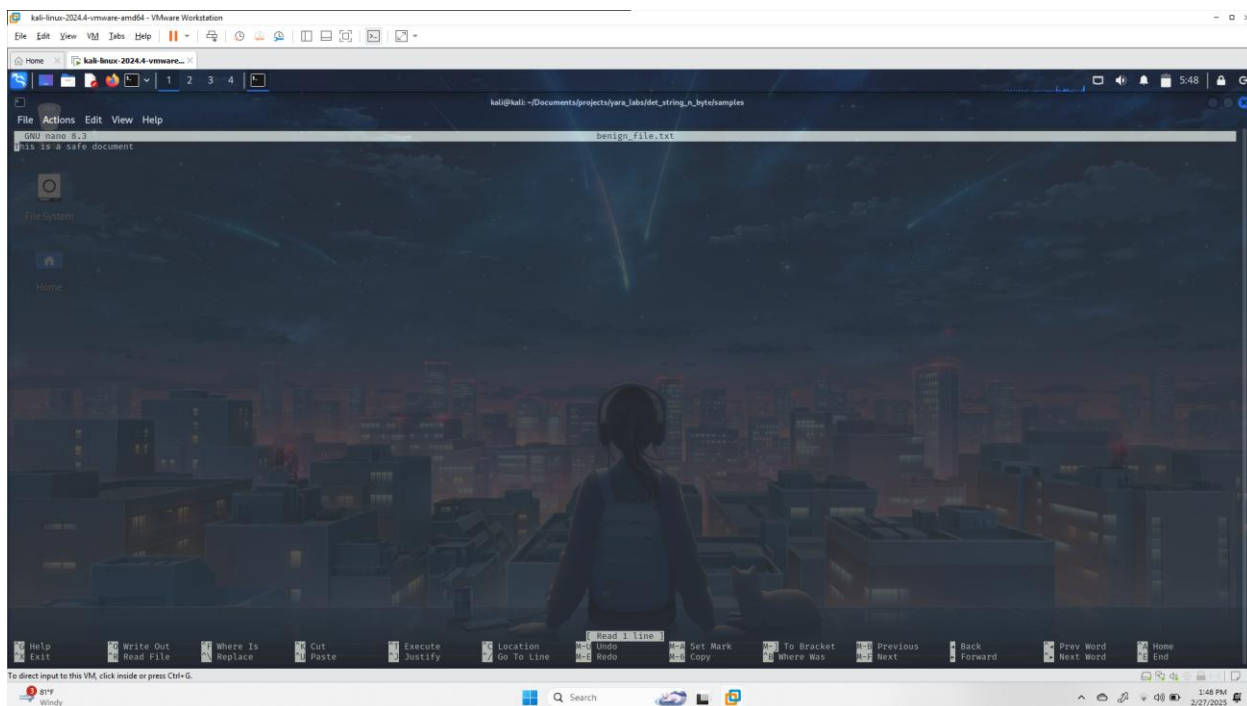


Figure 18: benign_file.txt

I proceeded to run the YARA rule and evaluate its outcome. I ran the command “**yara -r evil_software_rule.yara samples**”, where samples is the name of the folder where the sample files are stored.

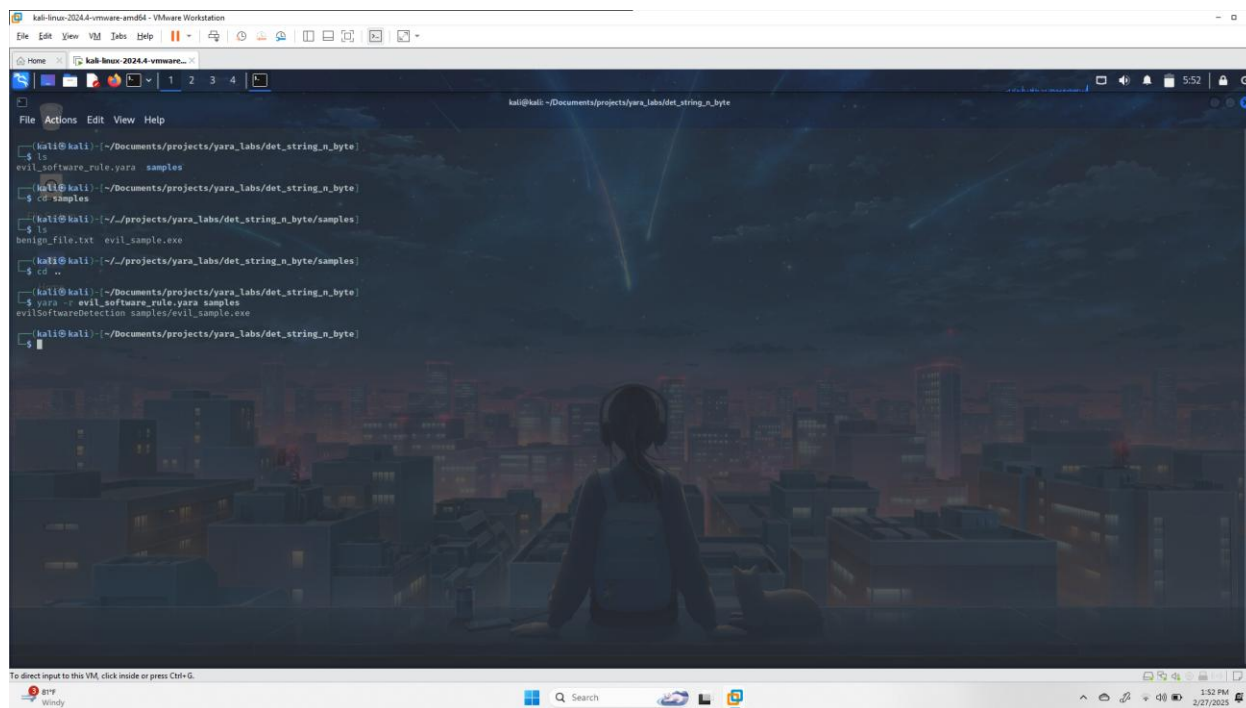


Figure 19: Running the YARA rule, evil_software_rule.yara

The YARA rule worked as expected: It matched the file **evil_sample.exe** and did not match the file **benign_file.txt**.

3.3 Testing regex_rule.yara

I tested the YARA rule **regex_rule.yara** on the sample files specified by the lab. The samples files are analyzed below:

The sample files include 2 files that are made to depict different versions of a malware family and are named using the pattern “**malware_vX**” where **X** is any number. The files are listed below:

- **malware_v1.exe**, This file contains the string “**Malicious content**”

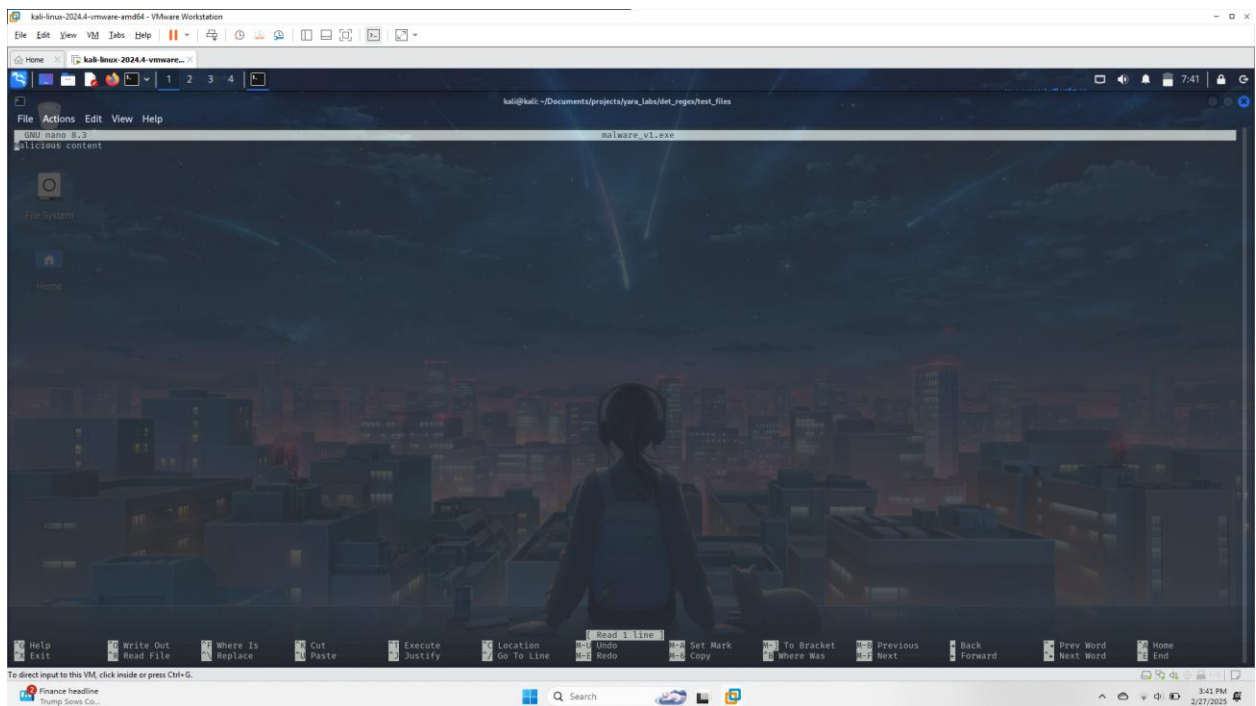


Figure 20: malware_v1.exe

- **malware_v2.exe**, This file contains the string “**Malicious content**”

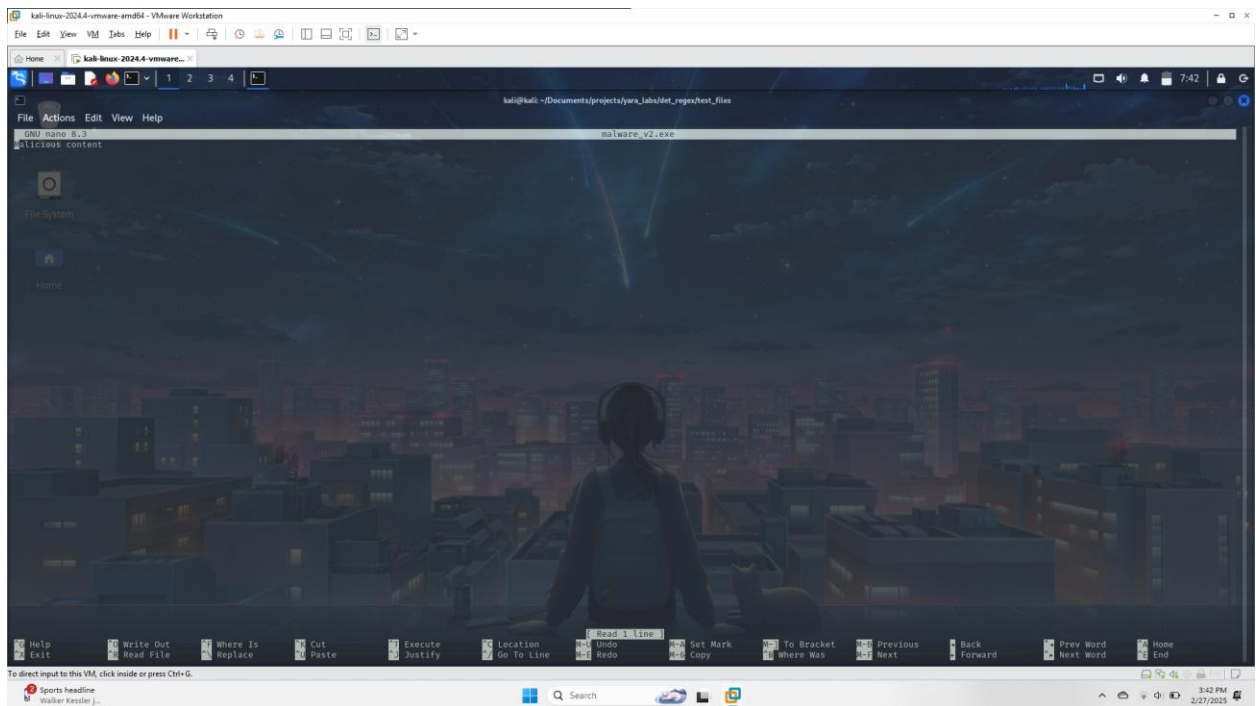


Figure 21: malware_v2.exe

- **random_file.txt**, This file contains the string “**Safe text**”.

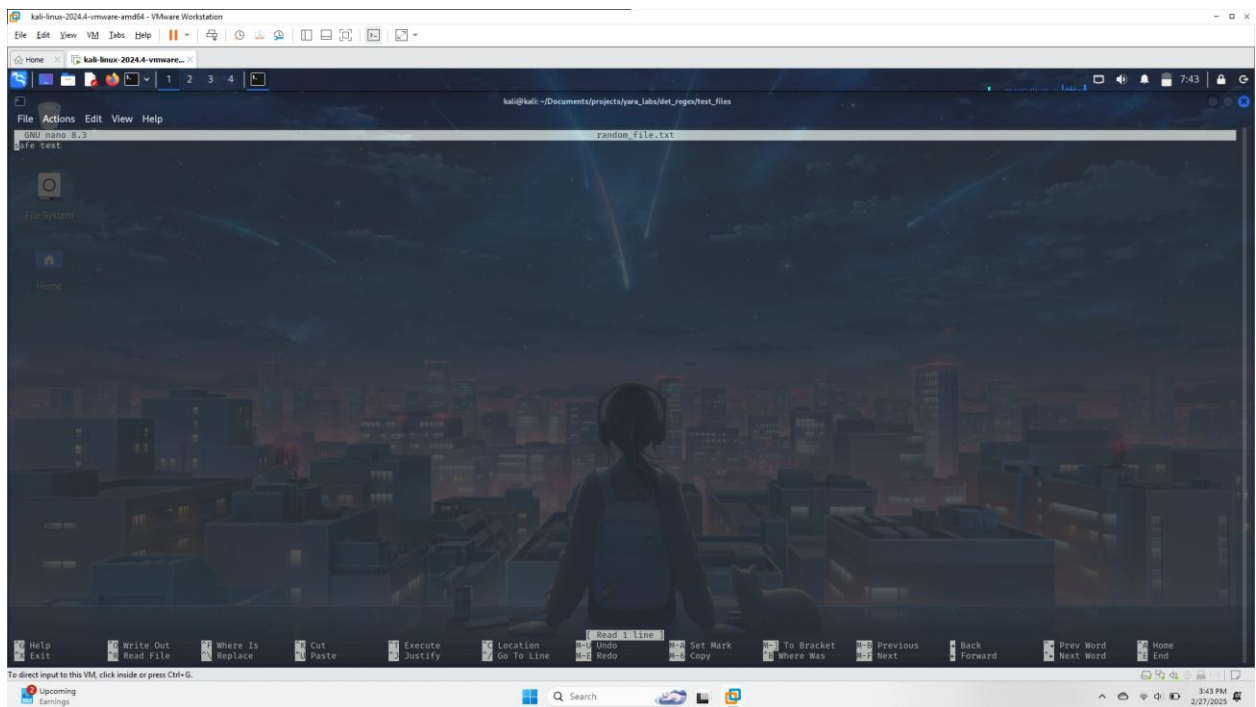


Figure 22: random_file.txt

I ran the YARA rule on the sample files using the command, “**yara -r regex_rule.yara test_files**” where **test_files** is the name of the directory where the sample files are stored.

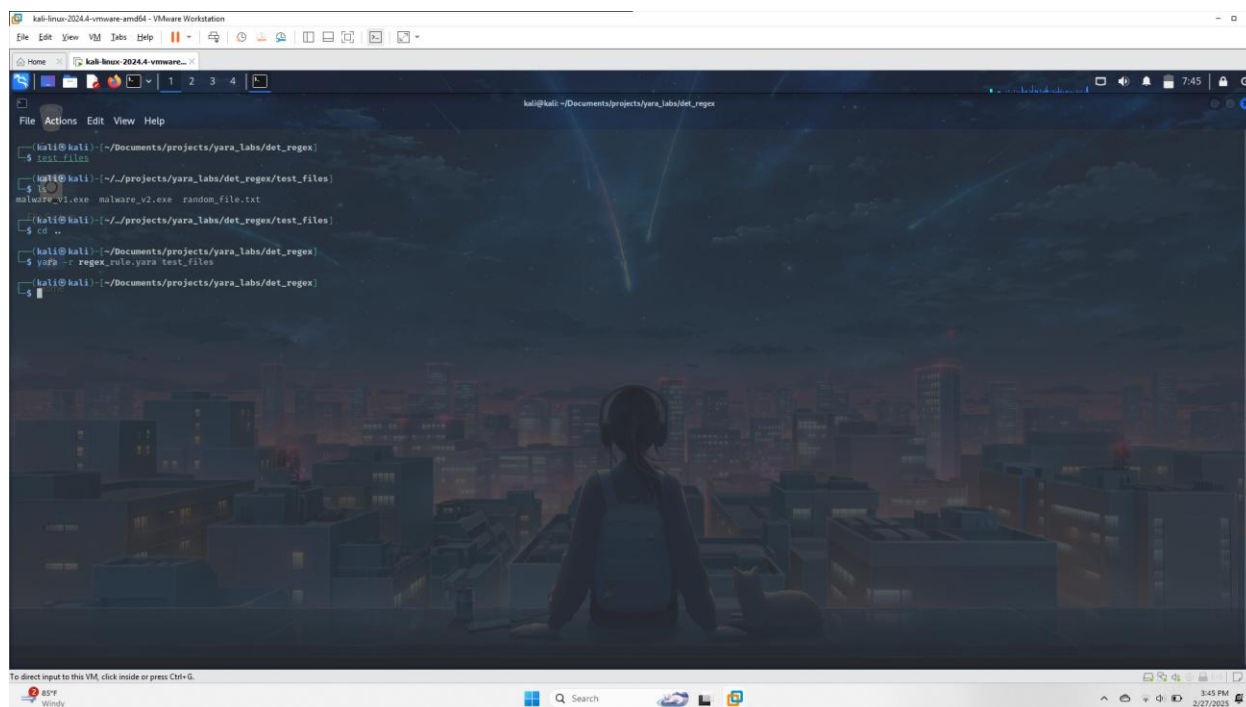


Figure 23: Running `regex_rule.yara`

I ran the YARA rule **regex_rule.yara** and received an unexpected outcome which will be further discussed in the next section.

3.4 Testing optimized_rule.yara

I tested the YARA rule “**optimized_rule.yara**” on some sample files provided by the lab for this activity. The sample files are discussed below:

- **benign_file.txt**, this file contains the string “**This file is safe. success**”.

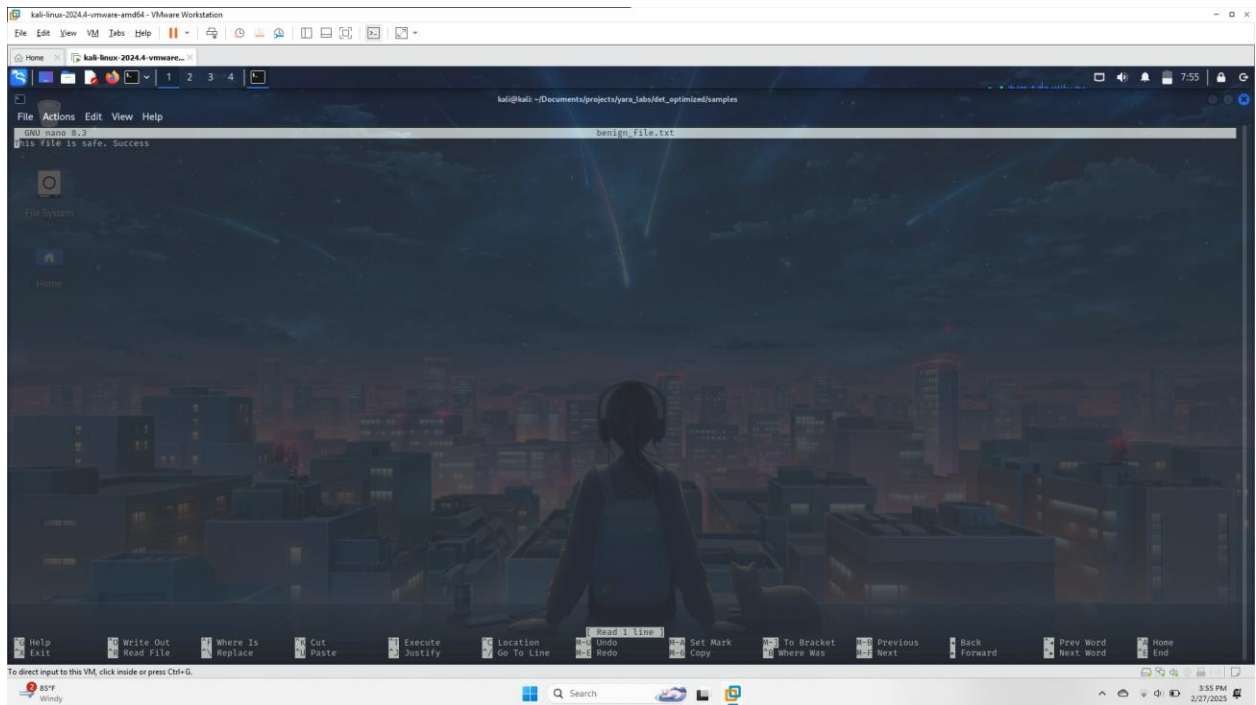


Figure 24: benign_file.txt

- **malware_1.exe**, this file contains the string “**This is a malicious file with malicious behavior**”

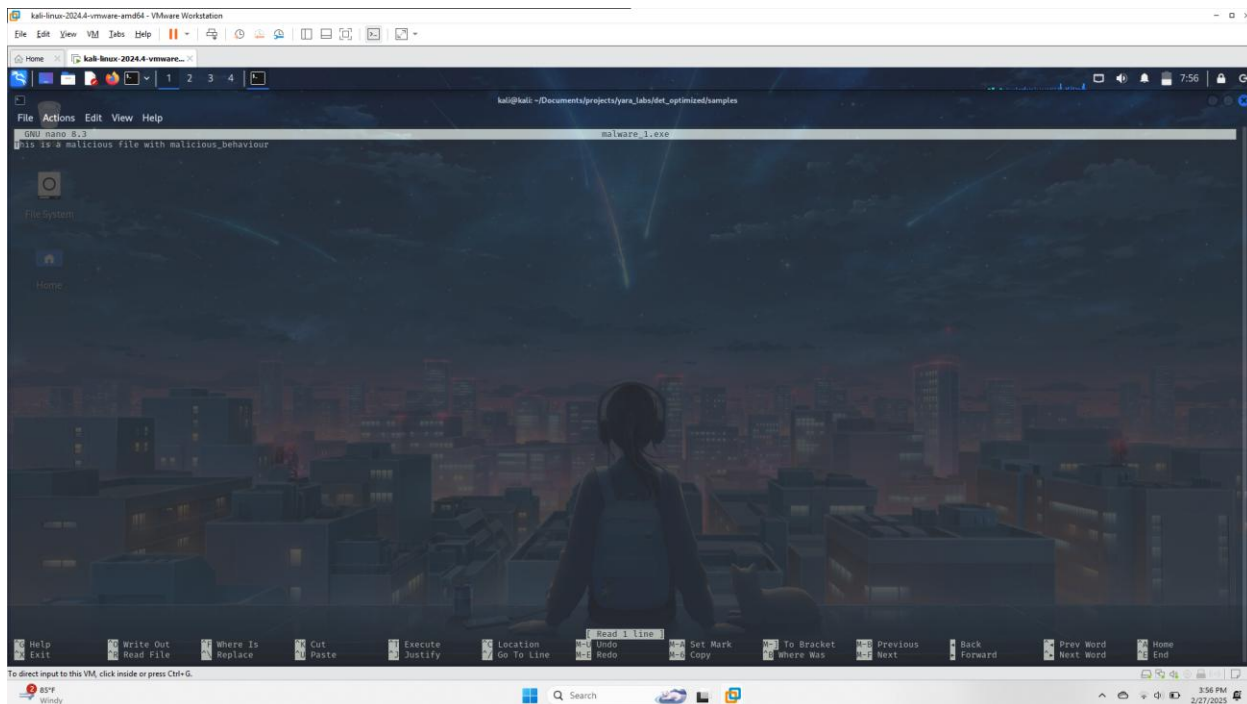


Figure 25: malware_1.exe

- **malware_2.exe**, This file contains the string “**This file contains both malicious_behaviour and success**”

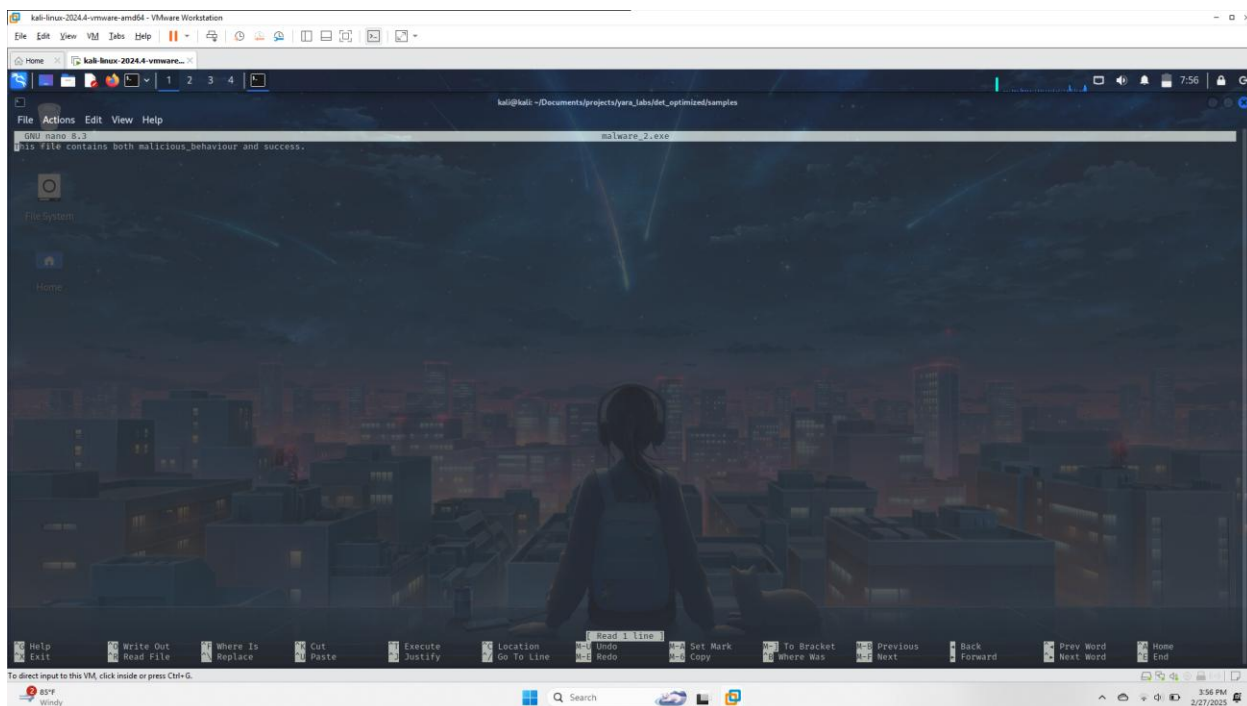


Figure 26: malware_2.exe

I proceeded to run the YARA rule using the command, “**yara -r optimized_rule.yara samples**” where samples is the directory where the sample files are stored.

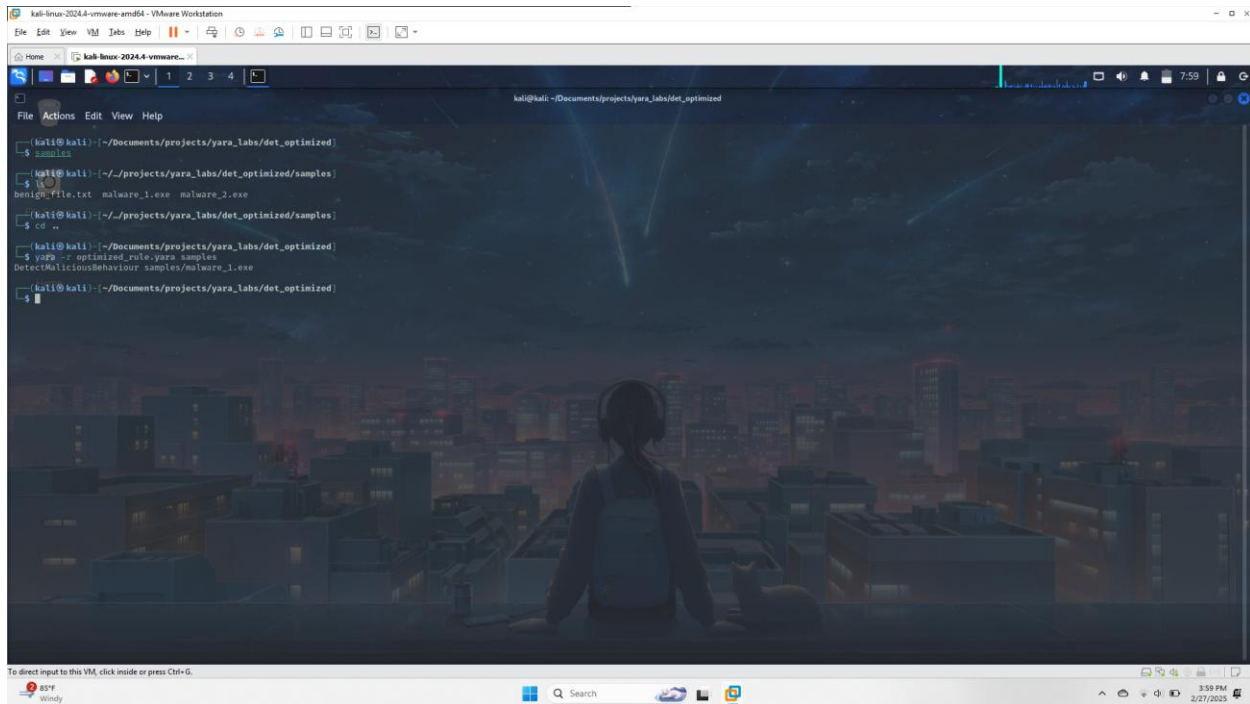


Figure 27: Running `optimized_rule.yara`

The YARA rule, **optimized_rule.yara** ran as expected, it matched the file **malware_1.exe** and did not match the files **malware_2.exe** and **benign_file.txt**.

3.5 Testing hash_rule.yara

I tested the YARA rule “**hash_rule.yara**” on sample files provided in the lab for this activity. The sample files are discussed below:

- **known_malware.exe**, this file contains the string “**Malware with known hash**” and its hash is known.

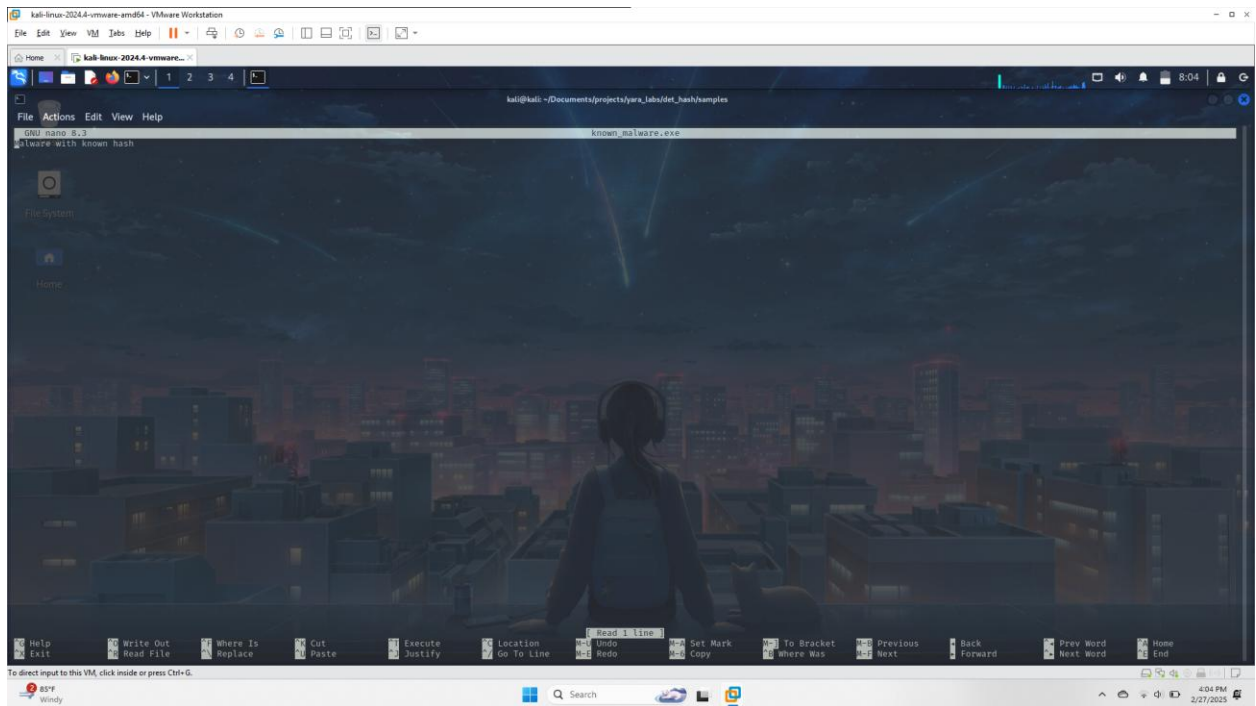


Figure 28: *known_malware.exe*

- **benign_file.txt**, this file contains the string “This file is safe”

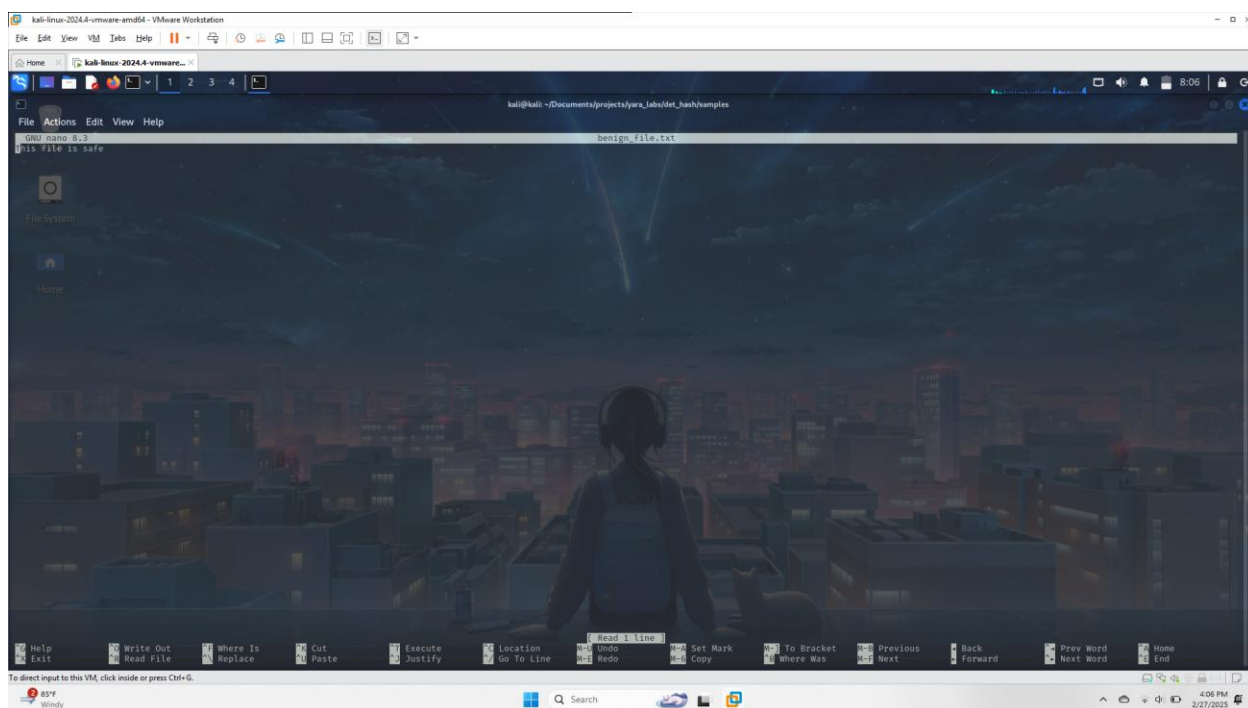


Figure 29: benign_file.txt

I ran the YARA rule using the command, “**yara -r hash_rule.yara samples**” where samples is the name of the directory where the sample files are stored.

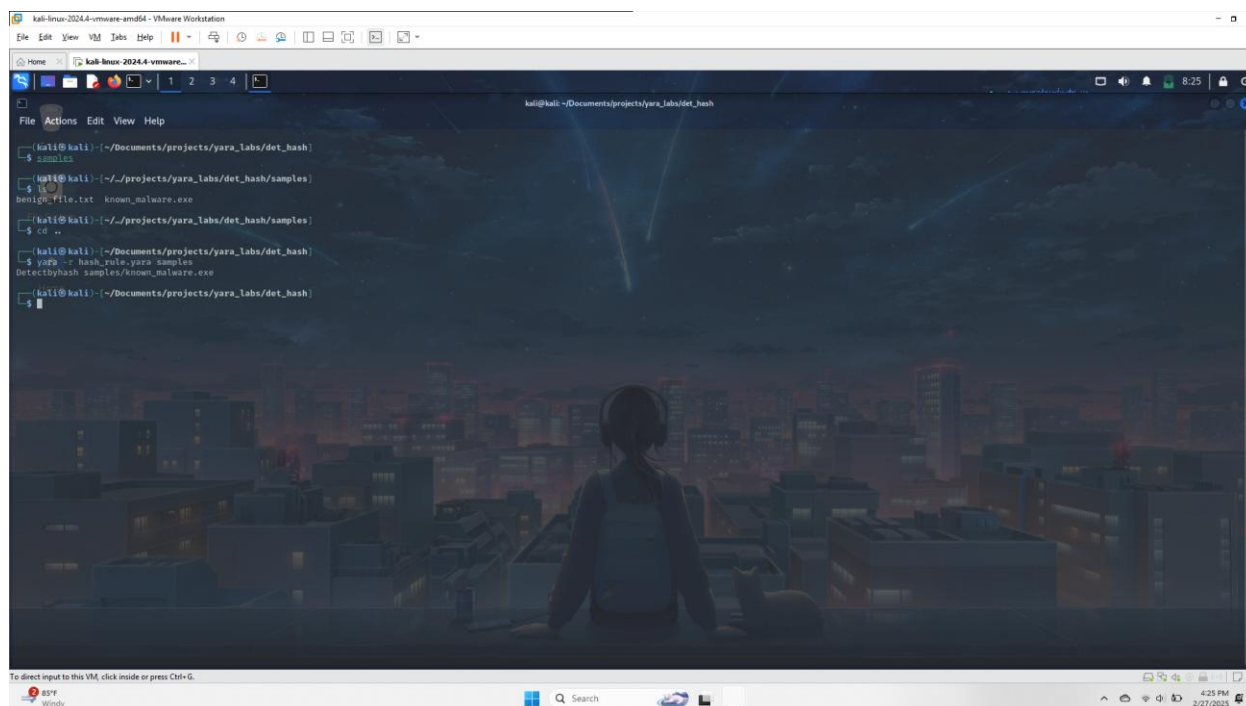


Figure 30: Running hash_rule.yara

The rule ran as expected. It matched the file **known_malware.exe** and did not match the file **benign_file.txt**.

4. CHALLENGES AND LEARNINGS

4.1 Unsolved issue using Regular Expressions in YARA.

I encountered a problem using Regular Expressions to detect different versions of a malware family in section **2.3**. I was unable to solve the issue in YARA and had to employ different methods to achieve it.

Scouring through the YARA rules documentation, I did not find a solution to the problem and so I resorted to using bash script to detect the malware families. I wrote a script that created a list that contained the names of each file in a directory. It then used Regular Expression to pinpoint each file.

In all fairness this method did not use YARA rules and I am still stuck here in terms of the use of Regular Expression, which explains the delayed submission of this report.

5. CONCLUSION

I conclude that the objectives of the lab were achieved by employing the methods taught to detect malware using YARA rules. I have understood the robustness of YARA rules as an important tool in a malware analyst's arsenal and through the activities of this lab, I am confident in my skills.