**CYBERSECURITY PORTFOLIO**

Prepared by: Edgar Abanum

Entry-Level Cybersecurity Analyst

September 2025

Email: edgar.tobore@gmail.com | LinkedIn: linkedin.com/in/engr-edgar-t-a

Table of Contents

[**About Me** 4](#_Toc209782950)

[Key Skills: 4](#_Toc209782951)

[**Project 1: Apply filters to SQL queries** 5](#_Toc209782952)

[Project description 5](#_Toc209782953)

[Retrieve after hours failed login attempts 5](#_Toc209782954)

[Retrieve login attempts on specific dates 5](#_Toc209782955)

[Retrieve login attempts outside of Mexico 6](#_Toc209782956)

[Retrieve employees in Marketing 7](#_Toc209782957)

[Retrieve employees in Finance or Sales 7](#_Toc209782958)

[Retrieve all employees not in IT 8](#_Toc209782959)

[Summary 9](#_Toc209782960)

[**Project 2: Incident Handler’s Journal** 10](#_Toc209782961)

[Overview 10](#_Toc209782962)

[Key Activities 10](#_Toc209782963)

[Table 1 11](#_Toc209782964)

[Table 2 12](#_Toc209782965)

[Table 3 12](#_Toc209782966)

[Table 5 13](#_Toc209782967)

[Reflections & Lessons Learned 14](#_Toc209782968)

[**File permissions in Linux** 15](#_Toc209782969)

[Project description 15](#_Toc209782970)

[Check file and directory details 15](#_Toc209782971)

[The permissions string 15](#_Toc209782972)

[Change file permissions 16](#_Toc209782973)

[Change file permissions on a hidden file 17](#_Toc209782974)

[Change directory permissions 17](#_Toc209782975)

[Summary 18](#_Toc209782976)

[**Project 3: Importing and Parsing Security Logs with Python** 19](#_Toc209782977)

[**Overview** 19](#_Toc209782978)

[Tools & Technologies 19](#_Toc209782979)

[Key Activities 19](#_Toc209782980)

[Outcome & Skills Gained 27](#_Toc209782981)

[**Project 4: Suricata IDS Rule Creation & Log Analysis** 28](#_Toc209782982)

[**Project Overview** 28](#_Toc209782983)

[**Objectives** 28](#_Toc209782984)

[**Task 1: Examine a custom rule in Suricata** 28](#_Toc209782985)

[**Rule Creation** 28](#_Toc209782986)

[Task 2. Rule Execution & Log Analysis 28](#_Toc209782987)

[Task 3. Log Analysis 30](#_Toc209782988)

[Correlating Events by flow\_id 33](#_Toc209782989)

[Skills Demonstrated 34](#_Toc209782990)

[Key Takeaways 34](#_Toc209782991)

[**Project 5: Update a file through a Python algorithm** 35](#_Toc209782992)

[Project description 35](#_Toc209782993)

[Open the file that contains the allow list 35](#_Toc209782994)

[For the first part of the algorithm, I opened the "allow\_list.txt" file. First, I assigned this file name as a string to the import\_file variable: 35](#_Toc209782995)

[Read the file contents 35](#_Toc209782996)

[Convert the string into a list 36](#_Toc209782997)

[Iterate through the remove list 36](#_Toc209782998)

[Remove IP addresses that are on the remove list 37](#_Toc209782999)

[Update the file with the revised list of IP addresses 38](#_Toc209783000)

[Summary 39](#_Toc209783001)

[**Project 6: Vulnerability Assessment Report** 40](#_Toc209783002)

[Vulnerability Assessment Project – E-Commerce Database Server 40](#_Toc209783003)

[Overview 40](#_Toc209783004)

[System Description 40](#_Toc209783005)

[Scope 40](#_Toc209783006)

[Purpose 40](#_Toc209783007)

[Risk Assessment 40](#_Toc209783008)

[Approach 41](#_Toc209783009)

[Remediation Strategy 41](#_Toc209783010)

[Key Skills Demonstrated 41](#_Toc209783011)

[**Project 7: Packet Capture and Analysis with Wireshark and Tcpdump** 42](#_Toc209783012)

[Overview 42](#_Toc209783013)

[Tools & Technologies 42](#_Toc209783014)

[Key Activities & Screenshots 42](#_Toc209783015)

[Outcome & Skills Gained 47](#_Toc209783016)

# **About Me**

Hello! My name is **Edgar Abanum**, and I am an entry-level **Cybersecurity Analyst** with a solid foundation in network security, threat detection, and incident response. I recently completed the **Google Cybersecurity Professional Certificate**, where I gained hands-on experience with tools such as **Wireshark, Splunk, Linux, SQL, and Python** for security analysis and automation.

I am passionate about protecting organizations from evolving cyber threats and enjoy working on projects related to **log analysis, vulnerability assessments, phishing detection, and network monitoring**. My goal is to continuously develop my technical and analytical skills while contributing to a team dedicated to improving security posture.

Outside of learning and building projects, I stay up to date with the latest security trends by following resources like **Krebs on Security, Dark Reading, and OWASP**.

### Key Skills:

* Security Monitoring & Incident Response
* SIEM (Splunk)
* Log Analysis & Threat Detection
* Networking (TCP/IP, DNS, Firewalls)
* Python & SQL for Security Automation

# **Project 1: Apply filters to SQL queries**

### Project description

My organization is working to make their system more secure. It is my job to ensure the system is safe, investigate all potential security issues, and update employee computers as needed. The following steps provide examples of how I used SQL with filters to perform security-related tasks.

### Retrieve after hours failed login attempts

There was a potential security incident that occurred after business hours (after 18:00). All after hours login attempts that failed need to be investigated.

The following code demonstrates how I created a SQL query to filter for failed login attempts that occurred after business hours.



The first part of the screenshot is my query, and the second part is a portion of the output. This query filters for failed login attempts that occurred after 18:00. First, I started by selecting all data from the log\_in\_attempts table. Then, I used a WHERE clause with an AND operator to filter my results to output only login attempts that occurred after 18:00 and were unsuccessful. The first condition is login\_time > '18:00', which filters for the login attempts that occurred after 18:00. The second condition is success = FALSE, which filters for the failed login attempts.

### Retrieve login attempts on specific dates

A suspicious event occurred on 2022-05-09. Any login activity that happened on 2022-05-09 or on the day before needs to be investigated.

The following code demonstrates how I created a SQL query to filter for login attempts that occurred on specific dates.



The first part of the screenshot is my query, and the second part is a portion of the output. This query returns all login attempts that occurred on 2022-05-09 or 2022-05-08. First, I started by selecting all data from the log\_in\_attempts table. Then, I used a WHERE clause with an OR operator to filter my results to output only login attempts that occurred on either 2022-05-09 or 2022-05-08. The first condition is login\_date = '2022-05-09', which filters for logins on 2022-05-09. The second condition is login\_date = '2022-05-08', which filters for logins on 2022-05-08.

### Retrieve login attempts outside of Mexico

After investigating the organization’s data on login attempts, I believe there is an issue with the login attempts that occurred outside of Mexico. These login attempts should be investigated.

The following code demonstrates how I created a SQL query to filter for login attempts that occurred outside of Mexico.



The first part of the screenshot is my query, and the second part is a portion of the output. This query returns all login attempts that occurred in countries other than Mexico. First, I started by selecting all data from the log\_in\_attempts table. Then, I used a WHERE clause with NOT to filter for countries other than Mexico. I used LIKE with MEX% as the pattern to match because the dataset represents Mexico as MEX and MEXICO. The percentage sign (%) represents any number of unspecified characters when used with LIKE.

### Retrieve employees in Marketing

My team wants to update the computers for certain employees in the Marketing department. To do this, I have to get information on which employee machines to update.

The following code demonstrates how I created a SQL query to filter for employee machines from employees in the Marketing department in the East building.



The first part of the screenshot is my query, and the second part is a portion of the output. This query returns all employees in the Marketing department in the East building. First, I started by selecting all data from the employees table. Then, I used a WHERE clause with AND to filter for employees who work in the Marketing department and in the East building. I used LIKE with East% as the pattern to match because the data in the office column represents the East building with the specific office number. The first condition is the department = 'Marketing' portion, which filters for employees in the Marketing department. The second condition is the office LIKE 'East%' portion, which filters for employees in the East building.

### Retrieve employees in Finance or Sales

The machines for employees in the Finance and Sales departments also need to be updated. Since a different security update is needed, I have to get information on employees only from these two departments.

The following code demonstrates how I created a SQL query to filter for employee machines from employees in the Finance or Sales departments.



The first part of the screenshot is my query, and the second part is a portion of the output. This query returns all employees in the Finance and Sales departments. First, I started by selecting all data from the employees table. Then, I used a WHERE clause with OR to filter for employees who are in the Finance and Sales departments. I used the OR operator instead of AND because I want all employees who are in either department. The first condition is department = 'Finance', which filters for employees from the Finance department. The second condition is department = 'Sales', which filters for employees from the Sales department.

### Retrieve all employees not in IT

My team needs to make one more security update on employees who are not in the Information Technology department. To make the update, I first have to get information on these employees.

The following demonstrates how I created a SQL query to filter for employee machines from employees not in the Information Technology department.



The first part of the screenshot is my query, and the second part is a portion of the output. The query returns all employees not in the Information Technology department. First, I started by selecting all data from the employees table. Then, I used a WHERE clause with NOT to filter for employees not in this department.

### Summary

I applied filters to SQL queries to get specific information on login attempts and employee machines. I used two different tables, log\_in\_attempts and employees. I used the AND, OR, and NOT operators to filter for the specific information needed for each task. I also used LIKE and the percentage sign (%) wildcard to filter for patterns.

# **Project 2: Incident Handler’s Journal**

### Overview

As part of my cybersecurity training, I developed an Incident Handler’s Journal to practice documenting and analyzing simulated incidents. This project demonstrates my ability to track incident response phases, use investigative tools, and reflect on lessons learned.

### Key Activities

#### 1. Documenting a Ransomware Incident

- Recorded incident details using the 5 W’s framework.  
- Identified detection, analysis, containment, and recovery phases.  
- Assessed prevention strategies such as phishing awareness training and system hardening.

#### 2. Network Traffic Analysis with Wireshark

- Captured and analyzed packet data using Wireshark.  
- Interpreted protocols and potential anomalies in a packet capture file.  
- Gained familiarity with packet structure and how malicious activity may appear in traffic.

#### 3. Capturing Traffic with tcpdump

- Used `tcpdump` on the command line to capture and filter traffic.  
- Practiced applying capture filters to isolate relevant packets.  
- Overcame syntax errors by iterating commands and debugging.

#### 4. Investigating a Suspicious File Hash with VirusTotal

- Investigated a suspicious hash flagged by an IDS alert.  
- Used VirusTotal to analyze the file hash and determine whether it was malicious.  
- Applied incident response thinking by evaluating user behavior and recommending awareness training.

### Table 1

|  |  |
| --- | --- |
| **Date:** September 9, 2025 | **Entry:**  #1 |
| Description | Documenting a cybersecurity incident  This incident occurred in the two phases:   1. **Detection and Analysis**: The scenario outlines how the organization first detected the ransomware incident. For the analysis step, the organization contacted several organizations for technical assistance. 2. **Containment, Eradication, and Recovery**: The scenario details some steps that the organization took to contain the incident. For example, the company shut down their computer systems. However, since they could not work to eradicate and recover from the incident alone, they contacted several other organizations for assistance. |
| Tool(s) used | None |
| The 5 W's | * **Who**: An organized group of unethical hackers * **What**: A ransomware security incident * **Where**: At a health care company * **When**: Tuesday 9:00 a.m. * **Why**: The incident happened because unethical hackers were able to access the company's systems using a phishing attack. After gaining access, the attackers launched their ransomware on the company's systems, encrypting critical files. The attackers' motivation appears to be financial because the ransom note they left demanded a large sum of money in exchange for the decryption key. |
| Additional notes | 1. How could the health care company prevent an incident like this from occurring again? 2. Should the company pay the ransom to retrieve the decryption key? |

### Table 2

|  |  |
| --- | --- |
| **Date:** September 10 2025 | **Entry:** #2 |
| Description | Analyzing a packet capture file |
| Tool(s) used | For this activity, I used Wireshark to analyze a packet capture file. Wireshark is a network protocol analyzer that uses a graphical user interface. The value of Wireshark in cybersecurity is that it allows security analysts to capture and analyze network traffic. This can help in detecting and investigating malicious activity. |
| The 5 W's | * **Who**: N/A * **What**: N/A * **Where**: N/A * **When**: N/A * **Why**: N/A |
| Additional notes | I've never used Wireshark before, so I was excited to begin this exercise and analyze a packet capture file. At first glance, the interface was very overwhelming. I can see why it's such a powerful tool for understanding network traffic. |

### Table 3

|  |  |
| --- | --- |
| **Date:** September 11 2025 | **Entry:**  #3 |
| Description | Capturing my first packet |
| Tool(s) used | For this activity, I used tcpdump to capture and analyze network traffic. Tcpdump is a network protocol analyzer that's accessed using the command-line interface. Similar to Wireshark, the value of tcpdump in cybersecurity is that it allows security analysts to capture, filter, and analyze network traffic. |
| The 5 W's | * **Who**: N/A * **What**: N/A * **Where**: N/A * **When**: N/A * **Why**: N/A |
| Additional notes | I'm still new to using the command-line interface, so using it to capture and filter network traffic was a challenge. I got stuck a couple of times because I used the wrong commands. But after carefully following the instructions and redoing some steps, I was able to get through this activity and capture network traffic. |

### Table 5

|  |  |
| --- | --- |
| **Date:** September 12, 2025 | **Entry:**  #4 |
| Description | Investigate a suspicious file hash |
| Tool(s) used | For this activity, I used VirusTotal, which is an investigative tool that analyzes files and URLs for malicious content such as viruses, worms, trojans, and more. It's a very helpful tool to use if you want to quickly check if an indicator of compromise like a website or file has been reported as malicious by others in the cybersecurity community. For this activity, I used VirusTotal to analyze a file hash, which was reported as malicious.  This incident occurred in the **Detection and Analysis** phase. The scenario put me in the place of a security analyst at a SOC investigating a suspicious file hash. After the suspicious file was detected by the security systems in place, I had to perform deeper analysis and investigation to determine if the alert signified a real threat. |
| The 5 W's | * **Who**: An unknown malicious actor * **What**: An email sent to an employee contained a malicious file attachment with the SHA-256 file hash of 54e6ea47eb04634d3e87fd7787e2136ccfbcc80ade34f246a12cf93bab527f6b * **Where**: An employee's computer at a financial services company * **When**: At 1:20 p.m., an alert was sent to the organization's SOC after the intrusion detection system detected the file * **Why**: An employee was able to download and execute a malicious file attachment via e-mail. |
| Additional notes | How can this incident be prevented in the future? Should we consider improving security awareness training so that employees are careful with what they click on? |

|  |
| --- |
| Reflections & Lessons Learned - Developed stronger documentation skills for incident tracking. - Learned the importance of detection and response lifecycle (Preparation → Detection → Containment → Eradication → Recovery → Lessons Learned). - Built confidence using network analysis tools despite challenges with command-line syntax. - Realized that security awareness training is as important as technical defenses in preventing incidents. |

# **File permissions in Linux**

### Project description

The research team at my organization needs to update the file permissions for certain files and directories within the projects directory. The permissions do not currently reflect the level of authorization that should be given. Checking and updating these permissions will help keep their system secure. To complete this task, I performed the following tasks:

### Check file and directory details

The following code demonstrates how I used Linux commands to determine the existing permissions set for a specific directory in the file system.



The first line of the screenshot displays the command I entered, and the other lines display the output. The code lists all contents of the projects directory. I used the ls command with the -la option to display a detailed listing of the file contents that also returned hidden files. The output of my command indicates that there is one directory named drafts, one hidden file named .project\_x.txt, and five other project files. The 10-character string in the first column represents the permissions set on each file or directory.

### The permissions string

The 10-character string can be deconstructed to determine who is authorized to access the file and their specific permissions. The characters and what they represent are as follows:

* **1st character**: This character is either a d or hyphen (-) and indicates the file type. If it’s a d, it’s a directory. If it’s a hyphen (-), it’s a regular file.
* **2nd-4th characters**: These characters indicate the read (r), write (w), and execute (x) permissions for the user. When one of these characters is a hyphen (-) instead, it indicates that this permission is not granted to the user.
* **5th-7th characters:** These characters indicate the read (r), write (w), and execute (x) permissions for the group. When one of these characters is a hyphen (-) instead, it indicates that this permission is not granted for the group.
* **8th-10th characters:** These characters indicate the read (r), write (w), and execute (x) permissions for other. This owner type consists of all other users on the system apart from the user and the group. When one of these characters is a hyphen (-) instead, that indicates that this permission is not granted for other.

For example, the file permissions for project\_t.txt are -rw-rw-r--. Since the first character is a hyphen (-), this indicates that project\_t.txt is a file, not a directory. The second, fifth, and eighth characters are all r, which indicates that user, group, and other all have read permissions. The third and sixth characters are w, which indicates that only the user and group have write permissions. No one has execute permissions for project\_t.txt.

### Change file permissions

The organization determined that other shouldn't have write access to any of their files. To comply with this, I referred to the file permissions that I previously returned. I determined project\_k.txt must have the write access removed for other.

The following code demonstrates how I used Linux commands to do this:



The first two lines of the screenshot display the commands I entered, and the other lines display the output of the second command. The chmod command changes the permissions on files and directories. The first argument indicates what permissions should be changed, and the second argument specifies the file or directory. In this example, I removed write permissions from other for the project\_k.txt file. After this, I used ls -la to review the updates I made.

### Change file permissions on a hidden file

The research team at my organization recently archived project\_x.txt. They do not want anyone to have write access to this project, but the user and group should have read access.

The following code demonstrates how I used Linux commands to change the permissions:



The first two lines of the screenshot display the commands I entered, and the other lines display the output of the second command. I know .project\_x.txt is a hidden file because it starts with a period (.). In this example, I removed write permissions from the user and group, and added read permissions to the group. I removed write permissions from the user with u-w. Then, I removed write permissions from the group with g-w, and added read permissions to the group with g+r.

### Change directory permissions

My organization only wants the researcher2 user to have access to the drafts directory and its contents. This means that no one other than researcher2 should have execute permissions.

The following code demonstrates how I used Linux commands to change the permissions:



The first two lines of the screenshot display the commands I entered, and the other lines display the output of the second command. I previously determined that the group had execute permissions, so I used the chmod command to remove them. The researcher2 user already had execute permissions, so they did not need to be added.

### Summary

I changed multiple permissions to match the level of authorization my organization wanted for files and directories in the projects directory. The first step in this was using ls -la to check the permissions for the directory. This informed my decisions in the following steps. I then used the chmod command multiple times to change the permissions on files and directories.

# **Project 3: Importing and Parsing Security Logs with Python**

**Overview**

In this project, I worked with Python to import, parse, and manipulate security log files. The exercise simulated a real-world security analyst task of handling log data, appending missing entries, and creating allow-lists of trusted IP addresses.

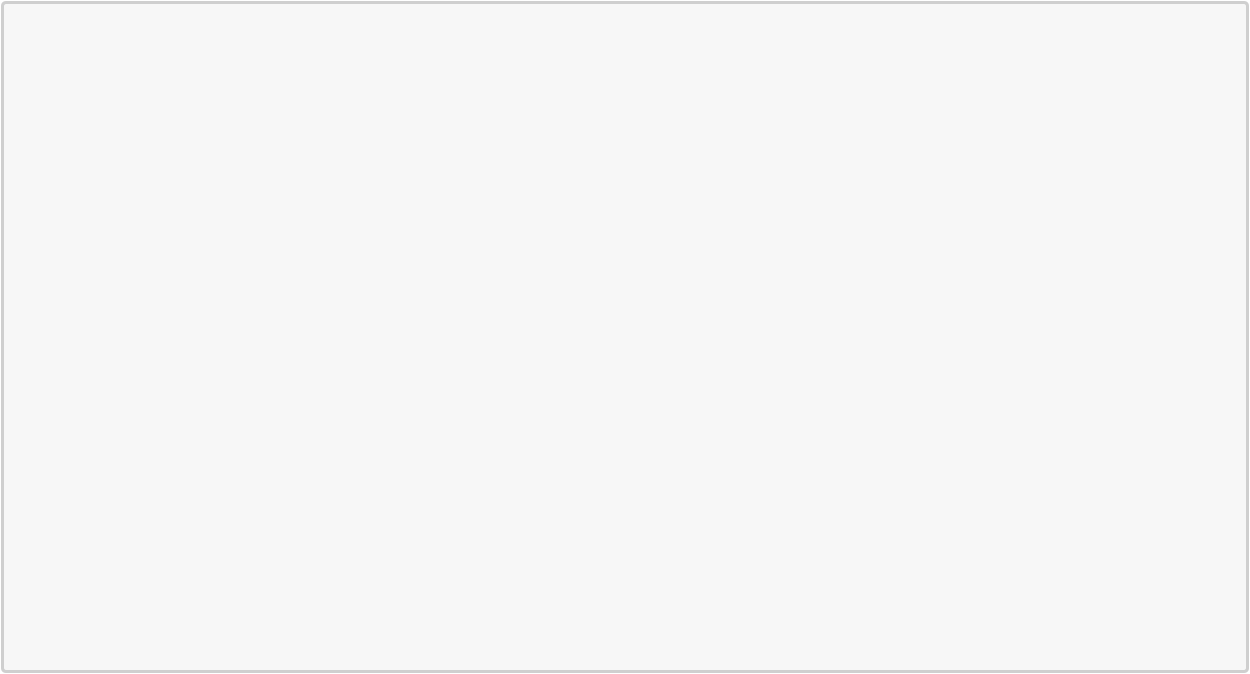
### ****Tools & Technologies****

* **Python** (file handling, string methods, list operations)
* **Text files (.txt)** for log storage and allow-lists

### ****Key Activities****

#### ****1. Importing and Reading Security Logs****

* Opened a log file (login.txt) using Python’s with open() statement.
* Read the entire log into a string variable using .read().
* Displayed the raw contents of the log file, which contained usernames, IP addresses, timestamps, and dates.



[1]: *# Assign `import\_file` to the name of the text file that contains the security*␣ *,→log file*

import\_file = "data/login.txt"

* *The`with` statement*
* *Use `open()` to import security log file and store it as a string*

**with** open(import\_file, "r") **as** file:

* *Use `.read()` to read the imported file and store the result in a variable*␣ *,→named `text`*

text = file.read()

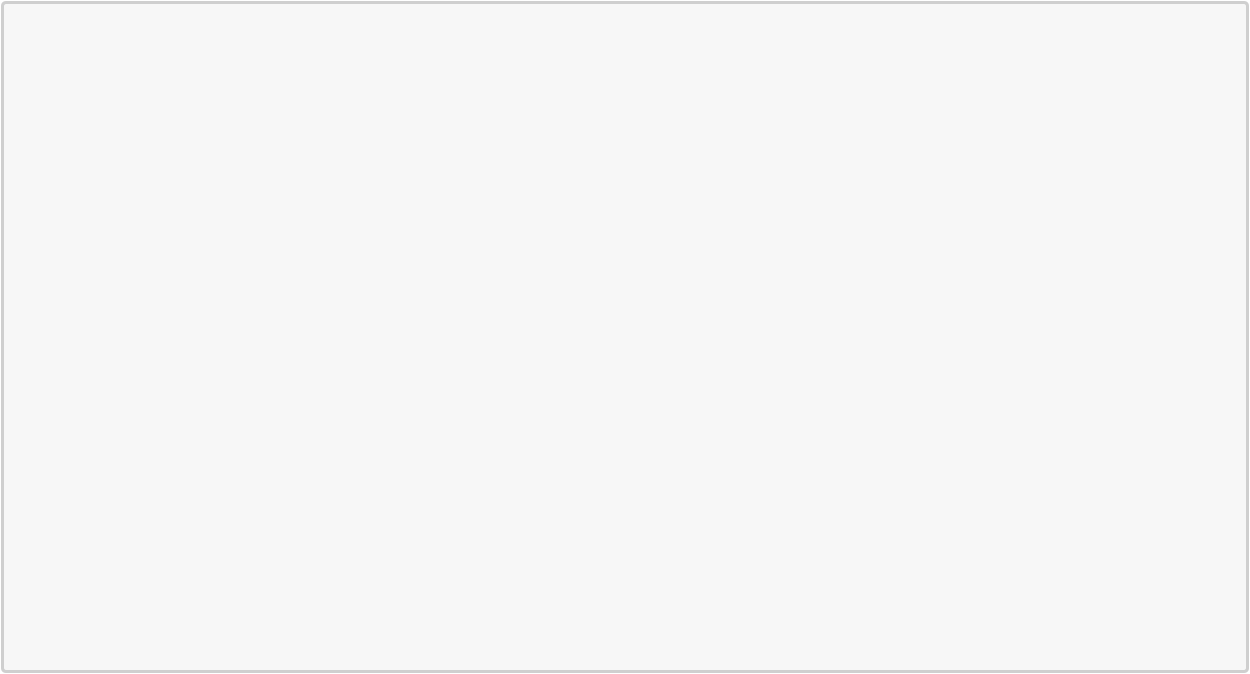
*# Display the contents of `text`*

print(text)

username,ip\_address,time,date tshah,192.168.92.147,15:26:08,2022-05-10 dtanaka,192.168.98.221,9:45:18,2022-05-09 tmitchel,192.168.110.131,14:13:41,2022-05-11 daquino,192.168.168.144,7:02:35,2022-05-08 eraab,192.168.170.243,1:45:14,2022-05-11 jlansky,192.168.238.42,1:07:11,2022-05-11 acook,192.168.52.90,9:56:48,2022-05-10 asundara,192.168.58.217,23:17:52,2022-05-12 jclark,192.168.214.49,20:49:00,2022-05-10 cjackson,192.168.247.153,19:36:42,2022-05-12 jclark,192.168.197.247,14:11:04,2022-05-12 apatel,192.168.46.207,17:39:42,2022-05-10 mabadi,192.168.96.244,10:24:43,2022-05-12 iuduike,192.168.131.147,17:50:00,2022-05-11 abellmas,192.168.60.111,13:37:05,2022-05-10 gesparza,192.168.148.80,6:30:14,2022-05-11 cgriffin,192.168.4.157,23:04:05,2022-05-09 alevitsk,192.168.210.228,8:10:43,2022-05-08 eraab,192.168.24.12,11:29:27,2022-05-11 jsoto,192.168.25.60,5:09:21,2022-05-09

#### ****2. Parsing Logs with Python****

* Applied the .split() method to convert the log data into a list, where each line became a separate element.
* This made it easier to analyze login activity line by line instead of working with one long string.



[2]: *# Assign `import\_file` to the name of the text file that contains the security*␣ *,→log file*

import\_file = "data/login.txt"

* *The`with` statement*
* *Use `open()` to import security log file and store it as a string*

**with** open(import\_file, "r") **as** file:

* *Use `.read()` to read the imported file and store the result in a variable*␣ *,→named `text`*

text = file.read()

*# Display the contents of `text` split into separate lines*

print(text.split())

['username,ip\_address,time,date',

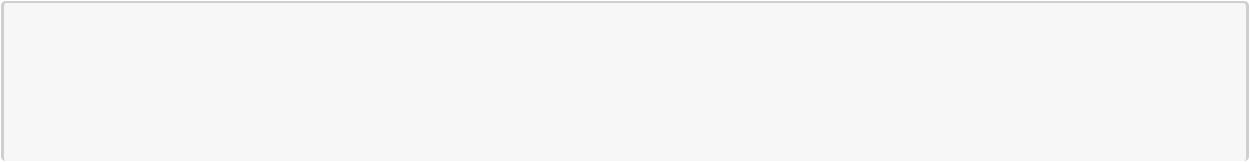
'tshah,192.168.92.147,15:26:08,2022-05-10', 'dtanaka,192.168.98.221,9:45:18,2022-05-09', 'tmitchel,192.168.110.131,14:13:41,2022-05-11', 'daquino,192.168.168.144,7:02:35,2022-05-08', 'eraab,192.168.170.243,1:45:14,2022-05-11', 'jlansky,192.168.238.42,1:07:11,2022-05-11',

'acook,192.168.52.90,9:56:48,2022-05-10', 'asundara,192.168.58.217,23:17:52,2022-05-12', 'jclark,192.168.214.49,20:49:00,2022-05-10','cjackson,192.168.247.153,19:36:42,2022-05-12', 'jclark,192.168.197.247,14:11:04,2022-05-12', 'apatel,192.168.46.207,17:39:42,2022-05-10', 'mabadi,192.168.96.244,10:24:43,2022-05-12', 'iuduike,192.168.131.147,17:50:00,2022-05-11', 'abellmas,192.168.60.111,13:37:05,2022-05-10', 'gesparza,192.168.148.80,6:30:14,2022-05-11', 'cgriffin,192.168.4.157,23:04:05,2022-05-09', 'alevitsk,192.168.210.228,8:10:43,2022-05-08', 'eraab,192.168.24.12,11:29:27,2022-05-11',

'jsoto,192.168.25.60,5:09:21,2022-05-09']

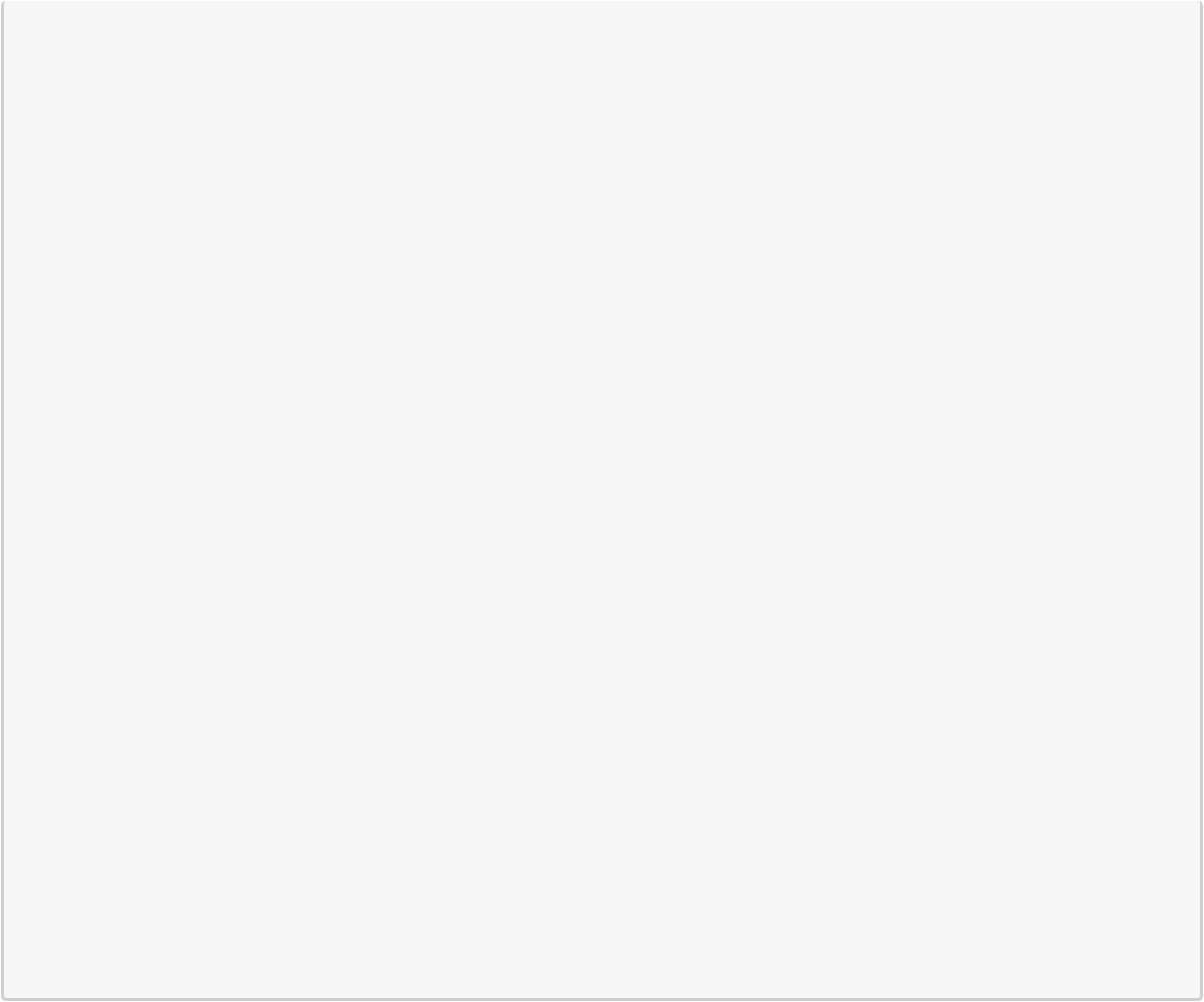
#### ****3. Appending Missing Entries****

* Detected a missing log entry and appended it back into the file using .write() in append ("a") mode.
* Verified that the new entry appeared correctly at the end of the log file.



[3]: *# Assign `import\_file` to the name of the text file that contains the security*␣ *,→log file*

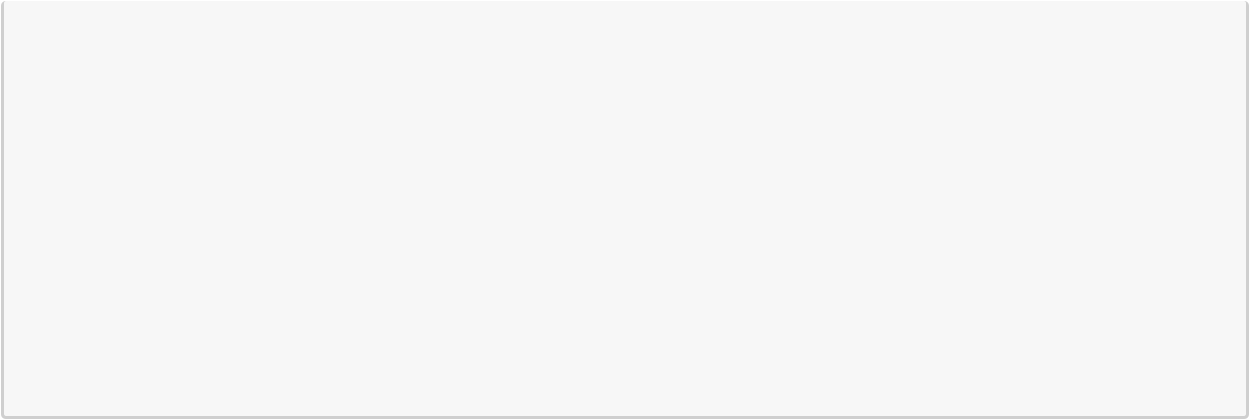
import\_file = "data/login.txt"



*# Assign `missing entry` to a log that was not recorded in the log file*

missing\_entry = "jrafael,192.168.243.140,4:56:27,2022-05-09"

* *Use `open()` to import security log file and store it as a string*



* *Pass in "a" as the second parameter to indicate that the file is being opened*␣ *,→for appending purposes*

**with** open(import\_file, "a") **as** file:

*# Use `.write()` to append `missing\_entry` to the log file*

file.write(missing\_entry)

* *Use `open()` with the parameter "r" to open the security log file for reading*␣ *,→purposes*

**with** open(import\_file, "r") **as** file:

* *Use `.read()` to read in the contents of the log file and store in a*␣ *,→variable named `text`*

text = file.read()

*# Display the contents of `text`*

print(text)

username,ip\_address,time,date tshah,192.168.92.147,15:26:08,2022-05-10 dtanaka,192.168.98.221,9:45:18,2022-05-09 tmitchel,192.168.110.131,14:13:41,2022-05-11 daquino,192.168.168.144,7:02:35,2022-05-08 eraab,192.168.170.243,1:45:14,2022-05-11 jlansky,192.168.238.42,1:07:11,2022-05-11 acook,192.168.52.90,9:56:48,2022-05-10 asundara,192.168.58.217,23:17:52,2022-05-12 jclark,192.168.214.49,20:49:00,2022-05-10 cjackson,192.168.247.153,19:36:42,2022-05-12 jclark,192.168.197.247,14:11:04,2022-05-12 apatel,192.168.46.207,17:39:42,2022-05-10 mabadi,192.168.96.244,10:24:43,2022-05-12 iuduike,192.168.131.147,17:50:00,2022-05-11 abellmas,192.168.60.111,13:37:05,2022-05-10 gesparza,192.168.148.80,6:30:14,2022-05-11 cgriffin,192.168.4.157,23:04:05,2022-05-09alevitsk,192.168.210.228,8:10:43,2022-05-08

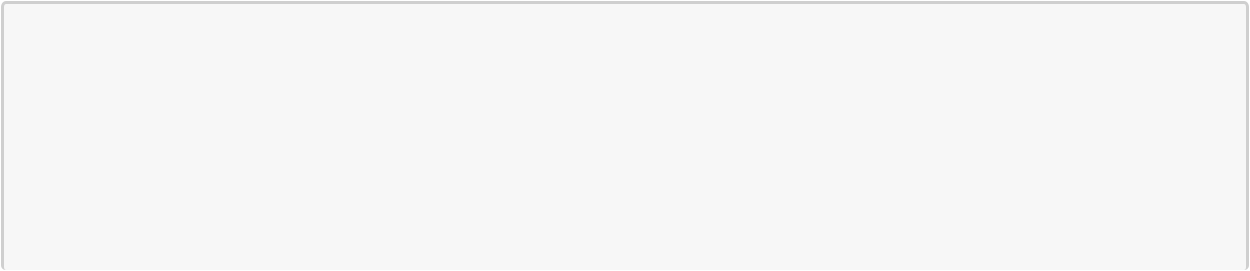
eraab,192.168.24.12,11:29:27,2022-05-11

jsoto,192.168.25.60,5:09:21,2022-05-09

jrafael,192.168.243.140,4:56:27,2022-05-09

#### ****4. Creating an Allow List of IP Addresses****

* Created a new file (allow\_list.txt) to document IP addresses permitted to access restricted resources.
* Used Python’s "w" mode with .write() to store the list of approved IPs.
* Reopened the file in "r" mode to confirm that the allow-list was correctly written and saved.

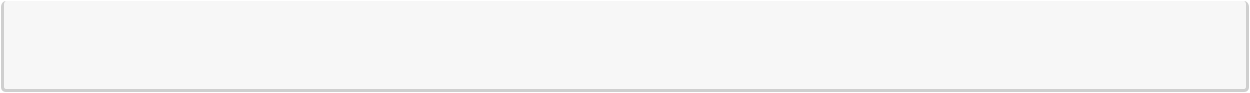


[4]: *# Assign `import\_file` to the name of the text file that you want to create*

import\_file = "data/allow\_list.txt"

* *Assign `ip\_addresses` to a list of IP addresses that are allowed to access*␣ *,→the restricted information*

ip\_addresses = "192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13␣ *,→*192.168.60.153 192.168.96.200 192.168.247.153 192.168.3.252 192.168.116.187␣ *,→*192.168.15.110 192.168.39.246"



*# Display `import\_file`*

print(import\_file)

*# Display `ip\_addresses`*

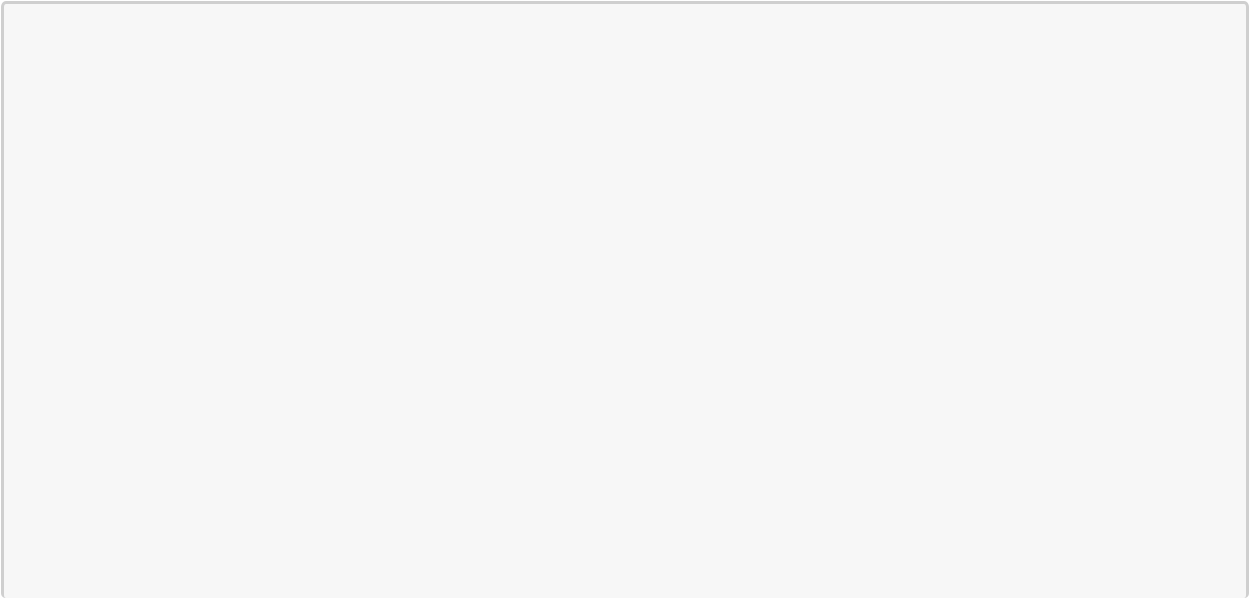
print(ip\_addresses)

data/allow\_list.txt

192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153

192.168.96.200 192.168.247.153 192.168.3.252 192.168.116.187 192.168.15.110

192.168.39.246



[5]: *# Assign `import\_file` to the name of the text file that you want to create*

import\_file = "data/allow\_list.txt"

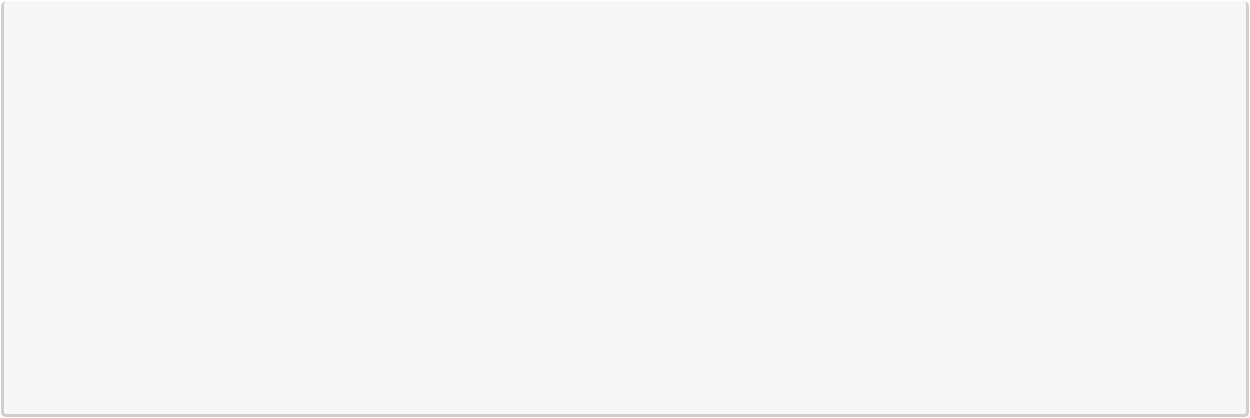
* *Assign `ip\_addresses` to a list of IP addresses that are allowed to access*␣ *,→the restricted information*

ip\_addresses = "192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13␣ *,→*192.168.60.153 192.168.96.200 192.168.247.153 192.168.3.252 192.168.116.187␣ *,→*192.168.15.110 192.168.39.246"

*# Create a `with` statement to write to the text file*

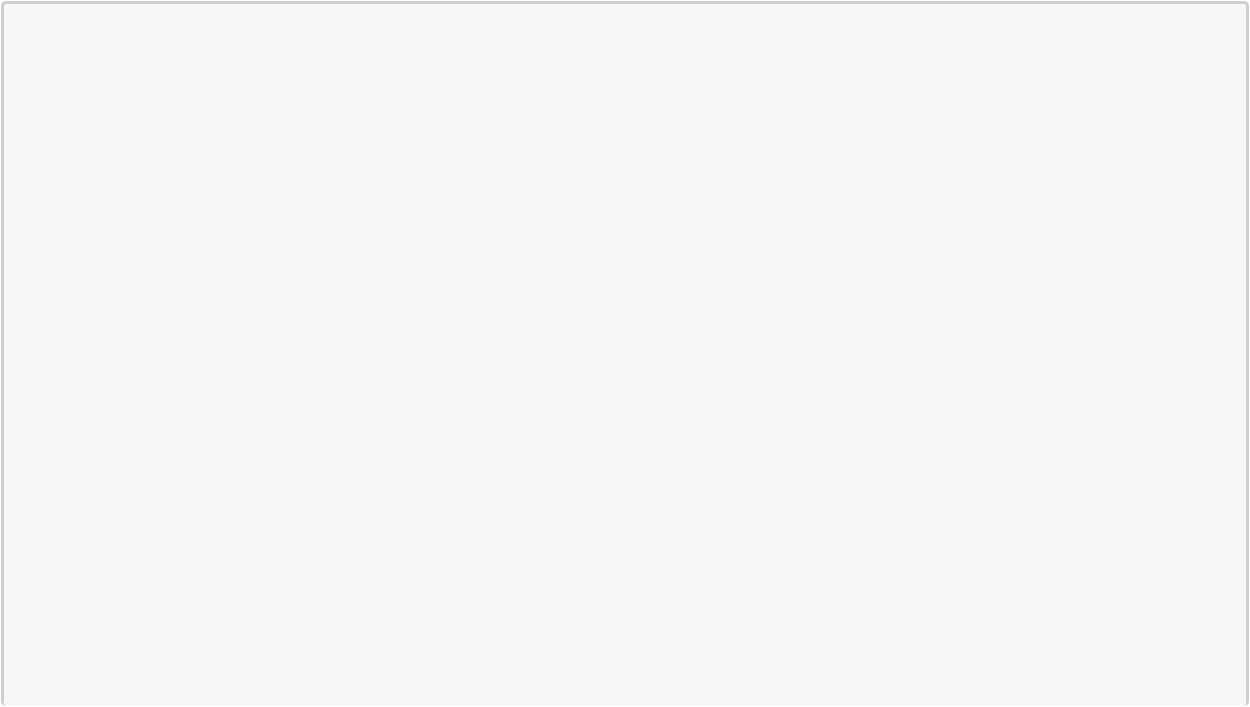
**with** open(import\_file, "w") **as** file:

*# Write `ip\_addresses` to the text file*



file.write(ip\_addresses)

[6]: *# Assign `import\_file` to the name of the text file that you want to create*



import\_file = "data/allow\_list.txt"

* *Assign `ip\_addresses` to a list of IP addresses that are allowed to access*␣ *,→the restricted information*

ip\_addresses = "192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13␣ *,→*192.168.60.153 192.168.96.200 192.168.247.153 192.168.3.252 192.168.116.187␣ *,→*192.168.15.110 192.168.39.246"

*# Create a `with` statement to write to the text file*

**with** open(import\_file, "w") **as** file:

*# Write `ip\_addresses` to the text file*

file.write(ip\_addresses)

*# Create a `with` statement to read in the text file*

**with** open(import\_file, "r") **as** file:

*# Read the file and store the result in a variable named `text`*

text = file.read()

*# Display the contents of `text`*

print(text)

192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153

192.168.96.200 192.168.247.153 192.168.3.252 192.168.116.187 192.168.15.110

192.168.39.246

### ****Outcome & Skills Gained****

This project gave me hands-on experience in:

* File handling with Python (read, write, append)
* Parsing raw log data into structured formats
* Detecting and correcting missing entries in security logs
* Creating allow-lists to support access control policies

Through this activity, I strengthened my ability to **automate security tasks with Python**, a critical skill for incident response and log analysis.

# **Project 4: Suricata IDS Rule Creation & Log Analysis**

**Project Overview**

This lab involved configuring and testing custom Intrusion Detection System (IDS) rules using **Suricata**, an open-source threat detection engine. The goal was to simulate real-world network monitoring and analyze traffic for potential threats using packet capture files and structured logs.

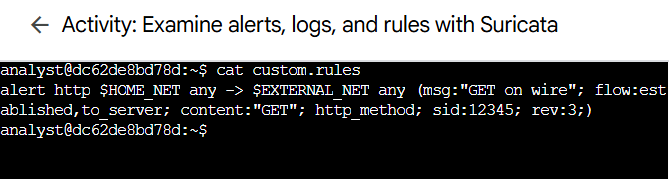
**Objectives**

* Develop and deploy custom Suricata rules
* Simulate network traffic using .pcap files
* Analyze Suricata logs (fast.log, eve.json) for triggered alerts
* Correlate events using flow identifiers and timestamps

**Task 1: Examine a custom rule in Suricata**

**Rule Creation**

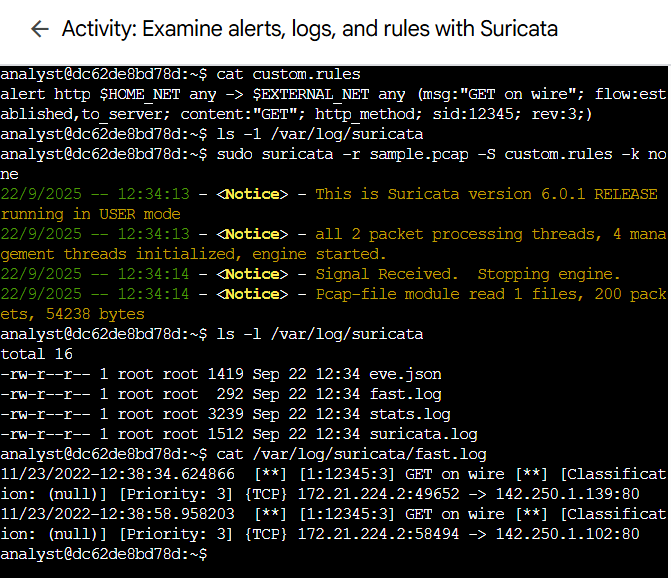
* Wrote a custom rule to detect HTTP GET requests from $HOME\_NET to $EXTERNAL\_NET
* Used Suricata rule syntax including alert, msg, flow, content, sid, and rev
* Integrated rules into Suricata’s configuration via custom.rules
* Use the cat command to display the rule in the custom.rules file: cat custom.rules

The command returns the rule as the output in the shell:

### Task 2. Rule Execution & Log Analysis

* Ran Suricata against a sample .pcap file to simulate network activity
* Verified rule effectiveness by checking fast.log for alert entries

Before running Suricata, I verified the contents of the default log directory:

ls -l /var/log/suricata

Observation: The directory was empty, confirming no prior logs existed.

I executed Suricata using a sample packet capture file and a custom rule set:

sudo suricata -r sample.pcap -S custom.rules -k none

Explanation:

* -r sample.pcap: Loads the packet capture file to simulate network traffic.
* -S custom.rules: Applies my custom detection rules.
* -k none: Disables checksum validation, which is unnecessary for static .pcap files.

Note: sudo was required to access packet-level data during this lab, though it may not be needed in production environments.

Result: Suricata processed the packets and generated alerts based on rule matches.

After running Suricata, I listed the contents of the log directory again:

ls -l /var/log/suricata

Observation: Four new files were created, including:

* fast.log: Contains human-readable alert entries.
* eve.json: A structured JSON log for deeper analysis.

To inspect triggered alerts, I used the cat command:

cat /var/log/suricata/fast.log

Output: Alert entries were successfully logged, confirming that my rule was triggered by the simulated traffic.

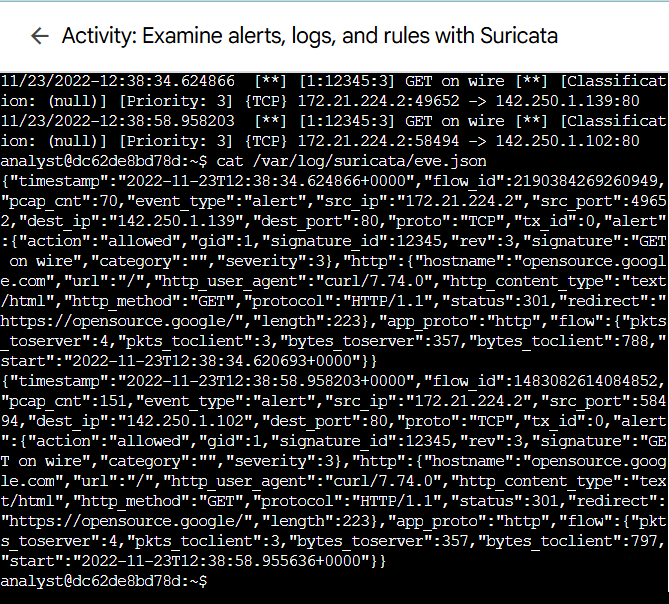
### Task 3. Log Analysis

* Parsed eve.json using jq to extract key fields:
  + timestamp, flow\_id, alert.signature, proto, dest\_ip
* Used flow\_id to correlate packets and reconstruct traffic flows
* Identified patterns and validated rule accuracy

#### Step 1: Viewing Raw Log Output

To begin, I used the cat command to display the contents of eve.json:

cat /var/log/suricata/eve.json

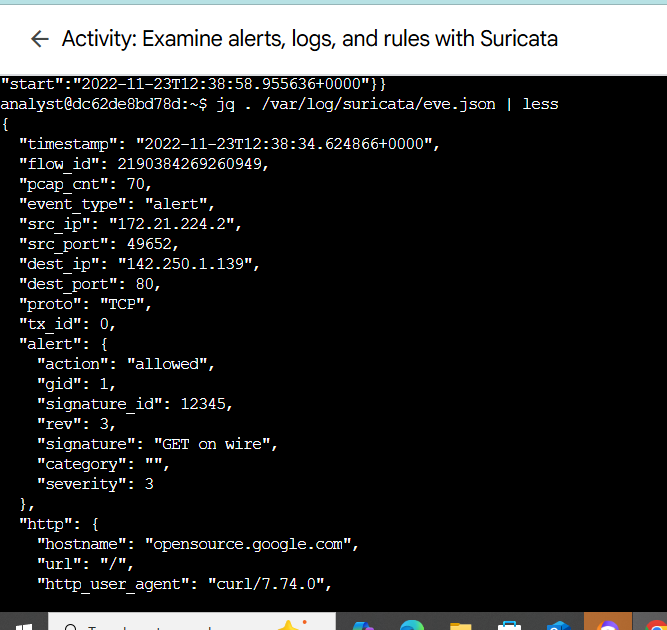


Observation: The output was extensive and difficult to interpret due to its raw JSON structure.

#### Step 2: Formatting with jq

To improve readability, I piped the output through the jq tool:

jq . /var/log/suricata/eve.json | less



Result: The formatted output was much easier to navigate. I used less to scroll through the entries and pressed Q to exit back to the terminal.

Note: jq is a powerful command-line tool for parsing and filtering JSON data essential for log analysis in cybersecurity workflows.

#### Step 3: Extracting Key Event Fields

To focus on specific threat indicators, I extracted select fields from each log entry:

jq -c "[.timestamp,.flow\_id,.alert.signature,.proto,.dest\_ip]" /var/log/suricata/eve.json

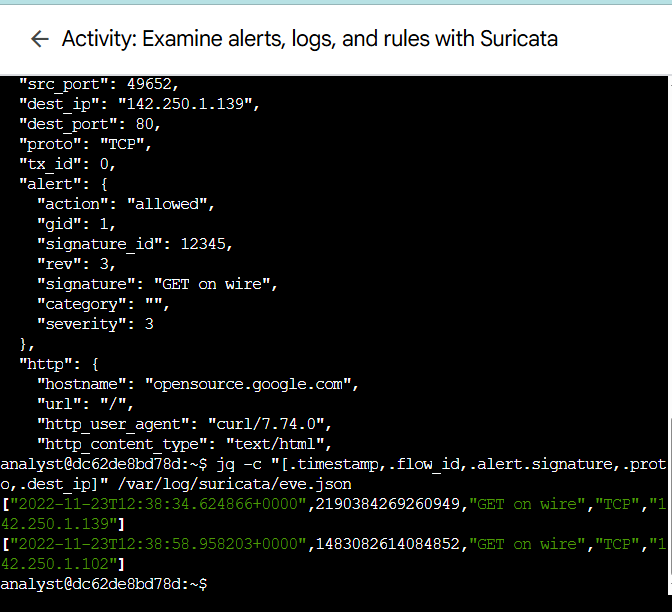
Fields Extracted:

* timestamp: When the event occurred
* flow\_id: Unique identifier for the network flow
* alert.signature: Description of the triggered rule
* proto: Protocol used (e.g., TCP)
* dest\_ip: Destination IP address

Sample Output:

["2022-11-23T12:38:34.624866+0000",14500150016149,"GET on wire","TCP","142.250.1.139"]

["2022-11-23T12:38:58.958203+0000",1647223379236084,"GET on wire","TCP","142.250.1.102"]



### Correlating Events by flow\_id

To analyze all logs related to a specific network flow, I filtered entries using a known flow\_id:

jq "select(.flow\_id==14500150016149)" /var/log/suricata/eve.json

Insight: Suricata assigns a unique flow\_id to each sequence of packets between a source and destination. This allows for precise event correlation and threat tracing across multiple log entries.

### Skills Demonstrated

|  |  |
| --- | --- |
| **Skill** | **Description** |
| IDS Rule Writing | Created precise detection logic using Suricata syntax |
| Log Analysis | Parsed JSON logs to extract and correlate threat data |
| CLI Proficiency | Used tools like jq, cat, and less for analysis |
| Network Security | Applied concepts of internal/external network boundaries |

### Key Takeaways

* Gained hands-on experience with Suricata’s rule engine and logging system
* Learned to simulate and analyze network traffic for threat detection
* Strengthened my ability to interpret IDS alerts and correlate events

# **Project 5: Update a file through a Python algorithm**

### Project description

At an organization, access to restricted content is controlled with an allow list of IP addresses. The "allow\_list.txt" file identifies these IP addresses. A separate remove list identifies IP addresses that should no longer have access to this content. I created an algorithm to automate updating the "allow\_list.txt" file and remove these IP addresses that should no longer have access.

### Open the file that contains the allow list

## For the first part of the algorithm, I opened the "allow\_list.txt" file. First, I assigned this file name as a string to the import\_file variable:



Then, I used a with statement to open the file:



In my algorithm, the with statement is used with the .open() function in read mode to open the allow list file for the purpose of reading it. The purpose of opening the file is to allow me to access the IP addresses stored in the allow list file. The with keyword will help manage the resources by closing the file after exiting the with statement. In the code with open(import\_file, "r") as file:, the open() function has two parameters. The first identifies the file to import, and then the second indicates what I want to do with the file. In this case, "r" indicates that I want to read it. The code also uses the as keyword to assign a variable named file; file stores the output of the .open() function while I work within the with statement.

### Read the file contents

In order to read the file contents, I used the .read() method to convert it into the string.



When using an .open() function that includes the argument "r" for “read,” I can call the .read() function in the body of the with statement. The .read() method converts the file into a string and allows me to read it. I applied the .read() method to the file variable identified in the with statement. Then, I assigned the string output of this method to the variable ip\_addresses.

In summary, this code reads the contents of the "allow\_list.txt" file into a string format that allows me to later use the string to organize and extract data in my Python program.

### Convert the string into a list

In order to remove individual IP addresses from the allow list, I needed it to be in list format. Therefore, I next used the .split() method to convert the ip\_addresses string into a list:



The .split() function is called by appending it to a string variable. It works by converting the contents of a string to a list. The purpose of splitting ip\_addresses into a list is to make it easier to remove IP addresses from the allow list. By default, the .split() function splits the text by whitespace into list elements. In this algorithm, the .split() function takes the data stored in the variable ip\_addresses, which is a string of IP addresses that are each separated by a whitespace, and it converts this string into a list of IP addresses. To store this list, I reassigned it back to the variable ip\_addresses.

### Iterate through the remove list

A key part of my algorithm involves iterating through the IP addresses that are elements in the remove\_list. To do this, I incorporated a for loop:



The for loop in Python repeats code for a specified sequence. The overall purpose of the for loop in a Python algorithm like this is to apply specific code statements to all elements in a sequence. The for keyword starts the for loop. It is followed by the loop variable element and the keyword in. The keyword in indicates to iterate through the sequence ip\_addresses and assign each value to the loop variable element.

### Remove IP addresses that are on the remove list

My algorithm requires removing any IP address from the allow list, ip\_addresses, that is also contained in remove\_list. Because there were not any duplicates in ip\_addresses, I was able to use the following code to do this:



First, within my for loop, I created a conditional that evaluated whether or not the loop variable element was found in the ip\_addresses list. I did this because applying .remove() to elements that were not found in ip\_addresses would result in an error.

Then, within that conditional, I applied .remove() to ip\_addresses. I passed in the loop variable element as the argument so that each IP address that was in the remove\_list would be removed from ip\_addresses.

### Update the file with the revised list of IP addresses

As a final step in my algorithm, I needed to update the allow list file with the revised list of IP addresses. To do so, I first needed to convert the list back into a string. I used the .join() method for this:



The .join() method combines all items in an iterable into a string. The .join() method is applied to a string containing characters that will separate the elements in the iterable once joined into a string. In this algorithm, I used the .join() method to create a string from the list ip\_addresses so that I could pass it in as an argument to the .write() method when writing to the file "allow\_list.txt". I used the string ("\n") as the separator to instruct Python to place each element on a new line.

Then, I used another with statement and the .write() method to update the file:



This time, I used a second argument of "w" with the open() function in my with statement. This argument indicates that I want to open a file to write over its contents. When using this argument "w", I can call the .write() function in the body of the with statement. The .write() function writes string data to a specified file and replaces any existing file content.

In this case I wanted to write the updated allow list as a string to the file "allow\_list.txt". This way, the restricted content will no longer be accessible to any IP addresses that were removed from the allow list. To rewrite the file, I appended the .write() function to the file object file that I identified in the with statement. I passed in the ip\_addresses variable as the argument to specify that the contents of the file specified in the with statement should be replaced with the data in this variable.

### Summary

I created an algorithm that removes IP addresses identified in a remove\_list variable from the "allow\_list.txt" file of approved IP addresses. This algorithm involved opening the file, converting it to a string to be read, and then converting this string to a list stored in the variable ip\_addresses. I then iterated through the IP addresses in remove\_list. With each iteration, I evaluated if the element was part of the ip\_addresses list. If it was, I applied the .remove() method to it to remove the element from ip\_addresses.. After this, I used the .join() method to convert the ip\_addresses back into a string so that I could write over the contents of the "allow\_list.txt" file with the revised list of IP addresses.

# **Project 6: Vulnerability Assessment Report**

### Vulnerability Assessment Project – E-Commerce Database Server

### Overview

As part of my cybersecurity training, I conducted a vulnerability assessment for a simulated e-commerce company. The project focused on evaluating risks associated with a publicly accessible database server and recommending remediation strategies to improve security. This exercise allowed me to practice applying NIST SP 800-30 guidelines and develop the skills required to communicate technical risks to business decision-makers.

### System Description

* **Infrastructure:** Linux server with MySQL database
* **Resources:** High-performance CPU, 128GB RAM
* **Connections:** IPv4 networking, SSL/TLS encryption enabled
* **Use case:** Stores customer, campaign, and analytics data for global remote teams

### Scope

The assessment was limited to **access controls** and the risks posed by leaving the database server open to the public. The review period was set to three months, and NIST SP 800-30 Rev. 1 guided the analysis.

### Purpose

The database server is a centralized computer system that stores and manages large amounts of data. The server is used to store customer, campaign, and analytic data that can later be analyzed to track performance and personalize marketing efforts. It is critical to secure the system because of its regular use for marketing operations.

### Risk Assessment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threat source** | **Threat event** | **Likelihood** | **Severity** | **Risk** |
| *Hacker* | *Obtain sensitive information via exfiltration* | *3* | *3* | *9* |
| *Employee* | *Disrupt mission-critical operations* | *2* | *3* | *6* |
| *Customer* | *Alter/Delete critical information* | *1* | *3* | *3* |

**Key Finding:** The highest risk came from external hackers exploiting the open public access to the database.

### Approach

Risks that were measured considered the data storage and management procedures of the business. Potential threat sources and events were determined using the likelihood of a security incident given the open access permissions of the information system. The severity of potential incidents were weighed against the impact on day-to-day operational needs.

### Remediation Strategy

To mitigate risks, I recommended:

* Enforcing **authentication, authorization, and auditing (AAA)** controls
* Using **role-based access control (RBAC)** and **multi-factor authentication**
* Migrating from **SSL to TLS** for secure communication
* Implementing **IP allow-listing** to restrict access only to corporate networks

### Key Skills Demonstrated

* Vulnerability assessment using structured methodology (NIST SP 800-30)
* Risk analysis and scoring (likelihood × severity)
* Development of remediation strategies (technical and procedural)
* Clear technical documentation for non-technical decision makers

# **Project 7: Packet Capture and Analysis with Wireshark and Tcpdump**

### Overview

As a cybersecurity analyst trainee, I carried out hands-on exercises using **Wireshark** and **tcpdump** to analyze and capture network traffic. These activities helped me build skills in packet inspection, protocol analysis, and filtering network traffic to identify relevant information.

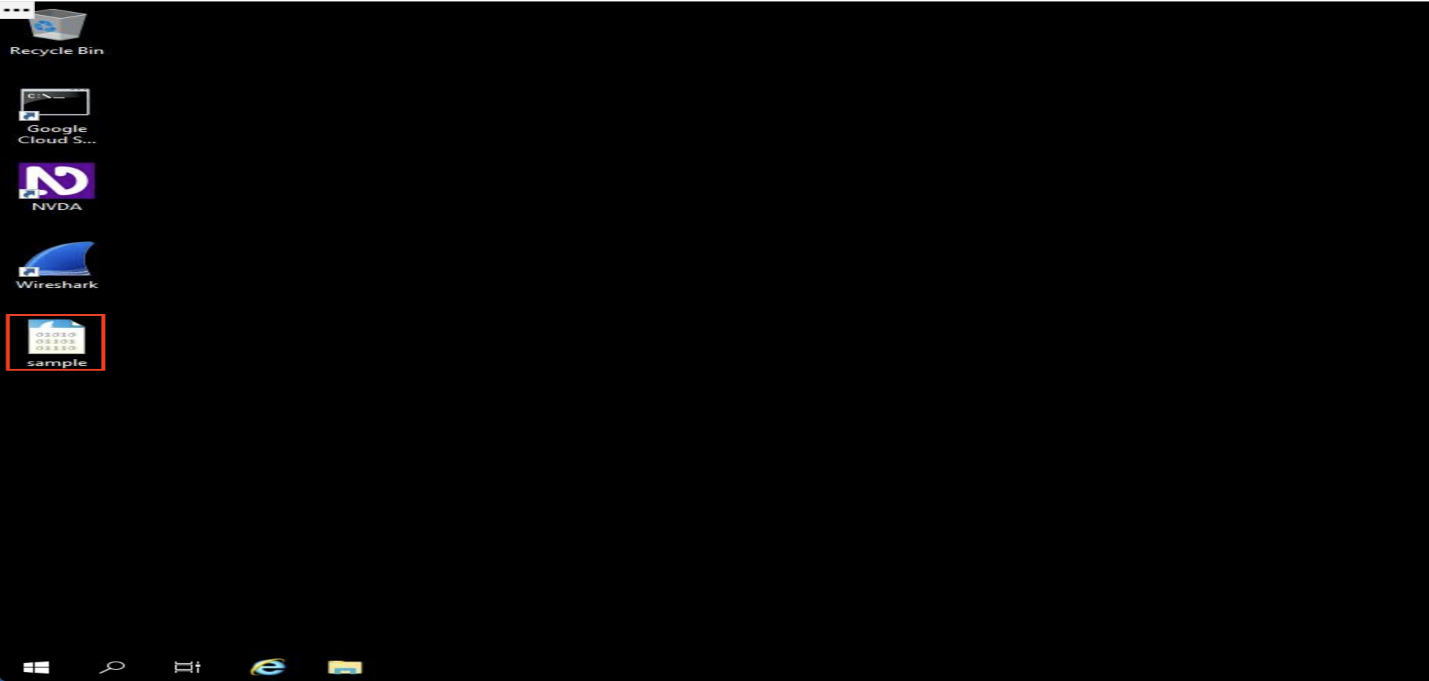
### ****Tools & Technologies****

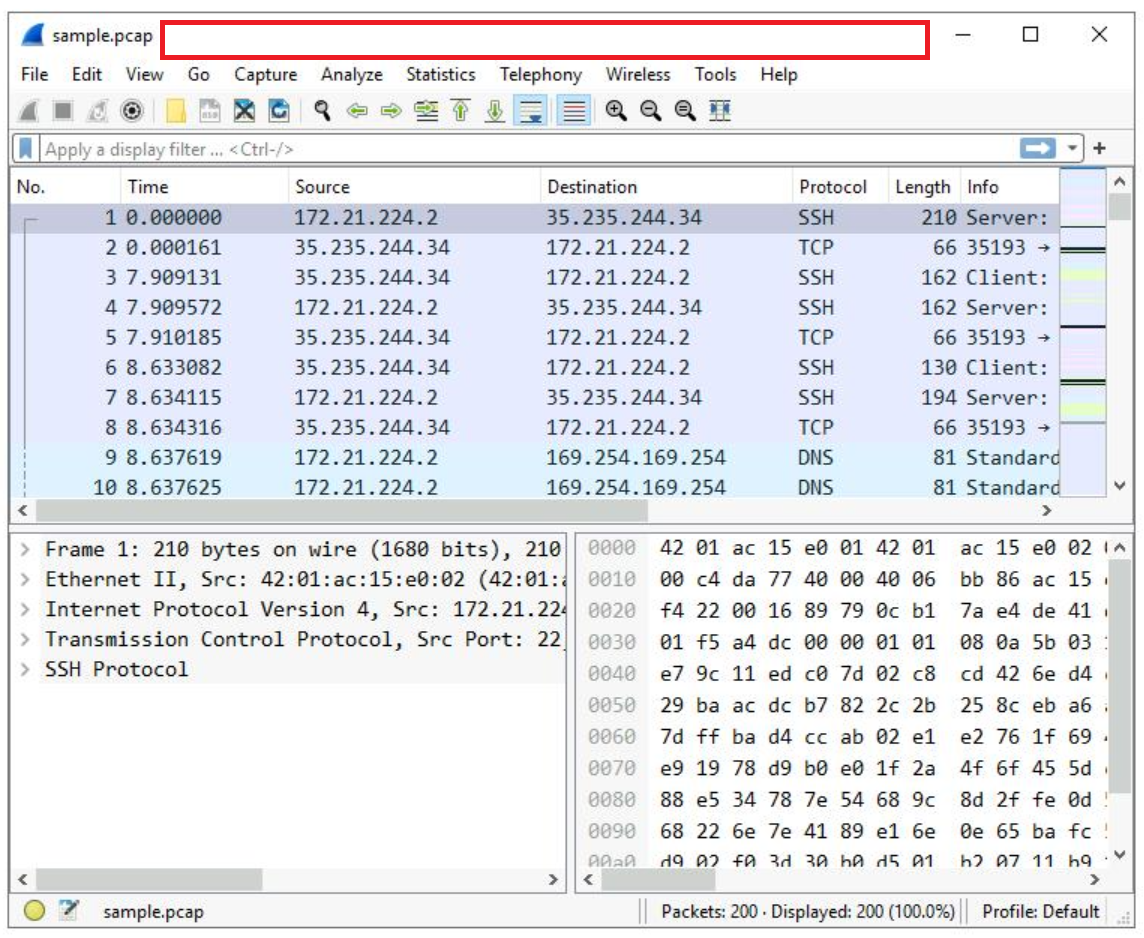
* **Wireshark** – GUI-based packet analyzer for detailed inspection of network traffic
* **tcpdump** – Command-line tool for live packet capture and filtering in Linux
* **PCAP files** – For storing and reviewing captured traffic

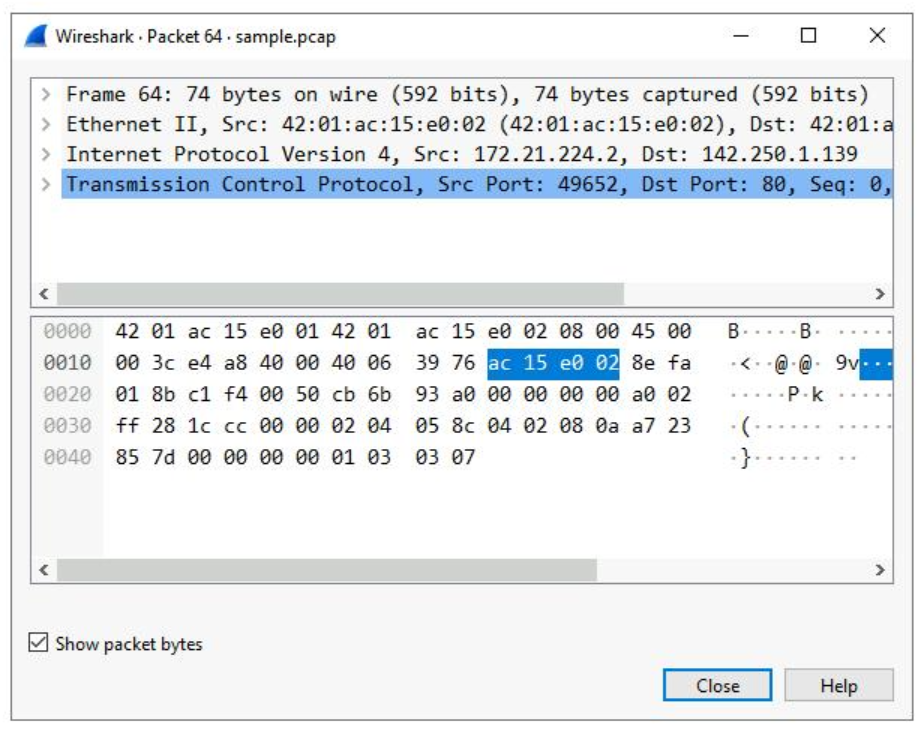
### ****Key Activities & Screenshots****

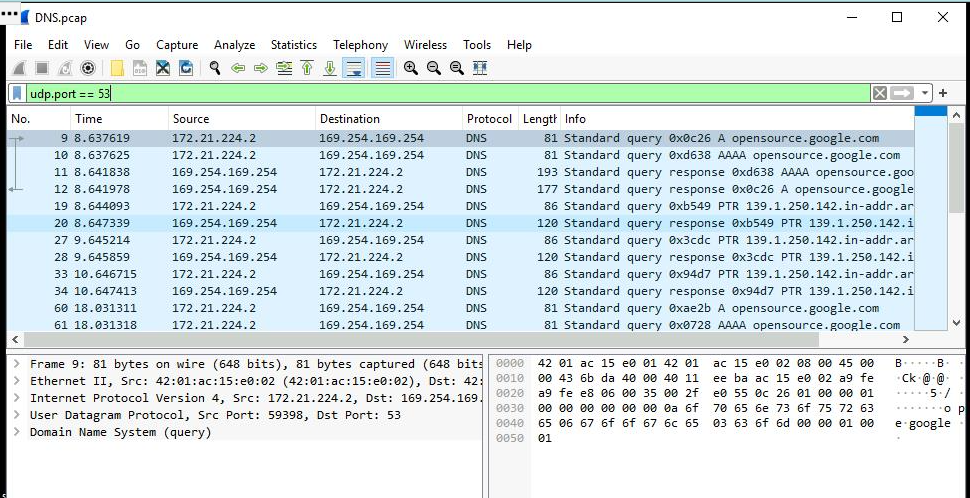
#### ****Analyzing Packets with Wireshark****

* Opened and explored a .pcap file in Wireshark.
* Examined packet details including frame length, source and destination IP addresses, and protocols.
* Applied filters such as:
  + ip.addr == <IP> (filter by IP address)
  + udp.port == 53 (filter DNS traffic)
  + tcp.port == 80 (filter HTTP traffic)
  + tcp contains "curl" (search payload text).
* Identified communication protocols (e.g., ICMP, TCP, UDP, DNS).
* Verified DNS queries and responses (e.g., resolving opensource.google.com to IP 142.250.1.139).
* Inspected TCP headers to analyze flags, sequence numbers, and ports.

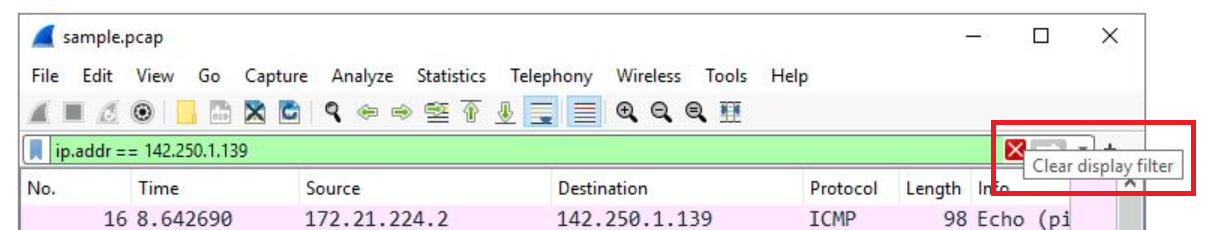
Fig. 1. VM window



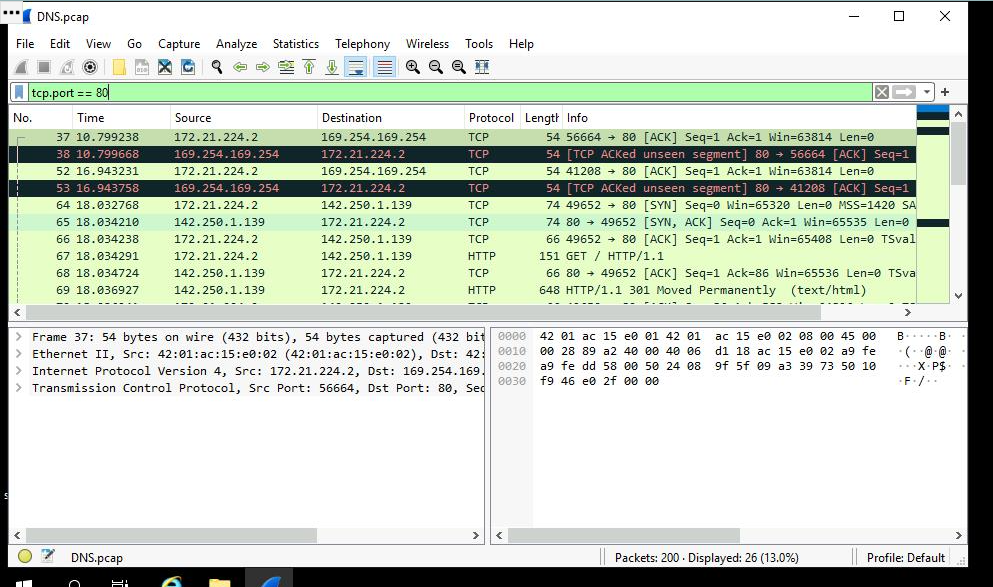
**Figure 2**  
Wireshark filter applied to isolate HTTP traffic on TCP port 80

****

**Figure 3**  
Caption: DNS query and response showing resolution of opensource.google.com to its IP address.



* + **Figure 4**  
    Caption: ICMP echo request and reply packets captured to demonstrate basic connectivity.

****

* + **Figure 5**  
    Caption: Detailed breakdown of a TCP packet showing Ethernet, IP, and TCP headers.

#### ****Capturing Packets with Tcpdump (Linux Environment)****

* Used ifconfig and tcpdump -D to identify available network interfaces.
* Ran live captures using:
  + sudo tcpdump -i eth0 -v -c5 (capture 5 packets with verbose details).
  + sudo tcpdump -i eth0 -nn -c9 port 80 -w capture.pcap (save 9 packets on port 80 to file).
* Generated HTTP traffic using curl opensource.google.com for testing.
* Inspected saved packet capture using:
  + sudo tcpdump -nn -r capture.pcap -v (verbose packet details).
  + sudo tcpdump -nn -r capture.pcap -X (hexadecimal and ASCII output for deeper inspection).
* Observed IP headers, TCP flags, and payload data for anomalies and communication patterns.
* eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
* inet 172.17.0.2 netmask 255.255.0.0 broadcast 172.17.255.255
* ether 02:42:ac:11:00:02 txqueuelen 0 (Ethernet)
* RX packets 784 bytes9379957 (8.9 MiB)
* RX errors 0 dropped 0 overruns 0 frame 0
* TX packets 683 bytes 56880 (55.5 KiB)
* TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
* lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
* inet 127.0.0.1 netmask 255.0.0.0
* loop txqueuelen 1000 (Local Loopback)
* RX packets 400 bytes 42122 (041.1 KiB)
* RX errors 0 dropped 0 overruns 0 frame 0
* TX packets 400 bytes 42122 (041.1 KiB)
* TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

**Fig. 6**  
tcpdump command-line identifying network interface

tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

10:57:33.427749 IP (tos 0x0, ttl 64, id 35057, offset 0, flags [DF], protot TCP (6), length 134)

7acb26dc1f44.5000 > nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.59788: Flags [P.], cksum 0x5851 (incorrect > 0x30d3), seq 1080713945:1080714027, ack 62760789, win 501, options [nop,nop,TS val 1017464119 ecr 3001513453], length 82

10:57:33.427954 IP (tos 0x0, ttl 64, id 21812, offset 0, flags [DF], protot TCP (6), length 52)

nginx-us-east1-c.c.qwiklabs-terminal-vms-prod-00.internal.59788 > 7acb26dc1f44.5000: Flags [.], cksum 0x9122 (correct), ack 82, win 510, options [nop,nop,TS val 3001513453 ecr 1017464119], length 0

2 packets captured

4 packets received by filter

0 packets dropped by kernel

Fig. 7. **Inspect the network traffic of a network interface with tcpdump**

reading from file capture.pcap, link-type EN10MB (Ethernet)

20:53:27.669101 IP (tos 0x0, ttl 64, id 50874, offset 0, flags [DF], proto TCP (6), length 60)

172.17.0.2:46498 > 146.75.38.132:80: Flags [S], cksum 0x5445 (incorrect), seq 4197622953, win 65320, options [mss 1420,sackOK,TS val 610940466 ecr 0, nop,wscale 7], length 0

20:53:27.669422 IP (tos 0x0, ttl 62, id 0, offset 0, flags [DF], proto TCP (6), length 60)

146.75.38.132:80: > 172.17.0.2:46498: Flags [S.], cksum 0xc272 (correct), seq 2026312556, ack 4197622953, win 65535, options [mss 1420,sackOK,TS val 155704241 ecr 610940466, nop,wscale 9], length 0

Fig. 8. **Filter the captured packet data.**

### ****Outcome & Skills Gained****

This project strengthened my ability to:

* Capture and interpret live and stored network traffic
* Use filters efficiently to investigate relevant data
* Differentiate between protocols (DNS, TCP, ICMP, etc.)
* Apply packet analysis in the context of **network security monitoring**