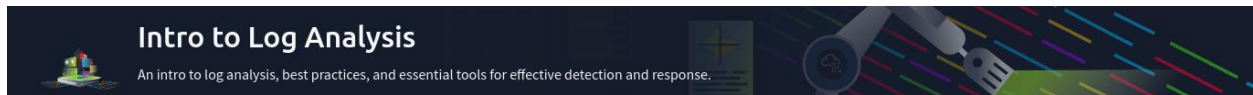


LinkedIn: [Kelvin Kimotho](#)

Intro to Log Analysis TryHackMe Module



A link to my module completion badge <https://tryhackme.com/r/p/Mr.kevin>

Introduction

Logs are records of events generated by devices, applications, and systems that capture information on their activities.

Log Analysis is the process of examining and interpreting log event data to monitor system performance, track user behavior, and identify security incidents or anomalies.

Importance of Log Analysis in Cybersecurity

- Helps **monitor metrics** (e.g., system performance, user activity).
- Identifies **security incidents** and **potential threats**.
- Provides insights into system vulnerabilities and possible breaches.
- Enables **proactive security measures** by analyzing historical data.

Log Analysis Process

1. **Collection.** We gather log data from various sources.
2. **Parsing.** This involves extracting and organizing data to make it usable.
3. **Processing.** This is analyzing the data to derive actionable insights.

Key Benefits of Effective Log Analysis

- **Timely Detection.** Helps security teams identify incidents early.
- **Incident Response.** Enables faster response to security breaches.
- **Proactive Monitoring.** Offers insights into potential future threats.
- **Compliance.** Ensures adherence to security standards and regulations.

Log Analysis Basics

Each log entry typically includes

- **Timestamp.** Date and time of the event.
- **Source.** System generating the log.
- **Event details.** Specifics about the event, such as its nature, severity, and related activities.

Log Severity Levels (Increasing severity)

- **Informational.** Contains general info about system activities.
- **Warning.** Potential issues or unusual behavior.
- **Error.** Issues that may require attention.
- **Critical.** High-priority issues requiring immediate action.

Why Are Logs Important?

- **System Troubleshooting.** Logs helps IT teams identify and resolve system errors or failures, improving uptime.
- **Cybersecurity Incidents.** They are vital for detecting unauthorized access, malware, or data breaches. Helps security teams respond quickly.
- **Threat Hunting.** They enable proactive searches for advanced threats or suspicious activity that might bypass traditional security defenses.
- **Compliance.** Ensures organizations meet regulatory requirements by maintaining detailed records of system activities (e.g., GDPR, HIPAA, PCI DSS).

Types of Logs

Different system components generate various log types, each serving a distinct purpose

- **Application Logs.** Provide insights into application operations, errors, warnings.
- **Audit Logs.** Record user actions and system changes, creating a history of interactions.
- **Security Logs.** Capture security-related events (e.g., logins, firewall activities).
- **Server Logs.** Provide system operation details, errors, and access logs for servers.

- **System Logs.** Contain kernel activity, boot sequences, hardware status, and system errors.
- **Network Logs.** Record network communications, connections, and data transfers.
- **Database Logs.** Track queries, actions, and changes within database systems.
- **Web Server Logs.** Monitor requests processed by web servers, including IP addresses, URLs, and response codes.

Log Analysis in Cybersecurity

Logs offer a comprehensive view of a system's behavior, helping cybersecurity teams to

- **Detect Security Events.** Recognize signs of a breach or attack.
- **Investigate Incidents.** Gather data to understand the cause and scope of security issues.
- **Monitor Systems.** Continuously track system health and security.

Log Analysis Workflow

- **Collection.** We start by gathering logs from different systems and devices.
- **Parsing.** Organize raw log data into a usable format.
- **Processing/Analysis.** Then we evaluate the log data to uncover trends, patterns, and anomalies for detection, response, and investigation.

Investigation Theory

Timeline Creation

A **Timeline** is a chronological sequence of events derived from logs, essential for understanding the progression of incidents.

- It helps analysts trace the sequence of events leading to an incident, identify the initial compromise, and analyze attacker tactics, techniques, and procedures (TTPs).

Timestamps and Time Zones

- **Timestamps** record when an event occurred in a log.
- **Challenges** with timestamps for example is, Distributed systems generate logs with varying time zones and formats.

- To solve this we should apply consistent time zone conversion (e.g., using UNIX time) ensures accurate log correlation across systems.
- Tools like **Splunk** automatically handle time zone conversion during indexing and searching, making investigations more efficient.

Super Timelines (Consolidated Timelines)

Super Timeline. A unified, chronological representation combining logs from multiple systems, devices, and applications to give a holistic view of events.

Plaso (Python Log2Timeline) is an open-source tool that automates the creation of super timelines, helping analysts understand correlations between different log sources (e.g., system logs, firewall logs, network traffic).

Data Visualization

- Tools like **Kibana** and **Splunk** allow security analysts to convert raw log data into visual representations (graphs, charts) for easier identification of patterns and anomalies.
- **We can for example visualize** failed login attempts over time to detect a brute-force attack.
- **Effective Visualization** requires clear objectives (e.g., monitoring login attempts) and an understanding of the data being collected.

Log Monitoring and Alerting

- **Monitoring. This involves continuous** analysing log data for abnormal activities.
- **Alerting.** Automated notifications for suspicious activities (e.g., multiple failed login attempts, privilege escalation).
- **SIEM Solutions** (e.g., Splunk, Elastic Stack) allow analysts to set up custom alerts based on log metrics, ensuring immediate response to potential threats.
- **Escalation Procedures.** Defined roles and notification processes ensure the right personnel are informed at each stage of the incident response.

External Research and Threat Intelligence

Threat Intelligence. This is information about potential threats (e.g., IP addresses, file hashes, domains) used to identify malicious activities.

- We use it to analyze log files for known malicious indicators such as suspicious IP addresses.
- **For example, a web server log shows an attempt to access an admin panel (e.g., suspicious IP address 54.36.149.64).**

Threat Intelligence Feeds. We can use tools like **ThreatFox** can be used to cross-reference log entries with known malicious indicators (IOCs).

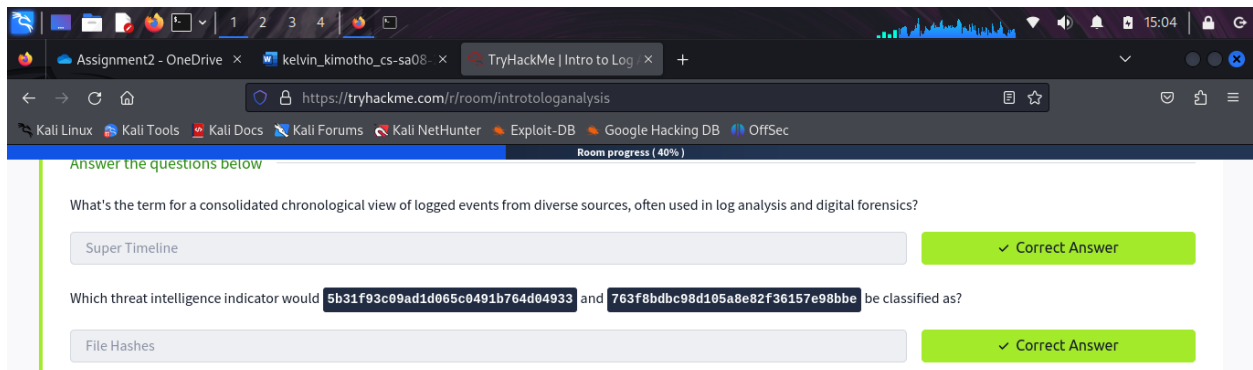
Log File Search with GREP

We can use GREP a command-line tool for searching log files for specific indicators (e.g., IP addresses or file hashes).

- For example, searching logs for a suspicious IP address (e.g., 54.36.149.64) saved in a log file highlighting its presence for further investigation.

Question: What's the term for a consolidated chronological view of logged events from diverse sources, often used in log analysis and digital forensics?

Answer: super timeline



Question: Which threat intelligence indicator would 5b31f93c09ad1d065c0491b764d04933 and 763f8bdb98d105a8e82f36157e98bbe be classified as?

Answer: File Hashes

The provided indicators look similar and both consist of 32 hexadecimal characters, which is characteristic of an output from the MD5 algorithm.

Detection Engineering

Common Log File Locations

Understanding where logs are located is essential for effective log analysis and threat detection. While paths can vary by system configuration, software version, or custom settings, knowing the typical log file locations for various services helps streamline investigations.

Web Servers

- **Nginx:**
 - Access Logs: /var/log/nginx/access.log
 - Error Logs: /var/log/nginx/error.log
- **Apache:**
 - Access Logs: /var/log/apache2/access.log
 - Error Logs: /var/log/apache2/error.log

Databases

- **MySQL:**
 - Error Logs: /var/log/mysql/error.log
- **PostgreSQL:**
 - Error/Activity Logs: /var/log/postgresql/postgresql-{ version }-main.log

Web Applications

- **PHP:**
 - Error Logs: /var/log/php/error.log

Operating Systems

- **Linux:**
 - General System Logs: /var/log/syslog

- Authentication Logs: /var/log/auth.log

Firewalls/IDS/IPS

- **iptables:**
 - Firewall Logs: /var/log/iptables.log
- **Snort:**
 - IDS Logs: /var/log/snort/

Common Log Patterns

Recognizing typical patterns in logs is essential for detecting malicious behavior. Some common log patterns include abnormal user behavior, attack signatures, and system anomalies.

Abnormal User Behavior

- **Multiple Failed Login Attempts.** Could indicate a brute-force attack.
- **Unusual Login Times.** Logins occurring outside normal hours may suggest unauthorized access.
- **Geographic Anomalies.** Logins from unfamiliar or unusual geographic locations (including simultaneous logins from different locations) might indicate account compromise.
- **Frequent Password Changes.** Rapid, repeated changes could suggest an attempt to cover up unauthorized access.
- **Unusual User-Agent Strings.** Strange or unexpected user-agent strings in HTTP logs may point to automated tools or attacks (e.g., Nmap or Hydra).

Detection Tools

- **User Behavior Analytics (UBA).** Tools like **Splunk UBA**, **IBM QRadar UBA**, and **Azure AD Identity Protection** can help detect deviations from established normal behaviors.

Common Attack Signatures

Identifying attack signatures in log data to detect and respond to threats. These signatures are patterns or characteristics left by attackers.

SQL Injection (SQLi)

Exploits vulnerabilities in web applications that interact with databases.

Indicators of sql injection Include,

- Suspicious SQL queries: unexpected characters (e.g., single quotes, comments), UNION statements, or time delays.
- Example pattern: UNION SELECT null, null, username, password, null FROM users

A Log Example is,

- GET /products.php?q=books' UNION SELECT null, null, username, password, null FROM users--

Cross-Site Scripting (XSS)

Injects malicious scripts into web pages, often to steal user data or perform unauthorized actions.

Indicators

- Presence of <script> tags or event handlers like onmouseover, onclick, onerror.
- Example pattern: <script>alert(1);</script>

Log Example

- GET /products.php?search=<script>alert(1);</script>

Path Traversal

Exploits vulnerabilities to access files and directories outside the intended application structure.

Indicators Include

- Path traversal characters like ../ or ../../ to navigate out of the intended directory.
- Access to sensitive files like /etc/passwd.
- URL-encoded traversal characters %2E (.) and %2F (/).

Log Example

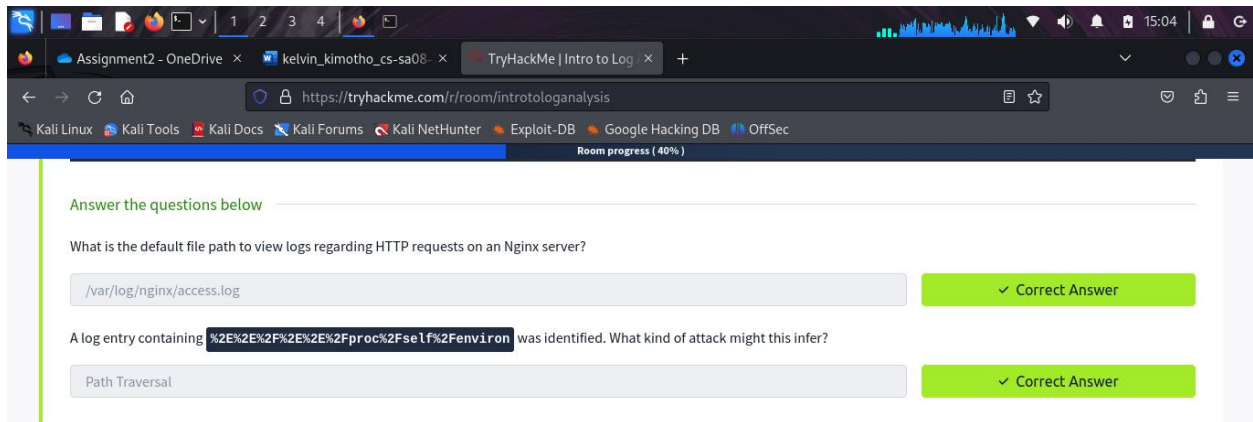
- GET ../../../../etc/passwd HTTP/1.1

Question: What is the default file path to view logs regarding HTTP requests on an Nginx server?

Answer: /var/log/nginx/access.log

Question: A log entry containing %2E%2E%2F%2E%2E%2Fproc%2Fself%2Fenviron was identified. What kind of attack might this infer?

Answer: path traversal



Automated vs. Manual Analysis

Automated Analysis

Involves using tools (e.g., XPLG, SolarWinds Loggly) that often incorporate AI/ML for log data processing and analysis.

Advantages

- **Time-saving.** Automates repetitive tasks, reducing the need for manual effort.
- **Pattern Recognition.** AI/ML can quickly identify patterns and trends in data.

Disadvantages

- **Cost.** Often requires commercial tools, which can be expensive.
- **Accuracy.** AI's effectiveness depends on the model; risk of false positives or missed threats, especially for new or unseen events.

Manual Analysis

Involves human analysts examining data and logs without the aid of automation tools.

Advantages

- **Cost-effective.** Requires minimal or no specialized tools (e.g., Linux commands).
- **Thorough Investigation.** Analysts can perform deep dives into data for a comprehensive review.
- **Reduced Risk of Overfitting.** Manual analysis can prevent false positives that automated tools might miss.

- **Contextual Analysis.** Analysts have a broader understanding of the environment, aiding in more accurate decision-making.

Disadvantages:

- **Time-Consuming.** Requires significant manual effort, especially when dealing with large data sets.
- **Potential for Missed Events.** High volume of data increases the risk of overlooking important alerts or events.

Automated Analysis is efficient and scalable but may have limitations in accuracy and cost.

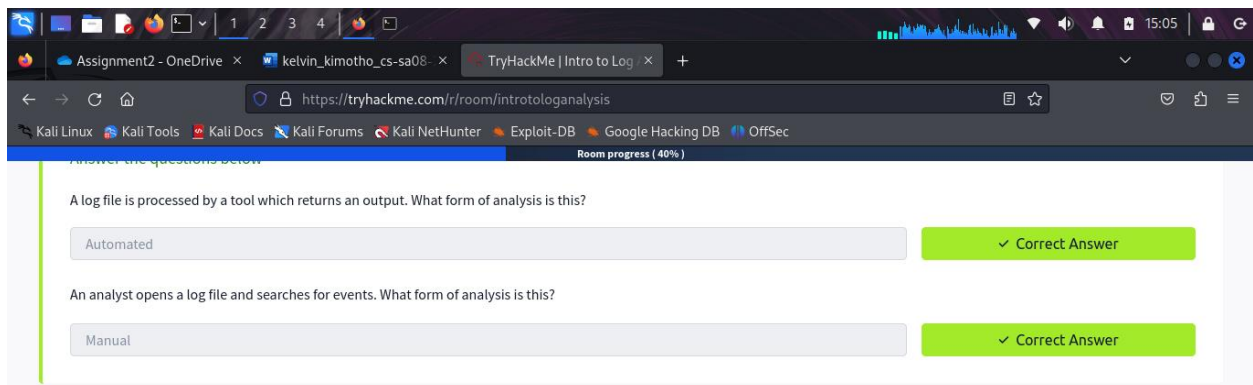
Manual Analysis provides deeper, more contextual insights and reduces the risk of false positives but is slower and more resource-intensive.

Question: A log file is processed by a tool which returns an output. What form of analysis is this?

Answer: Automated

Question: An analyst opens a log file and searches for events. What form of analysis is this?

Answer: Manual



Command Line Log Analysis Tools

We can use command line tools to analyze logs saved in files.

Cat command/tool

- Displays the entire content of a file.

- Example: **"cat apache.log"**
- Not ideal for large files due to extensive output.

less

- Allows scrolling through large files page by page.
- Example: **"less apache.log"**
- Press q to exit.

tail

- Views the last few lines of a file and follows it in real-time with -f option.
- Example: **"tail -f -n 5 apache.log"**
- Great for live log monitoring.

wc (word count)

- Provides the number of lines, words, and characters in a file.
- Example: **"wc apache.log"**
- Useful for quick statistics of log file size.

cut

- Extracts specific columns based on delimiters (e.g., space).
- Example: **"cut -d ' ' -f 1 apache.log"**
- Can extract specific fields like IP addresses or URLs.

sort

- Sorts data in ascending or descending order.
- Example: **"cut -d ' ' -f 1 apache.log | sort -n"**
- Useful for organizing log data chronologically or alphabetically.

uniq

- Removes adjacent duplicate lines from sorted input.

- Example: `"cut -d ' ' -f 1 apache.log | sort -n -r | uniq"`
- With -c, it counts occurrences of each line.

sed

- Used for text substitution (find and replace).
- Example: `"sed 's/31/Jul/2023/July 31, 2023/g' apache.log"`
- Can replace patterns in logs without modifying the original file (use -i for in-place edits).

awk

- Processes log data with conditions based on field values.
- Example: `"awk '$9 >= 400' apache.log"`
- Typically used for conditional filtering (e.g., HTTP status codes).

grep

- Searches for specific patterns in files.
- Example: `"grep 'admin' apache.log"`
- Options: -c for count, -n for line numbers, -v to exclude lines.
- Powerful for filtering specific keywords or patterns.

grep with pipe

- Combine with other commands to filter output (e.g., excluding certain entries).
- Example: `"grep -v '/index.php' apache.log | grep '203.64.78.90'"`

Command-line tools are essential for fast and efficient log analysis, offering a variety of filtering, searching, and manipulation options.

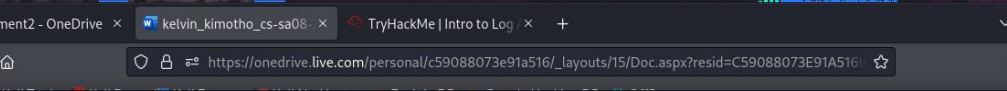
Complex datasets may require more robust log analysis tools (e.g., ELK Stack, Splunk), but these commands are invaluable for quick, on-the-fly tasks.

Question: Use cut on the apache.log file to return only the URLs. What is the flag that is returned in one of the unique entries?

Answer: c701d43cc5a3acb9b5b04db7f1be94f6

The screenshot shows a Kali Linux terminal window and a web browser window. The terminal window is in the foreground, displaying the command `cut -d ' ' -f 9 apache-1691435735822.log | grep 200` and its output, which is a list of 200 IP addresses. The web browser window is in the background, showing the OneDrive page for the file `1691435735822.log`.

- **cut -d ' ' -f 9 apache-1691435735822.log:** This extracts the 9th field (the status code) from the log file.
- **grep -c 200:** This counts the number of lines that exactly match 200 (using the ^ to indicate the start and \$ to indicate the end of the line, ensuring you match the whole status code).



The screenshot shows a Kali Linux terminal window. The prompt is `(kali@kali)-[~/Desktop]`. The user has entered the command `$ cut -d ' ' -f 9 apache-1691435735822.log | grep -c 200`. The terminal output shows the number `52` on the line following the command. The terminal window has a title bar with "kali@kali: ~/Desktop" and standard window controls. The background is a dark theme with a subtle grid pattern.

Answer: 145.76.33.201

In the `apache.log` file, which IP address generated the most traffic?

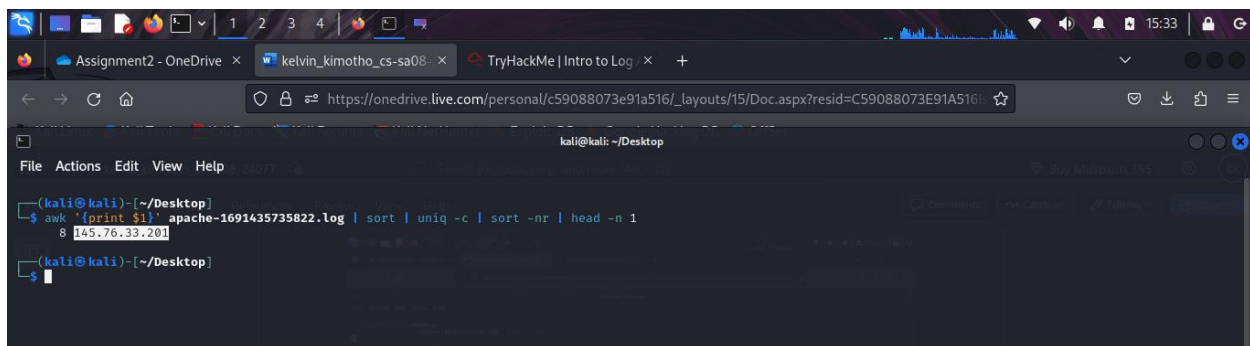
145.76.33.201

✓ Correct Answer

🔍 Hint

I ran the following command “`awk '{print $1}' apache-1691435735822.log | sort | uniq -c | sort -nr | head -n 1`”

- `awk '{print $1}'` extracts the IP address from each line.
- `sort` sorts the IP addresses.
- `uniq -c` counts the occurrences of each unique IP address.
- `sort -nr` sorts the results in numerical order, in reverse.
- `head -n 1` shows the IP address with the highest traffic.



```
kali@kali: ~/Desktop
(kali@kali)-[~/Desktop]
$ awk '{print $1}' apache-1691435735822.log | sort | uniq -c | sort -nr | head -n 1
8 145.76.33.201
(kali@kali)-[~/Desktop]
$
```

Question: What is the complete timestamp of the entry where `110.122.65.76` accessed `/login.php`?

Answer: 31/Jul/2023:12:34:40 +0000

What is the complete timestamp of the entry where `110.122.65.76` accessed `/login.php`?

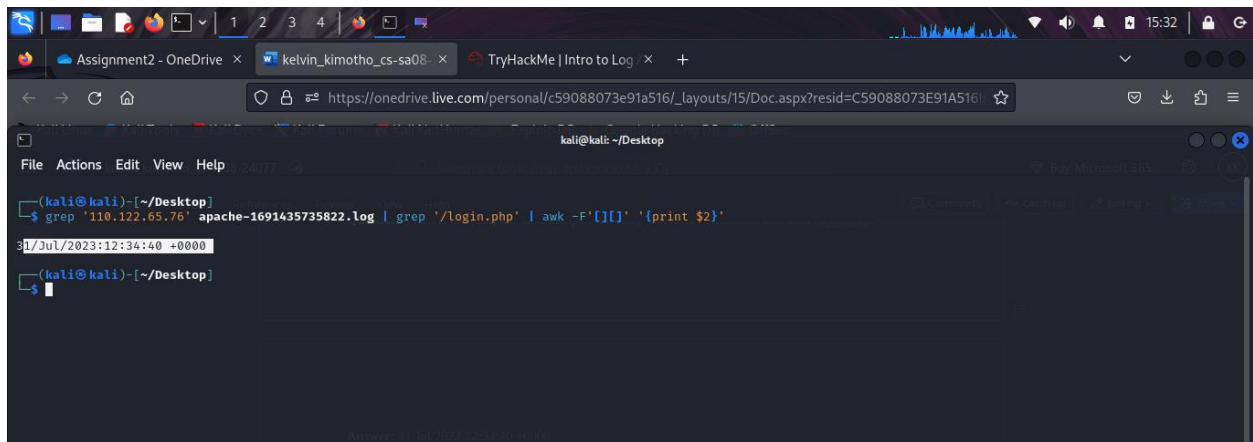
31/Jul/2023:12:34:40 +0000

✓ Correct Answer

🔍 Hint

I ran the command `grep '110.122.65.76' apache-1691435735822.log | grep '/login.php' | awk -F'[]' '{print $2}'`

- `grep '110.122.65.76' /path/to/access.log`: Filters lines with the IP 110.122.65.76.
- `grep '/login.php'`: Further filters the output to only include lines where `/login.php` was accessed.
- `awk -F'[]' '{print $2}'`: Uses `awk` with the delimiter `[]` to extract the timestamp (which is between the square brackets).



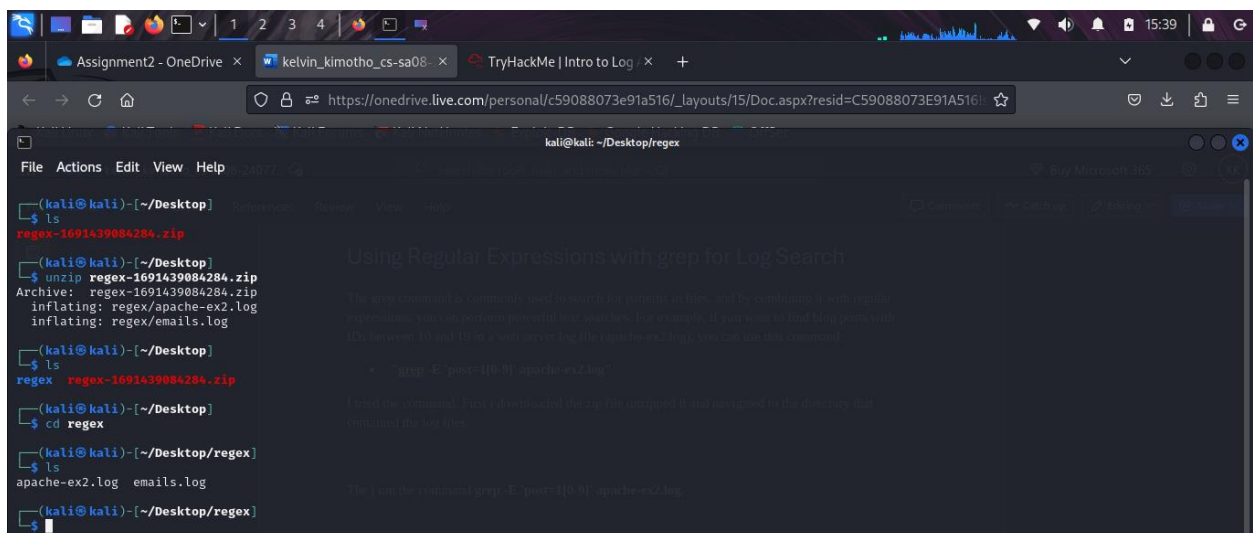
```
(kali@kali)-[~/Desktop]
$ grep '110.122.65.76' apache-1691435735822.log | grep '/login.php' | awk -F'[[]]' '{print $2}'
31/Jul/2023:12:34:40 +0000
(kali@kali)-[~/Desktop]
$
```

Using Regular Expressions with grep for Log Search

The grep command is commonly used to search for patterns in files, and by combining it with regular expressions, you can perform powerful text searches. For example, if you want to find blog posts with IDs between 10 and 19 in a web server log file (apache-ex2.log), you can use this command:

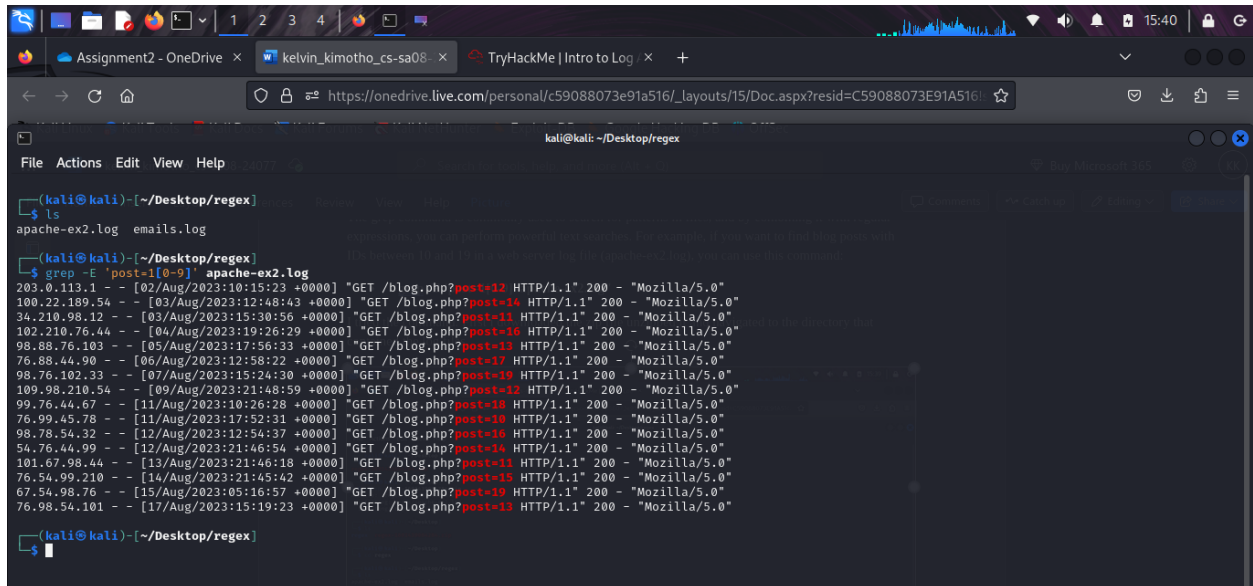
- **"grep -E 'post=1[0-9]' apache-ex2.log"**

I tried the command. First i downloaded the zip file unzipped it and navigated to the directory that contained the log files.



```
(kali@kali)-[~/Desktop]
$ ls
regex-1691439084284.zip
(kali@kali)-[~/Desktop]
$ unzip regex-1691439084284.zip
Archive:  regex-1691439084284.zip
  inflating: regex/apache-ex2.log
  inflating: regex/emails.log
(kali@kali)-[~/Desktop]
$ ls
regex  regex-1691439084284.zip
(kali@kali)-[~/Desktop]
$ cd regex
(kali@kali)-[~/Desktop/regex]
$ ls
apache-ex2.log  emails.log
(kali@kali)-[~/Desktop/regex]
$
```

The i ran the command **grep -E 'post=1[0-9]' apache-ex2.log**.



```
kali@kali: ~/Desktop/regex
$ ls
apache-ex2.log  emails.log
$ grep -E 'post=[0-9]' apache-ex2.log
203.0.113.1 - - [02/Aug/2023:10:15:23 +0000] "GET /blog.php?post=12 HTTP/1.1" 200 - "Mozilla/5.0"
100.22.189.54 - - [03/Aug/2023:12:48:43 +0000] "GET /blog.php?post=14 HTTP/1.1" 200 - "Mozilla/5.0"
34.210.98.12 - - [03/Aug/2023:15:30:56 +0000] "GET /blog.php?post=11 HTTP/1.1" 200 - "Mozilla/5.0"
102.210.76.44 - - [04/Aug/2023:19:26:29 +0000] "GET /blog.php?post=16 HTTP/1.1" 200 - "Mozilla/5.0"
98.88.76.103 - - [05/Aug/2023:17:56:33 +0000] "GET /blog.php?post=13 HTTP/1.1" 200 - "Mozilla/5.0"
76.88.44.90 - - [06/Aug/2023:12:58:22 +0000] "GET /blog.php?post=17 HTTP/1.1" 200 - "Mozilla/5.0"
98.76.102.33 - - [07/Aug/2023:15:24:30 +0000] "GET /blog.php?post=19 HTTP/1.1" 200 - "Mozilla/5.0"
109.98.210.54 - - [09/Aug/2023:21:48:59 +0000] "GET /blog.php?post=12 HTTP/1.1" 200 - "Mozilla/5.0"
99.76.44.67 - - [11/Aug/2023:10:26:28 +0000] "GET /blog.php?post=18 HTTP/1.1" 200 - "Mozilla/5.0"
76.99.45.78 - - [11/Aug/2023:17:52:31 +0000] "GET /blog.php?post=10 HTTP/1.1" 200 - "Mozilla/5.0"
98.78.54.32 - - [12/Aug/2023:12:54:37 +0000] "GET /blog.php?post=16 HTTP/1.1" 200 - "Mozilla/5.0"
54.76.44.99 - - [12/Aug/2023:21:46:54 +0000] "GET /blog.php?post=19 HTTP/1.1" 200 - "Mozilla/5.0"
101.67.98.44 - - [13/Aug/2023:21:46:18 +0000] "GET /blog.php?post=11 HTTP/1.1" 200 - "Mozilla/5.0"
76.54.99.210 - - [14/Aug/2023:21:45:42 +0000] "GET /blog.php?post=15 HTTP/1.1" 200 - "Mozilla/5.0"
67.54.98.76 - - [15/Aug/2023:05:16:57 +0000] "GET /blog.php?post=19 HTTP/1.1" 200 - "Mozilla/5.0"
76.98.54.101 - - [17/Aug/2023:15:19:23 +0000] "GET /blog.php?post=13 HTTP/1.1" 200 - "Mozilla/5.0"
$
```

Explanation of the regex pattern:

- 'post=': Matches the literal string "post=".
- '[0-9]': Matches any two-digit number that starts with 1. The [0-9] part allows any digit from 0 to 9, so this matches numbers like 10, 11, 12, and so on.

This will filter the log entries to only show those with a post ID between 10 and 19.

Log Parsing with Regular Expressions

RegExr is an online tool to help teach, build, and test regular expression patterns. I tried extracting an ip address from the log using this tool.

Log entries can often be unstructured, and using regular expressions allows us to extract specific pieces of information. For example, given this raw log entry

"126.47.40.189 - - [28/Jul/2023:15:30:45 +0000] 'GET /admin.php HTTP/1.1' 200 1275 "
'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.9999.999 Safari/537.36'"

From this log entry, we may want to extract the following information:

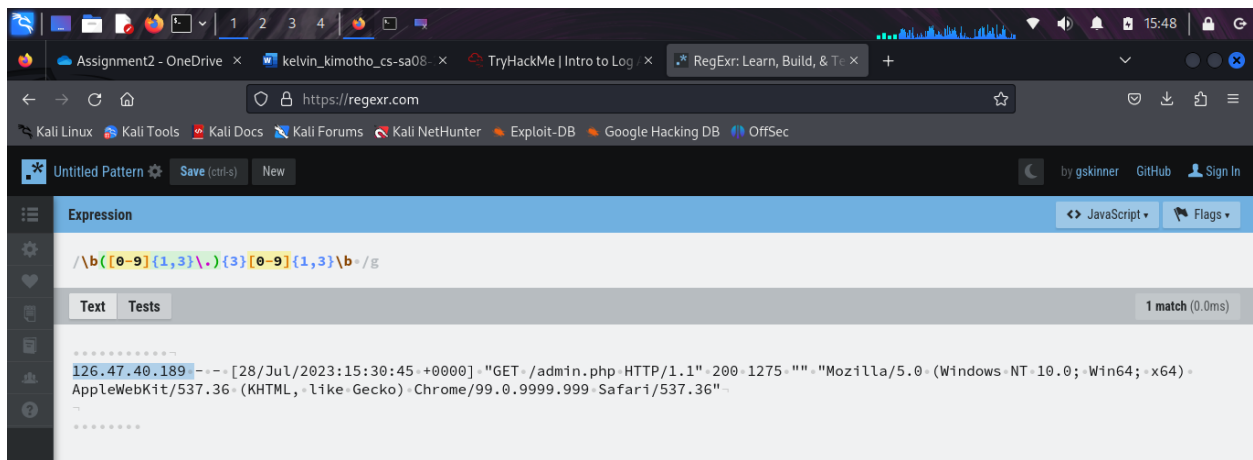
- The IP address

- The timestamp
- The HTTP method (e.g., GET)
- The URL (e.g., /admin.php)
- The user-agent string

To extract the **IP address** from the log entry, we can use the following regular expression:

- `"\b([0-9]{1,3}\.){3}[0-9]{1,3}\b"`

The Ip is highlighted when there is a match with the Regular expression supplied.



Explanation of the regex pattern:

- `\b`: Word boundary to ensure we match the complete IP address.
- `([0-9]{1,3}\.){3}`: Matches three groups of 1-3 digits followed by a literal period (.). This matches the first three octets of the IP address.
- `[0-9]{1,3}`: Matches the fourth octet (the last set of 1-3 digits in the IP address).
- `\b`: Another word boundary to ensure we match the full IP address.

This pattern will successfully extract the IP address 126.47.40.189 from the log entry.

Example: Using Grok for Log Parsing

Grok is a useful tool in Logstash for parsing unstructured log data into structured, searchable fields. Grok uses predefined patterns (like IPV4, DATE, WORD) to extract information from logs. You can also define custom patterns using regular expressions.

For example, to extract the IP address from a log message in Logstash, you can configure your logstash.conf file like this

```
“input {  
  
    ...  
  
}  
  
filter {  
  
    grok {  
  
        match => { "message" => "(?<ipv4_address>\\b([0-9]{1,3}\\.){3}[0-9]{1,3}\\b)" }  
  
    }  
  
}  
  
output {  
  
    ...  
  
}  
“
```

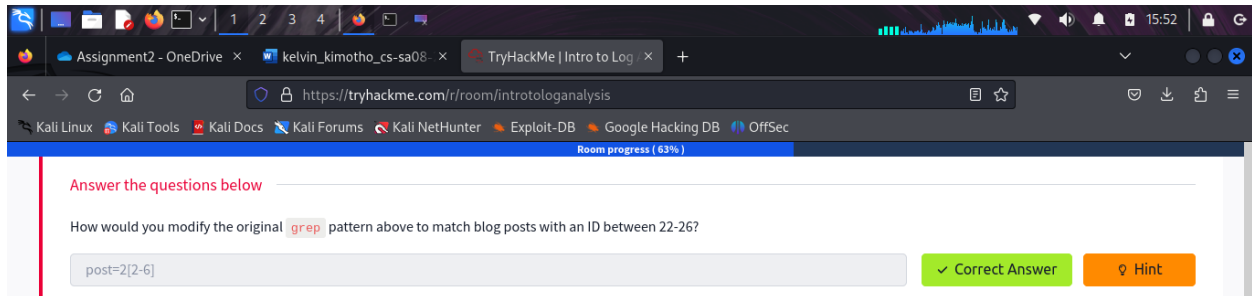
Explanation

- The "message" field contains the raw log data.
- The grok filter uses a regular expression to extract the IP address and assigns it to a custom field called ipv4_address.
- The (?<ipv4_address>...) syntax creates a named capture group for easier reference.

After this, the log data will be parsed, and the IP address will be available in the `ipv4_address` field, ready to be sent to an SIEM or further analyzed.

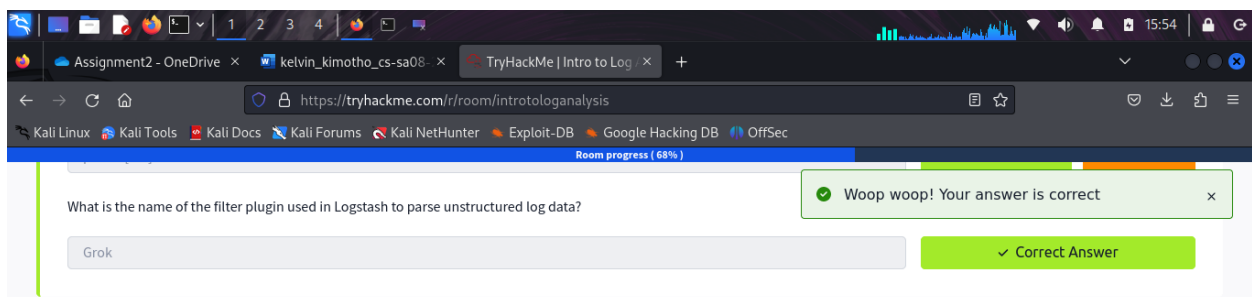
Question: How would you modify the original `grep` pattern above to match blog posts with an ID between 22-26?

Answer: `post=2[2-6]`



Question: What is the name of the filter plugin used in Logstash to parse unstructured log data?

Answer: Grok



Log Analysis Tools: CyberChef

CyberChef is a powerful, versatile tool created by GCHQ (the UK's Government Communications Headquarters) designed to handle a wide range of data analysis tasks. It's often referred to as the "Cyber Swiss Army Knife" because of its extensive set of over 300 operations that help analysts perform tasks like encoding/decoding, encryption, hashing, and data parsing. CyberChef is especially useful for analyzing log files, extracting information, and automating repetitive tasks.

Let's dive into how to use **CyberChef** for log analysis and parsing, particularly using regular expressions (regex) to extract useful data like IP addresses from log files.

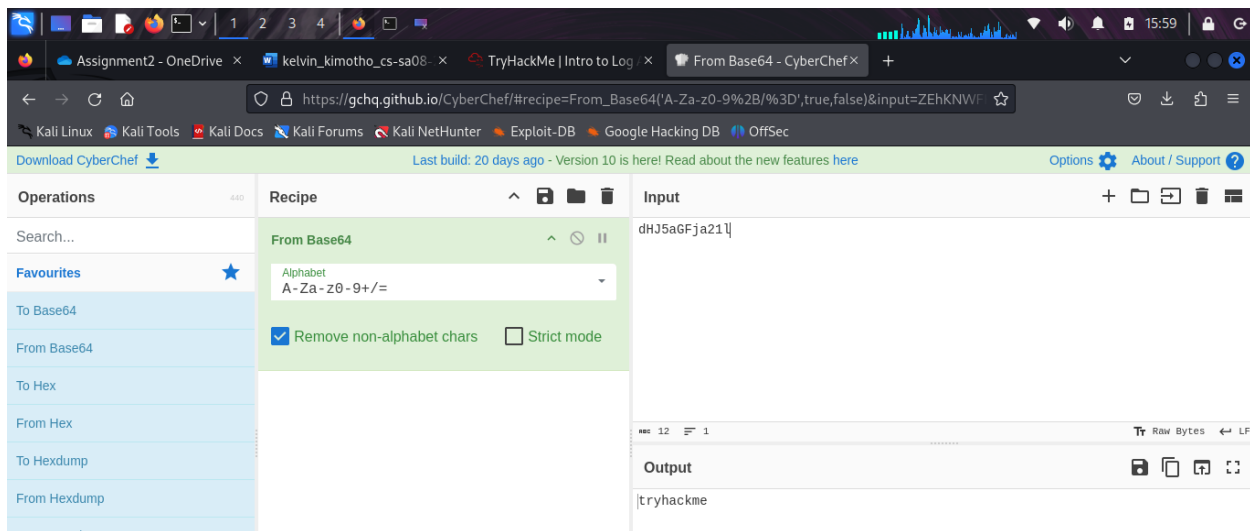
Decoding Base64 Data

To familiarize yourself with CyberChef's functionality, I started with a simple example. With a base64 encoded string

Input: "dHJ5aGFja211"

This is the base64 encoding of the word "tryhackme".

I tried this on cyberchef from base64 encoding.



Steps

1. In the **Operations** tab, I selected the "**From Base64**" operation.
2. The **Output** tab will show "**tryhackme**" as the decoded result.

Using Regular Expressions with CyberChef

One of the most powerful features of CyberChef is the ability to use regular expressions to parse and extract data from log files. For example, if you have a log file that contains multiple SSH authentication attempts, and you want to extract all the IP addresses, you can use a regex operation to filter the data.

Regex Pattern for Extracting IP Addresses

To extract IP addresses, you can use the following regular expression

```
"\b([0-9]{1,3}\.){3}[0-9]{1,3}\b"
```

This pattern is designed to match any valid IPv4 address in a log file. Let's break it down:

- `\b`: Word boundary (ensures we match complete IP addresses).
- `[0-9]{1,3}`: Matches 1 to 3 digits (0-999), which corresponds to an octet of an IPv4 address.
- `\.`: Matches a literal dot `.` character, which separates octets in the IP address.
- `{3}`: Indicates that the previous group (the octet with a dot) should be repeated exactly 3 times, to match the first three octets of the IP address.
- `[0-9]{1,3}`: Matches the final octet of the IP address.
- `\b`: Another word boundary to ensure the match ends after the IP address.

Steps to Extract IP Addresses in CyberChef

1. **Input Data.** We start by uploading or pasting your raw log data into the **Input** tab. If the log file is large or compressed (e.g., .zip or .tar.gz), you can upload the file directly.
2. **Apply the Regular Expression**
 - a. Go to the **Operations** tab, search for **Regex**, and choose the "Regex" operation.
 - b. In the **Regex** operation, paste the regex pattern `\b([0-9]{1,3}\.){3}[0-9]{1,3}\b` into the **Regex** field.
 - c. Select "**List matches**" in the **Output Format** option. This ensures that CyberChef will only display the matched IP addresses, ignoring other data in the log file.
3. **View the Output.** Once the operation is applied, the **Output** tab will show a list of IP addresses that were found in the log data. This eliminates all other noise from the log, making it easy to identify potential sources of authentication attempts or any suspicious activity.

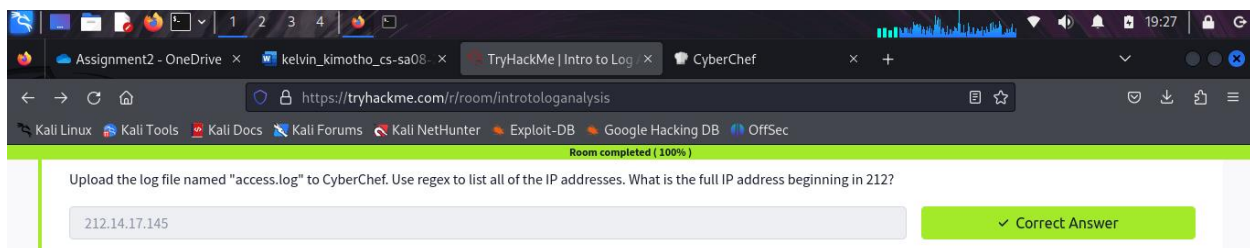
Uploading Files to CyberChef

CyberChef makes it easy to analyze large log files by allowing you to upload them directly. To do so:

1. **Upload File.** Click on the **box with an arrow pointing inside it** in the **Input** tab to upload your log file.
2. **Unzipping Compressed Files.** If your log files are compressed (e.g., .zip or .tar.gz), you can use CyberChef's built-in operations to extract the files. Look for the **"Gunzip"**, **"Unzip"**, or **"Untar"** operations under the **Operations** tab to handle different types of compressed files.
3. Once the file is uploaded or unzipped, we can now extract the data you need (like IP addresses) using regex.

Question: Upload the log file named "access.log" to CyberChef. Use regex to list all of the IP addresses. What is the full IP address beginning in 212?

Answer: 212.14.17.145



I uploaded **access.log** file to cyberchef.

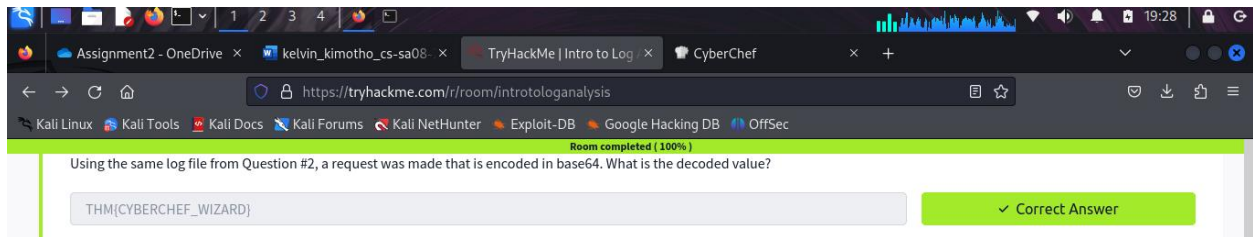
The screenshot shows the CyberChef web application. The 'Recipe' tab is selected, and a 'From Base64' recipe is applied. The 'Input' tab displays a log file named 'access.log' with two entries. The 'Output' tab shows the decoded content of the log file, which includes IP addresses and user agents.

Then using the regex pattern `\b([0-9]{1,3}\.){3}[0-9]{1,3}\b` to search for values that are IP addresses. And the Ip was ” 212.14.17.145”.

The screenshot shows the CyberChef web application. The 'Recipe' tab is selected, and a 'Regular expression' recipe is applied. The 'Input' tab displays a log file named 'access.log' with two entries. The 'Output' tab shows the results of the regex search, listing IP addresses.

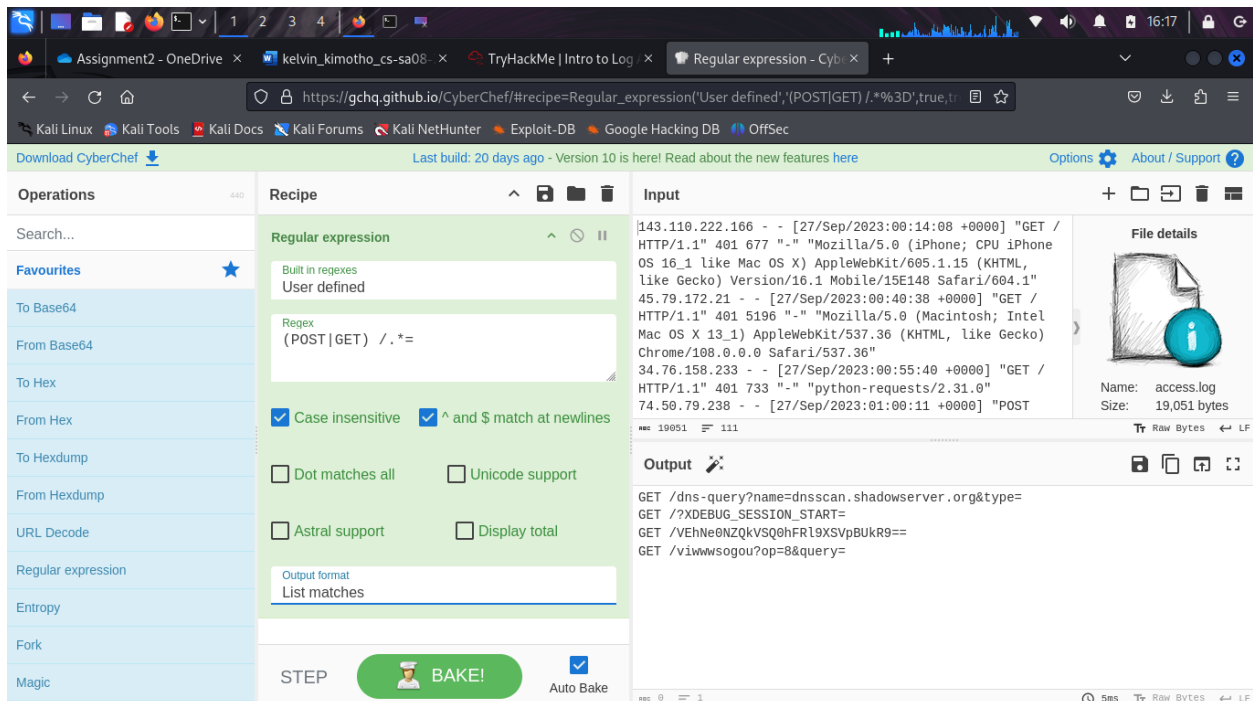
Question: Using the same log file from Question #2, a request was made that is encoded in base64. What is the decoded value?

Answer: THM{CYBERCHEF_WIZARD}

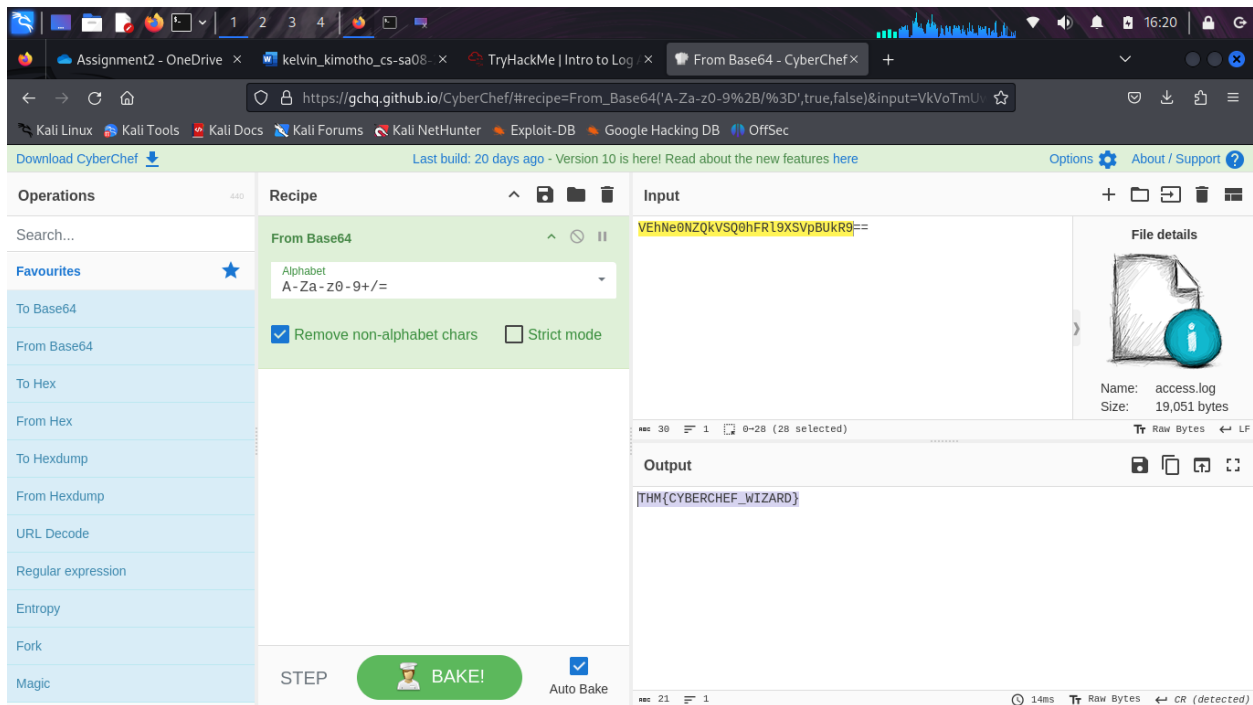


I needed to find a request in the log. HTTP requests in this log are either GET or POST requests. So we need the RegEx that will find any string that starts with either GET or POST and ends with an = sign, with however many characters in the middle. We can now build our expression:

- “(POST|GET) /.*=”

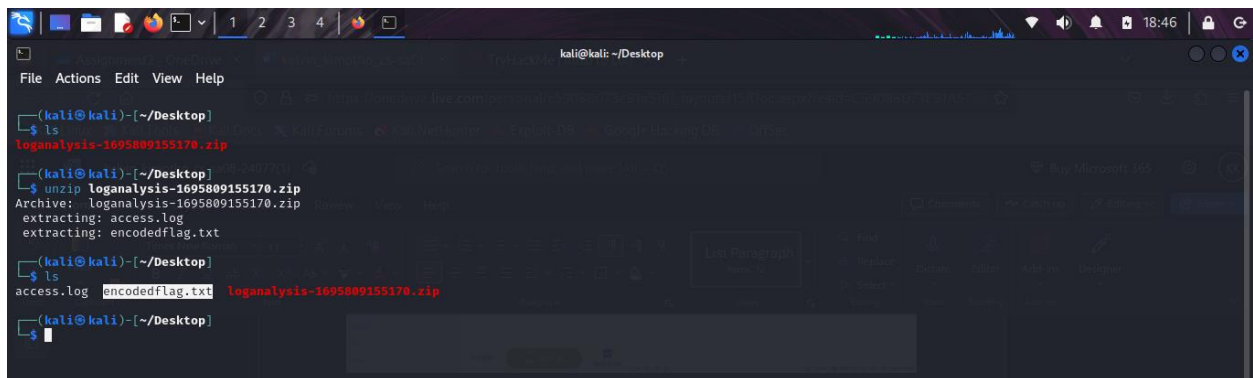
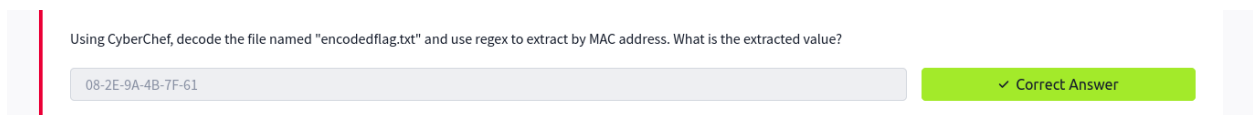


I found a base 64 encoding “VEhNe0NZQkVSQ0hFRl9XSVPBUkR9==” in a get request and used cyberchef to decode it and found the flag “THM{CYBERCHEF_WIZARD}”.



Question: Using CyberChef, decode the file named "encodedflag.txt" and use regex to extract by MAC address. What is the extracted value?

Answer: 08-2E-9A-4B-7F-61



I loaded the file “encodedflag.txt” into our skilled Chef and ask him to decode it from Base64. In the output, we will have what is seemingly a list of MAC addresses.

The screenshot shows the CyberChef web application. The 'Recipe' tab is selected, and the 'From Base64' operation is configured. The 'Input' field contains a long Base64-encoded string. The 'Output' field shows the decoded result, which is a hex dump of a file named 'encodedflag.txt' with a size of 4,344 bytes.

Look at the first address, for example, it starts out good but then there are 3 hexadecimal digits in the 5th group. I added another operation that uses regular expressions to find a MAC address

The predefined one in cyberchef for mac addresses.

- “[A-Fa-f\d]{2}(?:[-:][A-Fa-f\d]{2}){5}”

The screenshot shows the CyberChef web application. The 'Recipe' tab is selected, and the 'From Base64' operation is configured. The 'Input' field contains the same long Base64-encoded string. The 'Output' field shows the decoded result, which is a hex dump of a file named 'encodedflag.txt' with a size of 4,344 bytes.

Log Analysis Tools: Yara and Sigma

In log analysis, pattern matching is an essential technique for detecting and identifying events of interest. Two powerful tools for pattern matching in security analysis are **Sigma** and **Yara**.

Each tool serves a different purpose but can be highly effective in identifying patterns in log files or other data sources, such as network traffic or malware analysis.

Sigma

Sigma is an open-source tool that uses a standardized YAML-based rule format to describe log events and searches. It's primarily used in Security Information and Event Management (SIEM) systems to detect events in logs, automate searches, and identify potential threats. Sigma helps analysts create rules for searching logs in a structured format, enabling easier detection of suspicious or abnormal behavior.

Example: Sigma Rule for Failed SSH Logins

Below is an example of a Sigma rule designed to detect failed SSH login attempts in Linux sshd logs.

“title: Failed SSH Logins

description: Searches sshd logs for failed SSH login attempts

status: experimental

author: CMNatic

logsource:

product: linux

service: sshd

detection:

selection:

type: 'sshd'

a0|contains: 'Failed'

a1|contains: 'Illegal'

condition: selection

falsepositives:

- Users forgetting or mistyping their credentials

level: medium

“

Explanation of the Sigma Rule Components

1. **title:** Failed SSH Logins
2. This is the name or title of the Sigma rule, describing what the rule is intended to detect.
3. **description:** Searches sshd logs for failed SSH login attempts

A brief explanation of the rule's purpose.

4. **status:** experimental

Indicates the maturity of the rule. "Experimental" means the rule may need further testing or improvements before it's considered stable.

5. **author:** CMNatic

The author of the rule.

6. **logsource**

a. **product:** linux

This indicates that the logs are from a Linux system.

b. **service:** sshd

This specifies that the rule looks at SSH daemon logs (sshd).

7. **detection**

This section defines what the rule is searching for in the logs:

- a. type: 'sshd'

The log source type is sshd.

- b. a0|contains: 'Failed'

Looks for log entries containing the word "Failed".

- c. a1|contains: 'Illegal'

Also looks for entries containing the word "Illegal" (indicating failed login attempts).

- d. **condition:** selection

If both conditions ('Failed' and 'Illegal') are found, the rule will trigger.

8. **falsepositives**

Lists possible reasons for false alarms. In this case, it might be triggered by legitimate users mistyping their credentials.

9. **level:** medium

Indicates the severity or importance of the event. This is typically set based on how critical the event is in a security context.

Using Sigma in SIEM

A sigma rule like the one above can be used to automate log searches in a SIEM system. It's a way to translate human-readable logic into a structured, standard format that SIEM systems can process and use for alerting and reporting.

Sigma **helps streamline log analysis** by providing consistent, reusable, and shareable rules for detecting various activities, such as failed logins, privilege escalation, or anomalous behavior.

Yara

Yara is another tool used for pattern matching, but unlike Sigma, it's mainly focused on identifying binary and textual patterns. Yara is particularly well-known in malware analysis, where it helps detect malware by searching for specific byte sequences or strings. However, it can also be useful in log analysis by searching for textual patterns like IP addresses, URLs, or other identifiable markers.

Example: Yara Rule for Detecting IP Addresses in Logs

Here's an example of a Yara rule that detects IPv4 addresses in a log file

```
rule IPFinder {  
    meta:  
        author = "CMNatic"  
    strings:  
        $ip = /[([0-9]{1,3}\.){3}[0-9]{1,3}/ wide ascii  
    condition:  
        $ip  
}
```

Explanation of the Yara Rule Components

1. **rule:** IPFinder
2. The name of the rule. This identifies the specific rule used to search for patterns.
3. **meta**
 - a. author = "CMNatic"

Metadata for the rule. In this case, the author of the rule.

4. **strings**
 - a. \$ip = /[([0-9]{1,3}\.){3}[0-9]{1,3}/ wide ascii

This line defines a regular expression pattern (regex) to look for IPv4 addresses in the log file. It matches the typical format of an IPv4 address (e.g., 192.168.1.1).

5. condition

a. \$ip

If the regular expression for the IP address is found in the file, the rule triggers an alert.

Using Yara for IP Address Detection

We can use the above Yara rule to scan log files for IP addresses. For example, suppose you have a log file `apache2.txt` containing SSH login attempts, and you want to identify any IP addresses that appear in the log.

Here's how you would run the Yara rule

`“yara ipfinder.yar apache2.txt”`

If an IP address is found in the log, Yara will flag the file as matching the IPFinder rule. In this case, you might use it to identify potential malicious IPs or track IPs that repeatedly attempt login.

Expanding the Yara Rule

The above Yara rule can be expanded in several ways

- **Multiple IP Addresses.** You can modify the rule to match multiple IP addresses or specific address patterns.
- **IP Address Range.** If you want to detect IP addresses within a specific range or subnet, you can extend the rule to capture those ranges.
- **IP Address Frequency.** You can modify the rule to trigger if an IP address appears more than a certain number of times, helping you identify suspicious patterns (e.g., brute force attacks).

In log analysis, We can use Yara to flag specific patterns that may indicate an attack or other suspicious activity. For example, We could create a Yara rule to detect

- **Repeated failed login attempts** from the same IP address.

- **Access to sensitive resources** (e.g., /admin paths in URLs).
- **Unusual or known malicious IP addresses** that have been flagged in threat intelligence feeds.

Question: What languages does Sigma use?

Answer: YAML

Question: What keyword is used to denote the "title" of a Sigma rule?

Answer: title

3. Identify threats

Sigma uses the YAML syntax for its rules. This task will demonstrate Sigma being used to detect failed login events in SSH. Please note that writing a Sigma rule is out-of-scope for this room. However, let's break down an example Sigma rule for the scenario listed above:

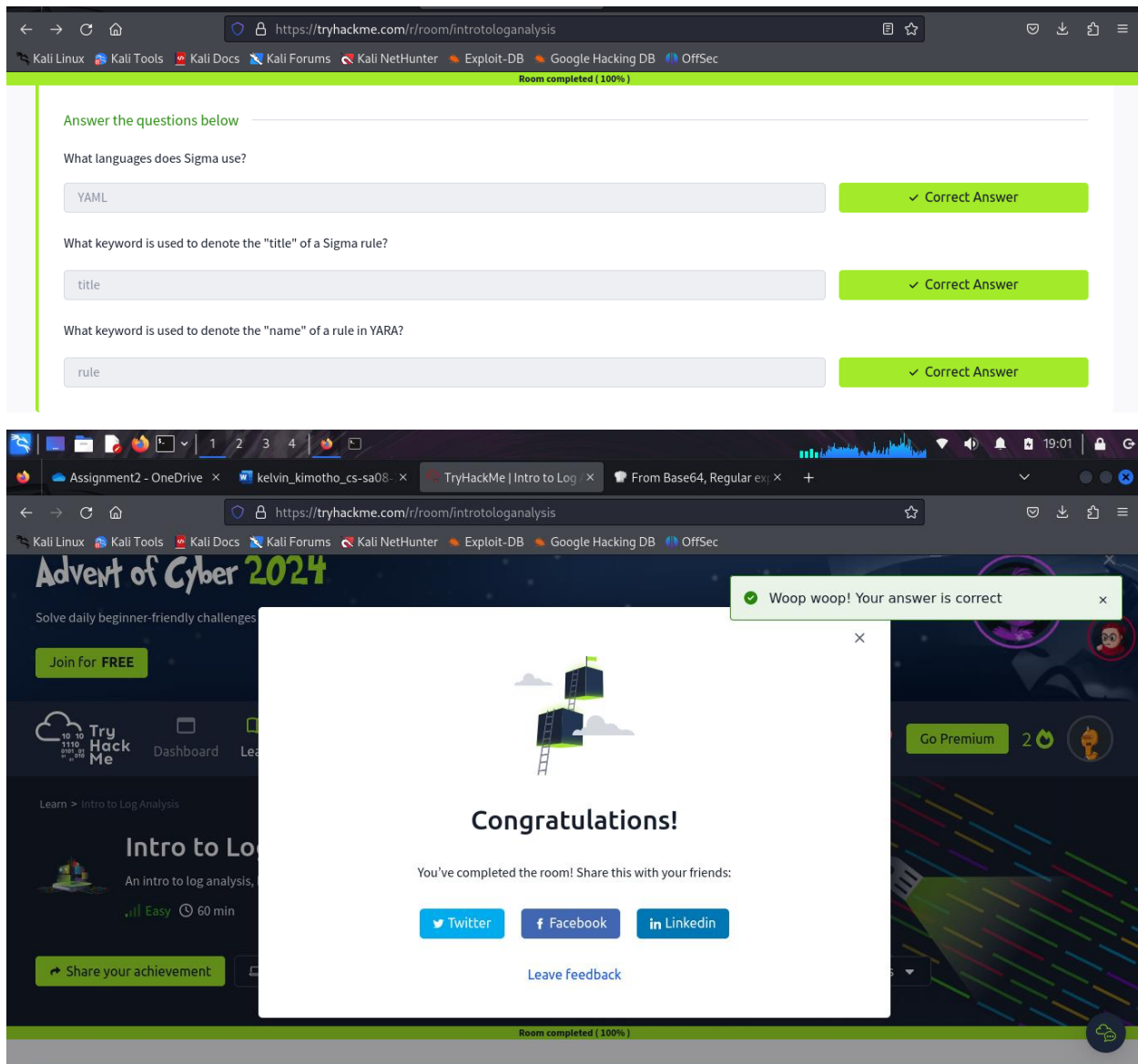
```
title: Failed SSH Logins
```

Question: What keyword is used to denote the "name" of a rule in YARA?

Answer: rule

Let's look at this example Yara rule called "IPFinder". This YARA rule uses regex to search for any IPV4 addresses. If the log file we are analyzing contains an IP address, YARA will flag it:

```
rule IPFinder {
```



Conclusion

In this module, I learned various essential techniques for log analysis and pattern matching, including the use of regular expressions (regex) with tools like grep to filter specific log entries, such as blog posts with particular IDs. I also explored how to parse unstructured log data using Logstash's Grok filter plugin, which helps extract key information like IP addresses, timestamps, and HTTP methods. Additionally, I gained hands-on experience with CyberChef for analyzing log files and extracting data using regex, as well as automating the process of decoding and parsing encoded information. Finally, I was introduced to Sigma and YARA, two powerful tools

for pattern detection, with Sigma enabling structured log searches in SIEM systems, and YARA helping to identify specific patterns, such as IP addresses, in both log and binary data. These tools and techniques provide a comprehensive approach to log analysis, enhancing my ability to identify and investigate security events.