

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | | Automating the Retrieval and Analysis of System Logs from Remote Machines | | |  | |
|  |  | | | | |  |
|  | | |  |  | | |
|  | | | Yash Devadiga |  | | |

Table of Contents

[1. Introduction 1](#_Toc142762254)

[2. Architecture 1](#_Toc142762255)

[3. Methodology 2](#_Toc142762256)

[3.1 Setting up the environment 2](#_Toc142762257)

[3.2 Log retrieval 2](#_Toc142762258)

[3.3 Log analysis 2](#_Toc142762259)

[3.4 Log reporting 2](#_Toc142762260)

[4. Instructions and Results 3](#_Toc142762261)

[5. Bash Script 6](#_Toc142762262)

[6. Conclusion and Possible Improvements 9](#_Toc142762263)

**Table of Figures**

[**Figure 1: Architecture 1**](#_Toc142762335)

[**Figure 2: Virtual Box Manager 3**](#_Toc142762336)

[**Figure 3: Installing rsyslog 3**](#_Toc142762337)

[**Figure 4: Configuring rsyslog.conf 4**](#_Toc142762338)

[**Figure 5: Creating a ssh key 4**](#_Toc142762339)

[**Figure 6: Configuring sudoers file 5**](#_Toc142762340)

[**Figure 7: Log report 5**](#_Toc142762341)

# Introduction

This project aims to automate retrieving logs from a remote machine and analyzing them. My primary reason for undertaking this project is to gain practical experience in log analysis, automation using Bash, and generating reports focusing on various security events and anomalies for an Ubuntu web server. Through this project, I have gained a thorough understanding and expertise in Bash scripting, system logging, and log analysis. At the end of the project, I have achieved the following deliverables:

1. A fully functional Bash script for automated log retrieval and analysis.
2. A detailed report presenting the analysis results, security events, and anomalies found during the process.

# 2. Architecture

The project architecture follows a well-defined sequence of steps to achieve its objectives. At its core, the architecture encompasses two primary phases: log retrieval and analysis. The process commences with the automation of log retrieval using Bash scripting techniques, facilitated by commands and utilities such as rsync and ssh. These tools establish secure connections to the remote machine, ensuring logs are fetched efficiently.

Once logs are securely fetched, the architecture transitions into the analysis phase. This involves leveraging a toolkit of tools including grep, awk, tail, etc. These tools collectively form the engine for deciphering log entries, identifying patterns, and uncovering security events and anomalies. The insights gained from the analysis are then meticulously structured into detailed reports, providing a comprehensive view of the server's security landscape.

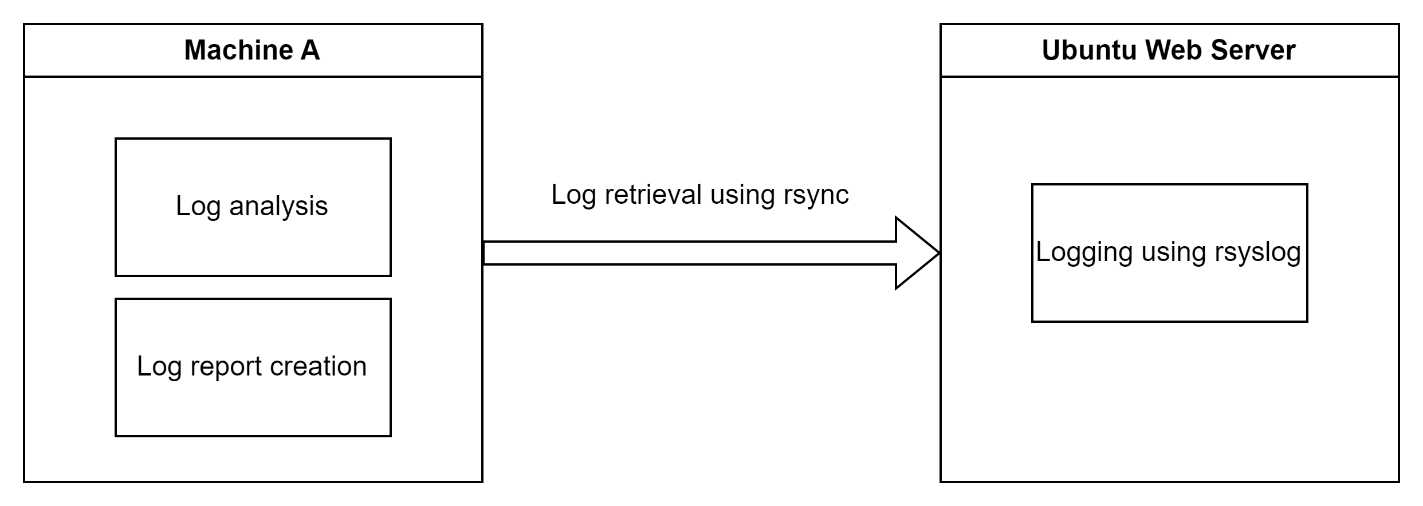


Figure 1: Architecture

# 3. Methodology

## 3.1 Setting up the environment

To create an effective environment for thorough log analysis, I've set up a two-machine arrangement, facilitated through the virtual orchestration platform VirtualBox. One machine serves as the central hub for log analysis, where the intricate process of deciphering log data takes place. The second machine, acting as a remote web server, uses rsyslog to efficiently capture and organize log entries. This separation of roles ensures that the analysis hub can focus solely on processing log information, while the remote web server efficiently logs events using rsyslog. This well-coordinated setup forms the core of our project, enabling us to seamlessly retrieve logs, conduct in-depth analysis, and generate valuable insights to enhance our understanding of cybersecurity and log interpretation.

## 3.2 Log retrieval

For this, I've taken a practical approach to log retrieval. To move log data from the remote web server to the analysis hub, I've chosen rsync, a reliable tool for smooth data transfer. To make this process secure and hassle-free, I've also set up an SSH key pair that lets us log in without needing a password. This not only boosts security but also speeds up the whole data exchange.

Moreover, on the remote web server, I've configured things so that rsync can run without asking for a password. This means it can do its job smoothly, transferring logs to the analysis hub without any interruptions. This setup, while keeping things safe, also ensures that everything runs efficiently.

These steps form the core of our methodology, ensuring a solid and secure foundation for log retrieval. With data moving seamlessly between the machines, we're all set for our detailed analysis, diving deep into the world of log interpretation.

## 3.3 Log analysis

My log analysis approach involves using Bash scripting to dissect log entries, tapping into commands like grep and awk to pull out important patterns. I am also extracting timestamps from these logs. This helps us focus on the latest logs, making sure we're always looking at the most recent activities. This combination of scripting, pattern recognition, and timestamp filtering gives us a sharp insight into what's happening in the logs, stepping up our game in understanding cybersecurity dynamics.

## 3.4 Log reporting

When it comes to presenting the log analysis findings, I've opted for a straightforward yet effective approach – log reporting directly on the terminal. This carefully designed reporting mechanism presents essential information in a neatly organized table format. This table encapsulates critical fields, providing a concise snapshot of key details extracted from the logs. By distilling complex data into an easily digestible table, we aim to enhance accessibility and facilitate quick decision-making. This direct and efficient presentation method ensures that important insights are readily available, allowing us to delve into the heart of security events and anomalies without any unnecessary details.

# 4. Instructions and Results

1. Set up two machines – one for log retrieval and analysis and one for log generation.

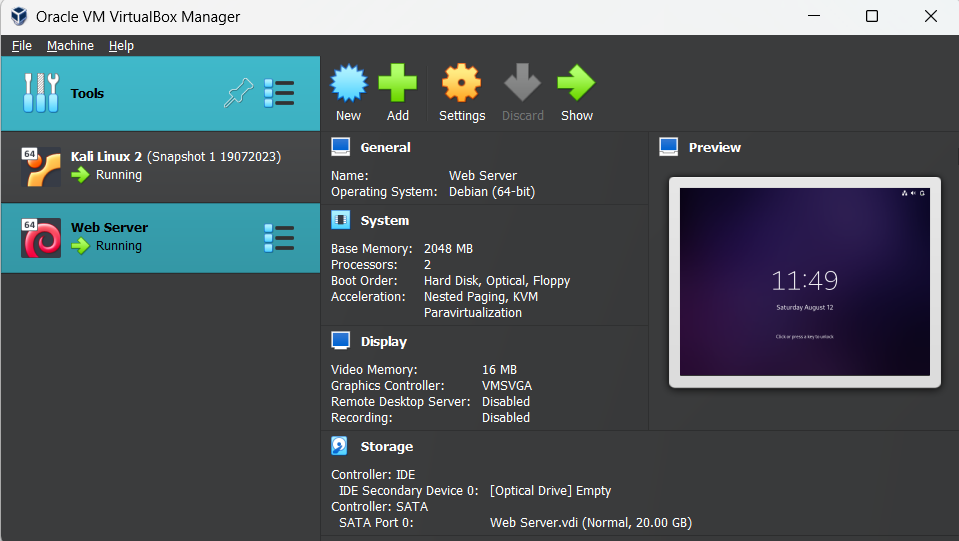


Figure 2: Virtual Box Manager

1. Install all necessary libraries on both machines like: rsyslog, apache2, net-tools, etc.

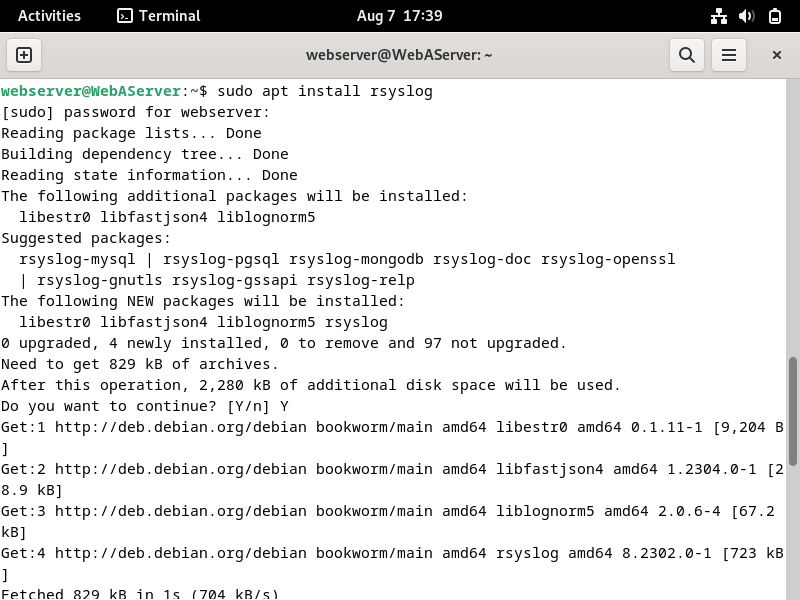


Figure 3: Installing rsyslog

1. Configure rsyslog.conf to log all necessary logs to a specific folder.

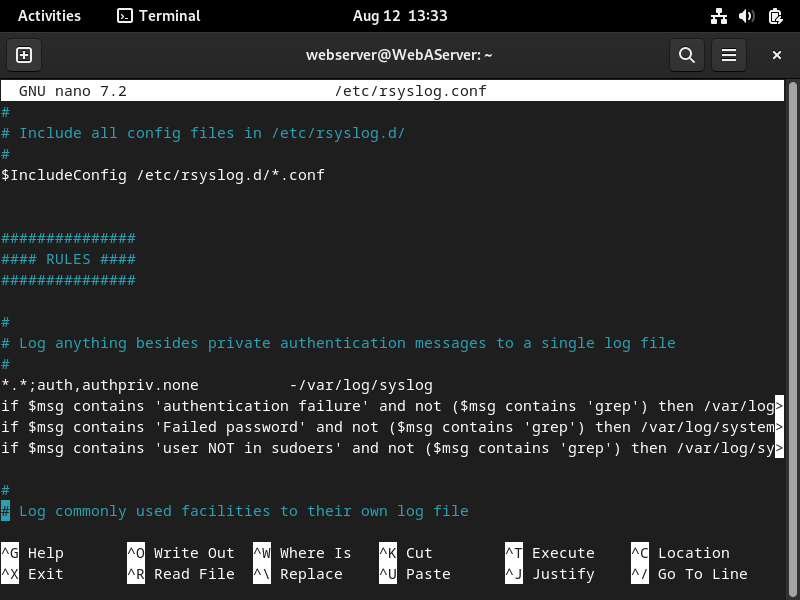


Figure 4: Configuring rsyslog.conf

1. Create ssh key for remote access to the log generating machine for better access and security.

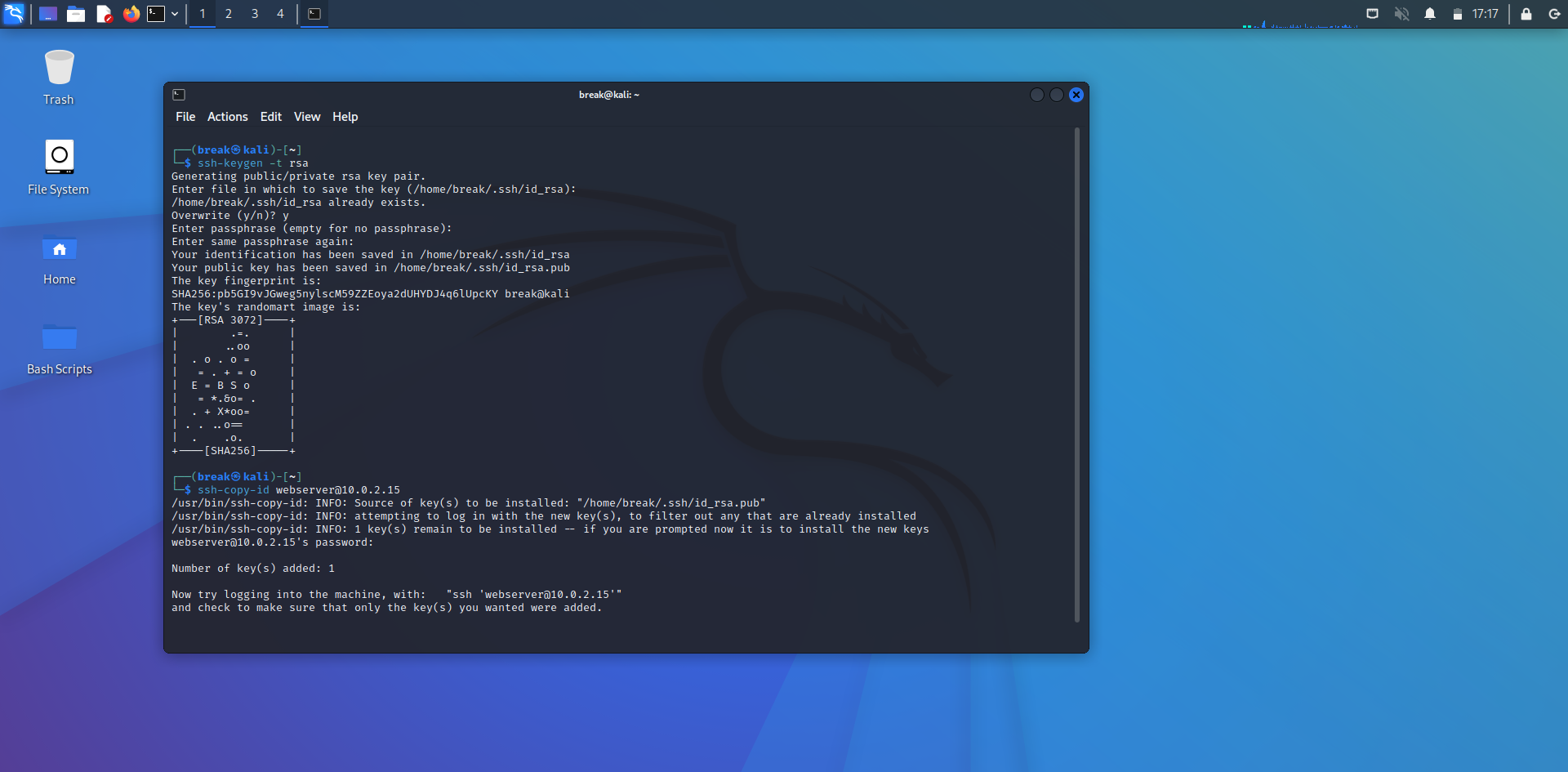


Figure 5: Creating a ssh key

1. Configure sudoers file to allow rsync to have password-less access to commands.

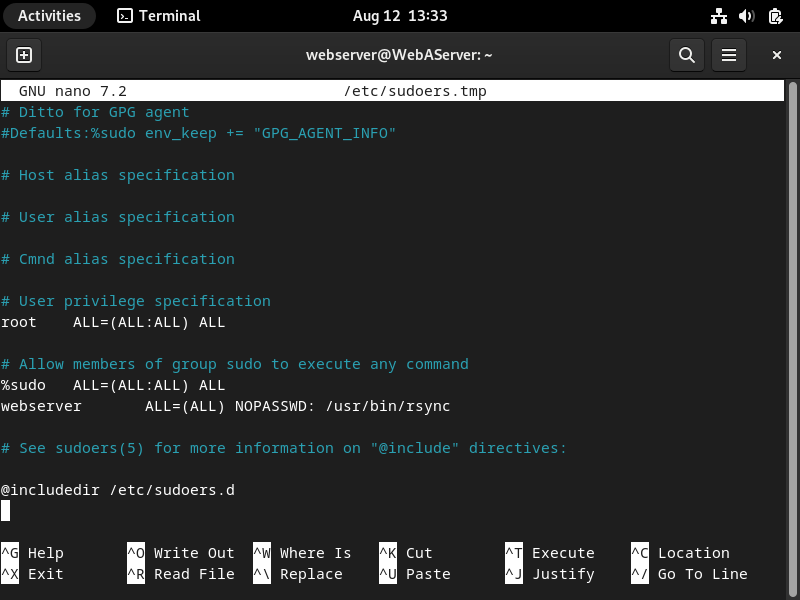


Figure 6: Configuring sudoers file

1. Write and run the bash script for log retrieval, analysis, and reporting.

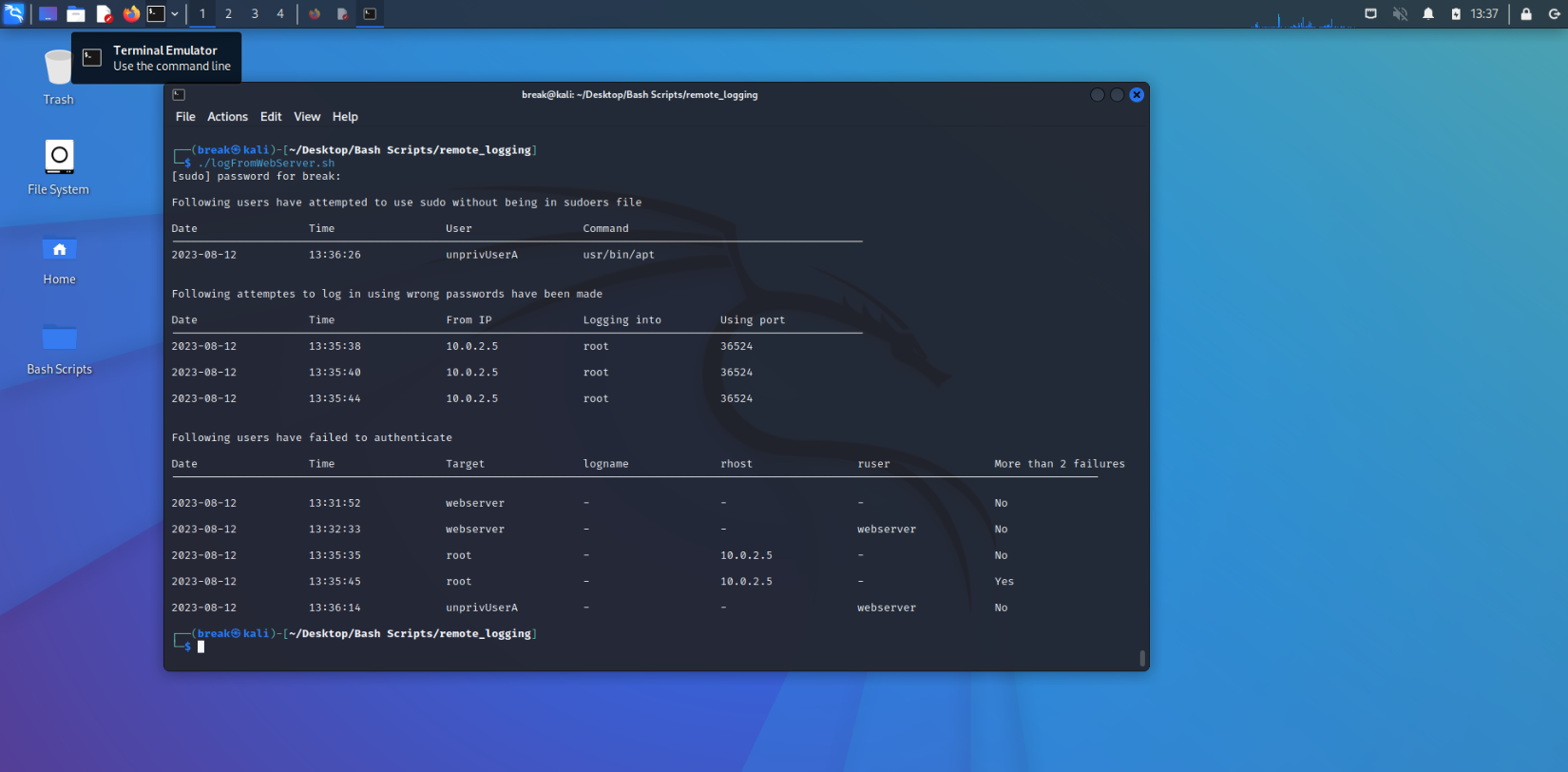


Figure 7: Log report

# 5. Bash Script

#!/bin/bash

rsync --rsync-path="sudo /usr/bin/rsync" -a webserver@10.0.2.15:/var/log/systemLogs .

rsync --rsync-path="sudo /usr/bin/rsync" -a webserver@10.0.2.15:/var/log/apache2 .

cd systemLogs

timestamp1=$(sudo sh -c 'tail -n 1 sudoFail | cut -b 1-26')

timestamp2=$(sudo sh -c 'tail -n 1 passwordFail | cut -b 1-26')

timestamp3=$(sudo sh -c 'tail -n 1 authFailure | cut -b 1-26')

if [ -e "timestampSudo" ]; then

    timestampSudo=$(sudo cat timestampSudo)

    sudoFailLogs=$(sudo awk -v timestamp=$timestampSudo 'found; $0 ~ timestamp {found=1}' sudoFail)

else

    sudoFailLogs=$(sudo cat sudoFail)

fi

sudo echo "$timestamp1" > timestampSudo

if [ -e "timestampPassword" ]; then

    timestampPassword=$(sudo cat timestampPassword)

    passwordFailLogs=$(sudo awk -v timestamp=$timestampPassword 'found; $0 ~ timestamp {found=1}' passwordFail)

else

    passwordFailLogs=$(sudo cat passwordFail)

fi

sudo echo "$timestamp2" > timestampPassword

if [ -e "timestampAuth" ]; then

    timestampAuth=$(sudo cat timestampAuth)

    authFailLogs=$(sudo awk -v timestamp=$timestampAuth 'found; $0 ~ timestamp {found=1}' authFailure)

else

    authFailLogs=$(sudo cat authFailure)

fi

sudo echo "$timestamp3" > timestampAuth

if [ ! -z "$sudoFailLogs" ]; then

    printf "\n"

    echo "Following users have attempted to use sudo without being in sudoers file"

    printf "\n%-20s %-20s %-20s %-20s\n" "Date" "Time" "User" "Command"

    echo "----------------------------------------------------------------------------------------------------------"

    while IFS= read -r line; do

        dateSudo=$(echo $line | grep -oE "([0-9]{4}-[0-9]{2}-[0-9]{2})")

        timeSudo=$(echo $line | grep -oE "T([0-9]{2}:[0-9]{2}:[0-9]{2})" | cut -c 2-)

        attemptUser=$(echo $line | grep -oP "sudo: \K.\*(?=\ : )")

        commandUsed=$(echo $line | grep -oP "COMMAND=/\K[^ ]+")

        printf "%-20s %-20s %-20s %-20s\n\n" $dateSudo $timeSudo $attemptUser $commandUsed

    done <<< "$sudoFailLogs"

else

    echo "No sudo failures since last report"

fi

if [ ! -z "$passwordFailLogs" ]; then

    printf "\n"

    echo "Following attemptes to log in using wrong passwords have been made"

    printf "\n%-20s %-20s %-20s %-20s %-30s\n" "Date" "Time" "From IP" "Logging into" "Using port"

    echo "----------------------------------------------------------------------------------------------------------"

    while IFS= read -r line; do

        dateSudo=$(echo $line | grep -oE "([0-9]{4}-[0-9]{2}-[0-9]{2})")

        timeSudo=$(echo $line | grep -oE "T([0-9]{2}:[0-9]{2}:[0-9]{2})" | cut -c 2-)

        fromIP=$(echo $line | grep -oP "from \K.\*(?=\ port )")

        usingPort=$(echo $line | grep -oP "\bport \K\d+")

        logInto=$(echo $line | grep -oP "for \K.\*(?=\ from )")

        printf "%-20s %-20s %-20s %-20s %-30s\n\n" $dateSudo $timeSudo $fromIP $logInto $usingPort

    done <<< "$passwordFailLogs"

else

    echo "No wrong password attempts since last report"

fi

if [ ! -z "$authFailLogs" ]; then

    printf "\n"

    echo "Following users have failed to authenticate"

    printf "\n%-20s %-20s %-20s %-20s %-20s %-20s %-20s\n" "Date" "Time" "Target" "logname" "rhost" "ruser" "More than 2 failures"

    echo "----------------------------------------------------------------------------------------------------------------------------------------------"

    while IFS= read -r line; do

        dateSudo=$(echo $line | grep -oE "([0-9]{4}-[0-9]{2}-[0-9]{2})")

        timeSudo=$(echo $line | grep -oE "T([0-9]{2}:[0-9]{2}:[0-9]{2})" | cut -c 2-)

        target=$(echo $line | grep -oP "(^| )user=\K\S+")

        [ -z $target ] && target="-"

        logname=$(echo $line | grep -oP "logname=\K\S+")

        [ -z $logname ] && logname="-"

        rhost=$(echo $line | grep -oP "rhost=\K\S+")

        [ -z $rhost ] && rhost="-"

        ruser=$(echo $line | grep -oP "ruser=\K\S+")

        [ -z $ruser ] && ruser="-"

        if [[ $line =~ "PAM 2 more authentication failures" ]]; then

            moreThan="Yes"

        else

            moreThan="No"

        fi

        printf "\n%-20s %-20s %-20s %-20s %-20s %-20s %-20s\n" $dateSudo $timeSudo $target $logname $rhost $ruser $moreThan

    done <<< "$authFailLogs"

else

    echo "No authentication failures since last report"

fi

# 6. Conclusion and Possible Improvements

In wrapping up, this project has been a fun journey into the world of log analysis and cybersecurity. I've dug deep into logs, uncovered security events, and put together reports that present crucial insights in a clear table format. This approach, powered by Bash scripting and smart command use, has proven effective in decoding log data in a simple yet efficient manner. I've also honed my expertise in areas such as bash scripting, configuring virtual machines, and navigating the Linux environment.

As always, there's room to make things even better and I will be working on improving this project. Visualizing the findings with graphs and charts with the help of python could add another layer of understanding. Expanding the analysis to cover different types of logs and systems could broaden our insights. And fine-tuning our automation to handle changes in log formats and settings could make our system even stronger. By learning and growing, I will try getting closer to unraveling the full story hidden within logs, making strides in the world of cybersecurity.

Last Updated: 12-08-2023

Yash Devadiga