Inject 24

Script to Process Log Files

Team 9

IT Manager,

Here is the script that will run every 15 minutes that will search through the current collective log file in the designated IP ranges:

```
#!/bin/bash

# Define the IP ranges
ip_ranges=("10.140.1.0/24" "172.18.0.0/16" "192.168.251.0/24" "10.128.0.0/12")

# Get the last processed line number from a marker file
last_line=$(cat marker_file)

# Get the current line number of the log file
total_lines=$(wc -l < "log_file")

# Loop through each line of the log file, starting from the last processed line
for ((i=last_line; i<=total_lines; i++)); do
  # Get the source IP address from the current line
  source_ip=$(sed "${i}q;d" log_file | awk '{print $1}')

  # Check if the source IP is in any of the IP ranges
  for ip_range in "${ip_ranges[@]}"; do
    if echo "$source_ip" | grep -qP "$ip_range"; then
```

```
    # Log the matching source IP
    echo "$source_ip matches $ip_range"
  fi
 done
done


# Update the marker file with the current line number
echo "$total_lines" > marker_file
*/15 * * * * /bin/bash /path/to/script.sh
```

We are running this script on our Debian 8.5 server, it keeps track of the last processed line using a marker file, so that it only searches the new entries in the log files in each iteration.

To verify that it runs every 15 minutes, we utilized the cron utility and the last line in the file which will ensure it stays running at the regular intervals.

The way that we implemented the logic of processing new entries and avoiding redundant processing is by doing the following:

1. A marker file **marker_file** is used to store the last processed line number.
2. At the start of each run, the script reads the last processed line number from the marker file and stores it in the **last_line** variable.
3. The script then finds the current line number of the log file and stores it in the **total_lines** variable.
4. The script uses a for loop to iterate through each line of the log file, starting from the **last_line** and ending at **total_lines**.
5. For each line, the script extracts the source IP address and checks if it matches any of the IP ranges. If it does, it logs the match.
6. Finally, the script updates the marker file with the current **total_lines** so that the next time it runs, it will start from the next unprocessed line in the log file.

An example output for this script might look like this:

10.140.1.5 matches 10.140.1.0/24

172.18.2.0 matches 172.18.0.0/16

192.168.251.1 matches 192.168.251.0/24

Thank you,

Team 9