Process and File monitoring script report

Robert Onuoha

Introduction

This report is documentation for two shell scripts designed to enhance the visibility of linux systems. Part A is focused on process monitoring and tracking the resource utilisation of every user. Part B is focused on file monitoring and tracking the permissions and number of files opened by each user. Each part addresses a critical part of system security and the two work together to give a holistic solution. Together these scripts offer a framework for improving security by giving admins extra visibility into their linux systems, aiding them in preventing potential threats.

Overview

Part A: Process monitoring:

This script was developed to provide a detailed report of resource utilisation sorted per user. It aggregates data across processes belonging to the user, and outputs an overview of cpu utilisation, memory consumption and the number of open files. This allows for admins to see resource intensive users. To achieve this the script uses linux commands like ps to loop through all the users currently with a process active on the linux system. Awk is used to get the cpu time rather than percentage and to convert the memory used by each user from KB to MB. Grep is used to find the number of files opened by each user along with wc -l to actually count each file as one per line.

Part B File monitoring:

This script was developed to monitor the issue of file permissions for individual users. It works by identifying files with specific permissions that may pose a risk. By numbering the amount of Setuid/Setgid permissions and world writable files per user, the script allows admins to have a comprehensive view of potential security flaws. The users are looped through and the find command is used along with the -perm option to find setgid and setuid files. Permission 4000 returns all setuid files while permission 2000 returns all setgid files, in order to not overwhelm the admin all error codes are sent to /dev/null and are not printed in the terminal.

Issues and solutions

Issues

1. The users could not be sorted in descending order of cpu utilisation or number of setuid files.

2. The cpu time was in the 00:00:00 format which made adding them up in total seconds problematic.

3. The memory for each user was not totalled and was also in the KB unit instead of MB.

4. The ps -u option did not allow a search for the total amount of files opened per user.

5. The list of users had duplicate results.

6. The find command was giving a list of errors, clogging up the terminal.

Solutions

1. The sort option is used with the ps command to sort them alphabetically instead.

2. awk '{split($1, a, ":"); sum += a[1]*3600 + a[2]*60 + a[3]} END {print sum}') is used to split the format and all values together which gives the sum in seconds.

3. awk '{ sum+=$1 } END {print sum}') allowed for all the different values to be totalled, total_memorymb=$((total_memorykb / 24)) divided the total to give the MB value instead of the KB value.

4. The aux option on the ps command with grep user searches for all the files associated with said user, wc -l counts the number of files.

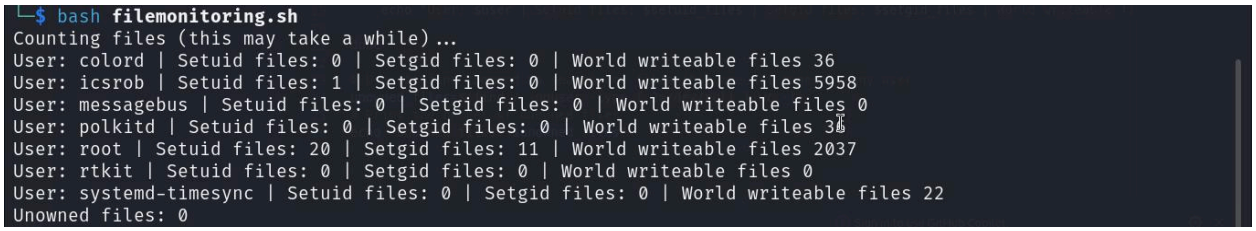
5. The uniq option for the ps command removed the duplicates from the list.

6. 2>/dev/null redirects error messages so the terminal is no longer clogged

Output comments

```
└─$ bash processmonitoring.sh
User: colord | Total CPU time: 0 Secs | Memory Usage: 372 MB | Files Opened: 2
User: icsrob | Total CPU time: 84 Secs | Memory Usage: 151979 MB | Files Opened: 81
User: messagebus | Total CPU time: 11 Secs | Memory Usage: 207 MB | Files Opened: 1
User: polkitd | Total CPU time: 0 Secs | Memory Usage: 386 MB | Files Opened: 2
User: root | Total CPU time: 525 Secs | Memory Usage: 10909 MB | Files Opened: 171
User: rtkit | Total CPU time: 0 Secs | Memory Usage: 122 MB | Files Opened: 2
User: systemd-timesync | Total CPU time: 0 Secs | Memory Usage: 165 MB | Files Opened: 2
```

There is no unfair usage from this output of the process monitoring script, users like root and icsrob have considerably more usage than the other users. This however is simply because of their roles, root is expected to have high usage because it has system wide privileges while icsrob is the main user account on the system.



The presence of such a high amount of world writable files on the system does present a security risk. This is because they could be potentially modified by any user on the system. In order to mitigate this frequent audits of permissions should be done making sure the amount of world writable files is limited to those only necessary for system functionality.

References

(No date). Available at: https://stackoverflow.com/questions/28905083/how-to-sum-a-column-in-awk.

(No date a). Available at: https://www.networkworld.com/article/969352/how-to-sort-ps-output.html.

(No date a). Available at: https://www.geeksforgeeks.org/ps-command-in-linux-with-examples/.

(No date a). Available at: https://www.cyberciti.biz/faq/find-files-that-do-not-have-any-owners-or-do-not-belong-to-any-user-under-linuxunix/.

(No date a). Available at: https://askubuntu.com/questions/679344/how-can-i-find-world-writable-files-and-folders-and-set-the-sticky-bit.

Appendix

```
#!/bin/bash



# Get a list of all users while removing duplicates
users=$(ps -e -o user= | sort | uniq)

# Loop through each user
echo "$users" | while read -r user; do
   # Calculate total CPU time for the user              # Remove the 00:00:00 format and get the total time in seconds
   total_cpu_time=$(ps -U "$user" -o time= --no-headers | awk '{split($1, a, ":"); sum += a[1]*3600 + a[2]*60 + a[3]} END {print sum}')
   # Calculate the total amount of memory used for the user in KB
   total_memorykb=$(ps -U "$user" -o rss= | awk '{ sum+=$1 } END {print sum}')
   # Convert the memory from KB to MB
   total_memorymb=$((total_memorykb / 24))
   # Finds the amount of files opened (1 file per line minus the input)
   total_files_opened=$(ps aux | grep "$user" | wc -l)
   # Print the user, their total CPU time, their memory used and number of files opened
   echo "User: $user | Total CPU time: $total_cpu_time Secs | Memory Usage: $total_memorymb MB | Files Opened: $total_files_opened"
done
```

—----------

```
#!/bin/bash

# Get a list of all users while removing duplicates
users=$(ps -e -o user= | sort | uniq)

echo "Counting files (this may take a while)..."

# Loop through each user
echo "$users" | while read -r user; do

    # Use the find command in the loop to count all the setuid files (permission 4000) while redirecting error messages with /dev/null
    setuid_files=$(find / -user "$user" -perm /4000 -type f 2>/dev/null | wc -l)
    # Use the find command in the loop to count all the setgid files (permission 2000) while redirecting error messages with /dev/null
    setgid_files=$(find / -user "$user" -perm /2000 -type f 2>/dev/null | wc -l)
    # Use the find command in the loop to count all the world writeable files (permission -o+w) while redirecting error messages with /dev/null
    world_writeable_files=$(find / -user "$user" -perm -o+w -type f 2>/dev/null | wc -l)

    # Print the user, number of setuid files, number of setgid files and world writeable files
    echo "User: $user | Setuid files: $setuid_files | Setgid files: $setgid_files | World writeable files $world_writeable_files"

done

# Use the find command to search for files that are not owned by any user
unowned_files=$(find / -nouser -type f 2>/dev/null | wc -l)
# Print the number of unowned files
echo "Unowned files: $unowned_files"
```