



CySA+ Lab Series

Lab 15: Log Analysis with Bash Scripting

Document Version: **2022-10-10**

| Material in this Lab Aligns to the Following | |
|--|--|
| CompTIA CySA+ (CS0-002) Exam Objectives | 3.1 - Given a scenario, analyze data as part of security monitoring activities 3.4 - Compare and Contrast automation concepts and technologies 4.3 - Given an incident, analyze potential indicators of compromise |
| All-In-One CompTIA CySA+ Second Edition ISBN-13: 978-1260464306 Chapters | 11: Data Analysis in Security Monitoring Activities 14: Automation Concepts and Technologies 17: Analyze Potential Indicators of Compromise |

Copyright © 2022 Network Development Group, Inc.
www.netdevgroup.com

NETLAB+ is a registered trademark of Network Development Group, Inc.
KALI LINUX™ is a trademark of Offensive Security.
ALIEN VAULT OSSIM V is a trademark of AlienVault, Inc.
Microsoft®, Windows®, and Windows Server® are trademarks of the Microsoft group of companies.
Greenbone is a trademark of Greenbone Networks GmbH.
VMware is a registered trademark of VMware, Inc.
SECURITY ONION is a trademark of Security Onion Solutions LLC.
Android is a trademark of Google LLC.
pfSense® is a registered mark owned by Electric Sheep Fencing LLC ("ESF").
All trademarks, logos, and brand names are the property of their respective owners.

Contents

| | |
|--|----|
| Introduction | 3 |
| Objectives..... | 3 |
| Lab Topology | 4 |
| Lab Settings | 5 |
| 1 Getting Started with Bash | 6 |
| 1.1 Understanding Variables..... | 7 |
| 1.2 Interacting with the User | 8 |
| 1.3 Working with Arrays..... | 9 |
| 1.4 FOR Loops..... | 10 |
| 1.5 Using the IF Statement..... | 13 |
| 2 Web Server Log Files..... | 18 |
| 2.1 Collecting Log Files | 18 |
| 2.2 Use Bash Scripting to Examine the Log Files..... | 24 |
| 2.3 Examining a Squid Log Using Bash Scripting | 27 |

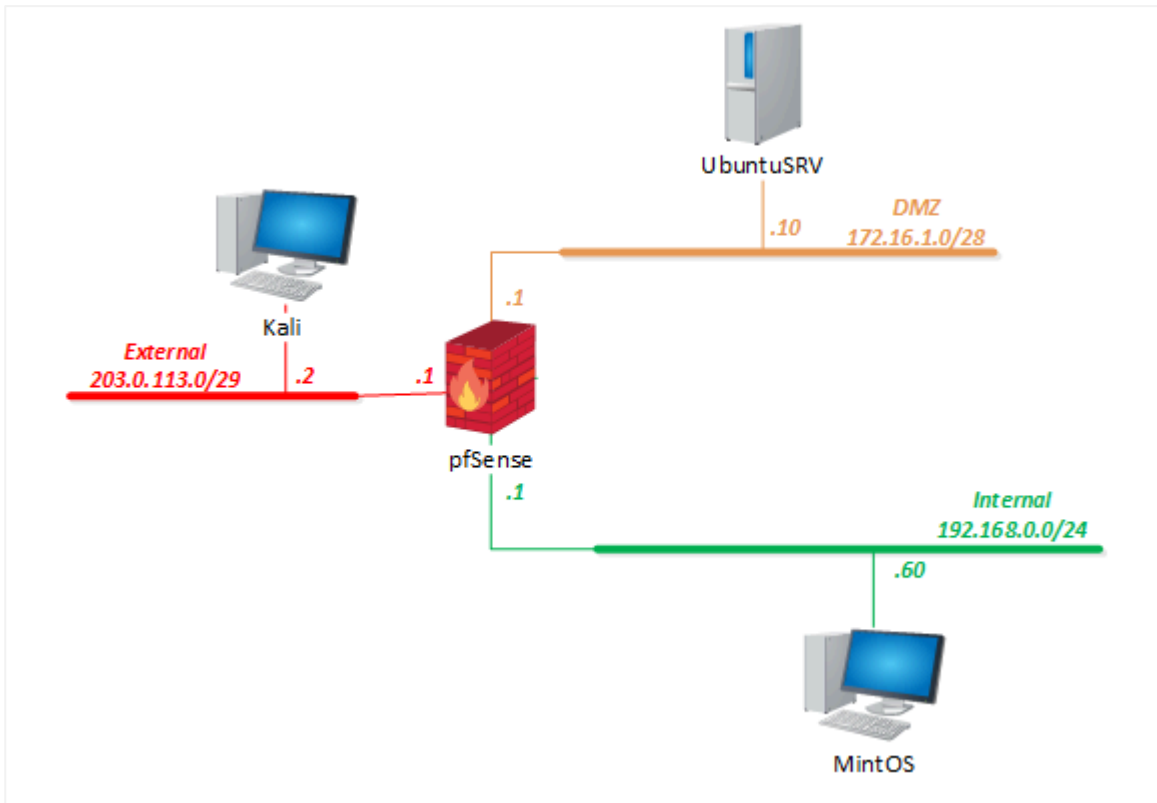
Introduction

In this lab, you will use Bash scripting to analyze logs. Determining the format of a single log record allows the Security Analyst to write scripts to dissect and look for specific user information. Studying what normal vs. abnormal traffic patterns look like in a log can go a long way toward detecting attacks.

Objectives

- Examine an Apache2 log entry
- View and reset access.log
- Create both normal and abnormal traffic to the Apache server
- Use bash scripting to examine the log files

Lab Topology



Lab Settings

The information in the table below will be needed in order to complete the lab. The task sections below provide details on the use of this information.

| Virtual Machine | IP Address | Account | Password |
|---------------------------|--|---------------|----------------|
| WinOS (Server 2019) | 192.168.0.50 | Administrator | NDGlabpass123! |
| MintOS (Linux Mint) | 192.168.0.60 | sysadmin | NDGlabpass123! |
| OSSIM (AlienVault) | 172.16.1.2 | root | NDGlabpass123! |
| UbuntuSRV (Ubuntu Server) | 172.16.1.10 | sysadmin | NDGlabpass123! |
| Kali | 203.0.113.2 | sysadmin | NDGlabpass123! |
| pfSense | 203.0.113.1 172.16.1.1 192.168.0.1 | admin | NDGlabpass123! |

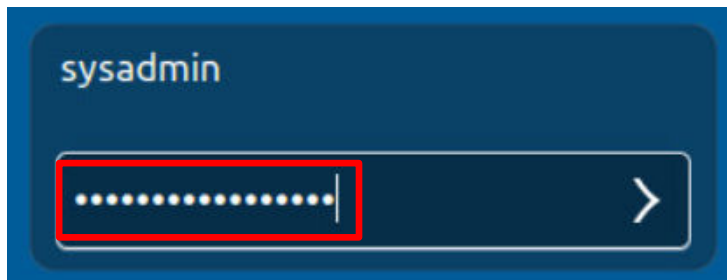
1 Getting Started with Bash

Bash is one of the command interpreters for Linux/Unix. What is a command interpreter? Simply put, a command interpreter reads and parses lines of text that were typed by a user, and if the text is a properly formatted command, it will execute the command and return the result back to the terminal. In Windows, the command interpreter is *CMD.EXE*.

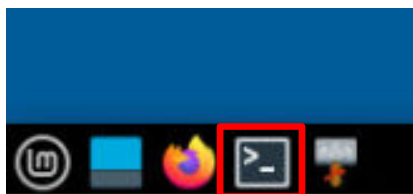
Bash is referred to as a “shell” (in fact, that’s what the name indicates, it’s the Bourne-Again Shell ... it’s an interesting story) which is just the term used to describe the interface between the user and operating system. Technically, the GUI is a type of shell as well. But, Bash is far more than just a Command Line Interpreter; it is actually a powerful programming language. What makes a programming language different than a scripting language (such as .BAT files in Windows CMD.EXE), it’s BLT ... Branching, Looping, and Testing. Bash includes the execution control structures, for, while, until, if/then, and case, as well as string and numeric variables and logical operators.

In this first part of the task, you will explore creating variables and displaying their contents on the screen.

1. Set focus on the **MintOS** computer.
2. Log in to the sysadmin using the password: NDGLabpass123!



3. Open a new terminal by clicking on the **Terminal** icon located at the bottom of the window.



4. Remain on the *MintOS* computer on the terminal window and continue to the next task.

1.1 Understanding Variables

1. Variables are a way to store a piece of data and associate it with an easy-to-read name that you can use to retrieve that data later. Think of an empty box; you can put stuff into it, see what's in the box, take out the contents and replace it with something else and even throw the stuff in the box away. The box needs a label or name to identify it. It is a best practice to choose a name that makes it easy to identify what data is stored in the variable. For example, it is easy to understand what is stored in the variable 'firstName'. To set a variable in Bash, type the following command at the terminal prompt:

```
myVariable=10
```

```
sysadmin@mintos:~$ myVariable=10
```

2. To display the contents of the variable, type the following command (Note that when *calling* a variable, you must begin with \$. This is not used when *setting* a variable.)

```
printf $myVariable
```

```
sysadmin@mintos:~$ printf $myVariable
10sysadmin@mintos:~$
```

3. Notice that the data displayed is not automatically given a new line, as it would be with the **echo** command. Type the following command to observe the difference:

```
echo $myVariable
```

```
sysadmin@mintos:~$ echo $myVariable
10
```

4. To output the character count of a string, type the following command:

```
printf ${#myVariable}
```

```
sysadmin@mintos:~$ printf ${#myVariable}
2sysadmin@mintos:~$
```

5. Notice that once more, there is no line inserted at the end of the output. To insert a line break, use the \n option at the end of the string, as below. When using *printf*, it is best practice (and often necessary) to place the printed text inside of double quotes.

```
printf "${myVariable}\n"
```

```
sysadmin@mintos:~$ printf "${myVariable}\n"
10
```

- You can store several words with line breaks into a single string. To view this behavior, type the following command:

```
test="Hello\nWorld\n"
```

```
sysadmin@mintos:~$ test="Hello\nWorld\n"
```

- Now call and print the variable with the following command:

```
printf "$test"
```

```
sysadmin@mintos:~$ printf "$test"
Hello
World
```

- Remain on the *MintOS* computer on the terminal window and continue to the next task.

1.2 Interacting with the User

In this task, you will explore how to request input from the user and how to combine that input with the output techniques you have just learned.

- To request the user input their first name into the **varFirstName** variable, type the following command in the terminal window. This command waits for your input and saves it into the **varFirstName** variable.

```
read -p "Please enter your first name: " varFirstName
```

- Type your first name and press **Enter**.

```
sysadmin@mintos:~$ read -p "Please enter your first name: " varFirstName
Please enter your first name: Jonn
```

- To request the user input their last name into a different variable (**varLastName**), type the following command.

```
read -p "Please enter your last name: " varLastName
```

- Enter your last name and press **Enter**.

```
sysadmin@mintos:~$ read -p "Please enter your last name: " varLastName
Please enter your last name: Smith
```


- Now that the user (you) has populated the variables with data, you can use this data to return a message in the form of output. Type the following command:

```
printf "Your first name is $varFirstName\nYour last name is $varLastName\n"
```

```
sysadmin@mintos:~$ printf "Your first name is $varFirstName\nYour last name is $varLastName\n"
Your first name is Jonn
Your last name is Smith
```

- Remain on the *MintOS* computer on the terminal window and continue to the next task.

1.3 Working with Arrays

In this task, you will explore arrays and how to populate them with data and print that data out using various commands. Arrays are useful for storing collections of data. Typically, an array will contain data of a similar type, such as a list of first names, phone numbers, etc. It organizes this data in contiguous memory locations for easy access. A visual representation of an array is shown below.

| Index Number | | | | | | | | | | |
|--------------|---|---|---|---|---|---|---|---|---|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| m | i | k | e | b | i | l | l | b | o | b |



The first element in the array is always at Index #0. So, an array with 5 elements would use indexes 0-4.

- To populate the array shown above, type the following command into your terminal.

```
arraySpecies=(Human Klingon Vulcan)
```

```
sysadmin@mintos:~$ arraySpecies=(Human Klingon Vulcan)
```

- When recalling data from an array, Bash identifies that input is an array rather than a variable with the use of {curly brackets}. To print out the array you just created, type the following command:

```
printf "${arraySpecies[*]}\n"
```

```
sysadmin@mintos:~$ printf "${arraySpecies[*]}\n"
Human Klingon Vulcan
```

3. To retrieve the number of elements in an array, type the following command:

```
printf "${#arraySpecies[*]}\n"
```



```
sysadmin@mintos:~$ printf "${#arraySpecies[*]}\n"
3
```

4. The use of the asterisk in the variable tells Bash that you want information regarding the entire array. The use of designated numbers will specify specific elements of the array, as shown in the diagram at the beginning of this task. To print the contents of the first element in the *arraySpecies* array, type the following command:

```
printf "${arraySpecies[0]}\n"
```



```
sysadmin@mintos:~$ printf "${arraySpecies[0]}\n"
Human
```

5. To print the contents of the second element in the *arraySpecies* array, type the following command:

```
printf "${arraySpecies[1]}\n"
```



```
sysadmin@mintos:~$ printf "${arraySpecies[1]}\n"
Klingon
```

6. To print the contents of the third and final element in the *arraySpecies* array, type the following command. Notice that each element corresponds to a piece of the output you received in step 2.

```
printf "${arraySpecies[2]}\n"
```



```
sysadmin@mintos:~$ printf "${arraySpecies[2]}\n"
Vulcan
```

7. Remain on the *MintOS* computer on the terminal window and continue to the next task.

1.4 FOR Loops

FOR loops repeat a block of commands a designated number of times. In this task, you will explore how to create FOR loops and utilize them to read a file a line at a time, or generate a list of IP addresses.

1. Type the following command to create a basic FOR loop that generates a list of numbers from 1-10 and prints each element on a separate line:

```
for index in {1..10}; do printf "$index\n"; done
```

This script is saying that for each integer from 1-10, print that variable and then a line break. Once all 10 integers have been printed, end the script. You will explore creating script files further in this lab.

```
sysadmin@mintos:~$ for index in {1..10}; do printf "$index\n"; done
1
2
3
4
5
6
7
8
9
10
```



The command above executes a script with one string of input. The semicolons denote a separation of commands. If you were creating the script in a file that will be executed, the input would look something like:

```
for index in {1..10}
do
    printf "$index\n"
done
```

2. To build an array and print its contents on separate lines using a FOR loop, type the following command:

```
arrayLOTR=(One Ring to Rule Them All); for index in ${arrayLOTR[*]}; do
printf "$index\n"; done
```

```
sysadmin@mintos:~$ arrayLOTR=(One Ring to Rule Them All); for index in ${arrayLOTR[*]};
do printf "$index\n"; done
One
Ring
to
Rule
Them
All
```

- Next, you will explore how to read a file's contents one line at a time utilizing a FOR loop. First, you must gain root access. Type the following command, typing the password **NDGlabpass123!** when prompted.

```
sudo su
```

```
sysadmin@mintos:~$ sudo su
[sudo] password for sysadmin:
root@mintos:/home/sysadmin#
```

- Now that you have root privileges, type the following command and examine the output received. This command reads each line of the **/etc/shadow** file and prints them on the terminal, one line at a time.

```
for line in $(cat /etc/shadow); do printf "$line\n"; done
```

```
root@mintos:/home/sysadmin# for line in $(cat /etc/shadow); do printf "$line\n"; done
root:$6$GMYKL2VJRokuZKF$RPEBQZFXFm9QT0zQJigQ//HjrzEofcTNL/w9d7okt17U0Fw/y2y2ZLDbsq.PxRH
80sFr11Bb14QIX1gfGl8se0:18942:0:99999:7:::
daemon*:18811:0:99999:7:::
bin*:18811:0:99999:7:::
sys*:18811:0:99999:7:::
sync*:18811:0:99999:7:::
games*:18811:0:99999:7:::
man*:18811:0:99999:7:::
lp*:18811:0:99999:7:::
mail*:18811:0:99999:7:::
news*:18811:0:99999:7:::
uucp*:18811:0:99999:7:::
proxy*:18811:0:99999:7:::
www-data*:18811:0:99999:7:::
backup*:18811:0:99999:7:::
list*:18811:0:99999:7:::
irc*:18811:0:99999:7:::
gnats*:18811:0:99999:7:::
nobody*:18811:0:99999:7:::
systemd-network*:18811:0:99999:7:::
systemd-resolve*:18811:0:99999:7:::
systemd-timesync*:18811:0:99999:7:::
```

- To exit *root*, type the following command:

```
exit
```

```
root@mintos:/home/sysadmin# exit
exit
sysadmin@mintos:~$
```



It is best not to keep root privileges on so that you do not cause accidental harm to your system. Once you perform the tasks you entered root to do, it is best practice to turn off root privileges immediately afterward.

6. Remain on the *MintOS* computer on the terminal window and continue to the next task.

1.5 Using the IF Statement

The branching part of the BLT (Branching/Looping/Testing) is largely handled by the IF statement. IF statements perform conditional operations in which the evaluation of a Boolean expression (true or false) will determine which block of code to execute. In this task, you will explore how to apply IF statements that check for the Boolean status of data. You will then apply IF statements to create files, check if users exist, and test the number of lines in files.

1. To use Boolean logic in IF statements, you must first understand how Bash reads Boolean values. The commands **true** and **false** set an exit value of 0 if the expression evaluates as true or sets it to 1 if it evaluates as false. To see the returned exit value for the **true** command, type the following command:

```
true; echo $?
```

```
sysadmin@mintos:~$ true; echo $?  
0
```

2. The above input was comprised of two separate commands; the **true** command sets the exit status value to 0, and the **echo \$?** command returned the contents of the **\$?** variable, which stores the return value of the last command that was executed. To see the returned exit value for the **false** command, type the following command:

```
false; echo $?
```

```
sysadmin@mintos:~$ false; echo $?  
1
```

3. You can view these exit values in a practical application using the **ls** (file list) command. Type the following **ls** command to return an exit value for a file that exists:

```
ls /etc/shadow; echo $?
```

```
sysadmin@mintos:~$ ls /etc/shadow; echo $?  
/etc/shadow  
0
```

4. Type the following **ls** command to return an exit value for a file that does not exist:

```
ls nopassword_file.txt; echo $?
```

```
sysadmin@mintos:~$ ls shadow; echo $?  
ls: cannot access 'shadow': No such file or directory  
2
```



The output for the previous command returned a 2 for an exit value. This occurs when there was an error, rather than a true or false. In this case, the **ls** command could not locate the file, as it did not exist, and returned an error message.

IF statements are based on an if/then/else structure. The format for this is typically:

if (Boolean statement)

<run commands>

else

<run command>

5. To utilize this logic, first create a file containing a list of file paths by typing the following command. Note that the last file path (**mirage**) does not exist:

```
echo /etc/passwd >> my_test_files.txt; echo /etc/group >> my_test_files.txt; echo  
/etc/shadow >> my_test_files.txt; echo /etc/mirage >> my_test_files.txt
```

```
sysadmin@mintos:~$ echo /etc/passwd >> my_test_files.txt; echo /etc/group/ >> my_test_files.txt;  
echo /etc/shadow >> my_test_files.txt; echo /etc/mirage >> my_test_files.txt
```

6. To confirm the contents of the newly created **my_test_files.txt** file, type the following command. Notice that all four paths are returned.

```
cat my_test_files.txt
```

```
sysadmin@mintos:~$ cat my_test_files.txt  
/etc/passwd  
/etc/shadow  
/etc/group  
/etc/mirage
```

7. You can now use an IF statement that will check each line of **my_test_files.txt**. It then checks to see if each file exists. If the file exists (*true*), it will print one message; else (*false*) it will print a different message. Type the following command:

```
for file in $(cat my_test_files.txt); do if [ -a $file ]; then printf "The file $file exists\n"; else printf "The file $file does not exist\n"; fi done
```

```
sysadmin@mintos:~$ for file in $(cat my_test_files.txt); do if [ -a $file ]; then printf "The file $file exists\n"; else printf "The file $file does not exist\n"; fi done
The file /etc/passwd exists
The file /etc/group exists
The file /etc/shadow exists
The file /etc/mirage does not exist
```



Spacing is important. There is a space before and after the statement between the `[]`.

- To create a Bash shell script containing these commands, use an editor to create the file, beginning with the following command:

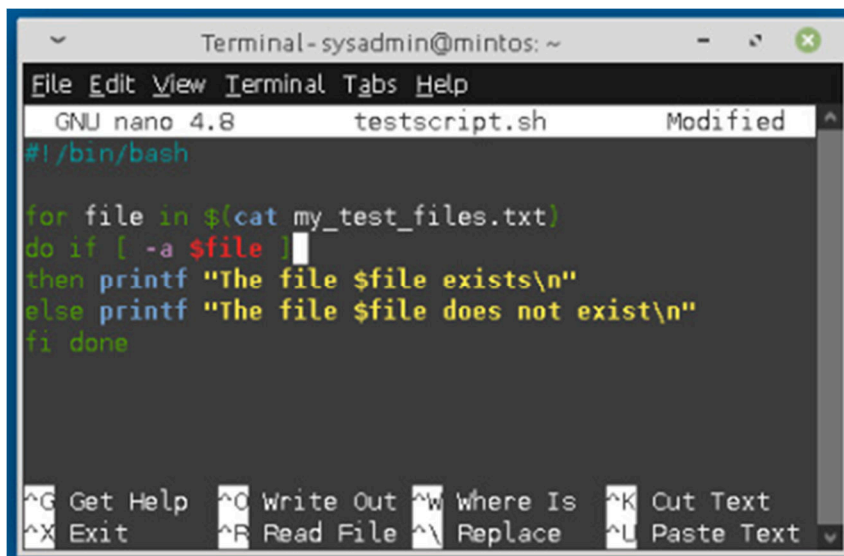
```
nano testscript.sh
```

```
sysadmin@mintos:~$ nano testscript.sh
```

- Type the following commands from the previous step into the Bash shell script, separating lines where semicolons exist. You must start the document with `#!/bin/bash`. This tells the shell how to run the script.

```
#!/bin/bash

for file in $(cat my_test_files.txt)
do if [ -a $file ]
then printf "The file $file exists\n"
else printf "The file $file does not exist\n"
fi done
```

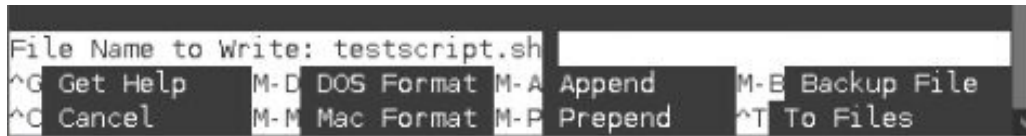


```
Terminal-sysadmin@mintos: ~
File Edit View Terminal Tabs Help
GNU nano 4.8 testscript.sh Modified
#!/bin/bash

for file in $(cat my_test_files.txt)
do if [ -a $file ]
then printf "The file $file exists\n"
else printf "The file $file does not exist\n"
fi done

^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^L Paste Text
```


10. When finished, press **Ctrl+O** to save the file. Press **Enter** when prompted to write the file out using the same file name.



11. Press **Ctrl-X** to exit from the *nano* editor.
12. Now that the script has been created, you must give yourself permission to use the file. Do so with the following command:

```
chmod u+x testscript.sh
```

```
sysadmin@mintos:~$ chmod u+x testscript.sh
```

13. Now you can run the script by typing the following command:

```
./testscript.sh
```

```
sysadmin@mintos:~$ ./testscript.sh
The file /etc/passwd exists
The file /etc/shadow exists
The file /etc/group exists
The file /etc/mirage does not exist
```

There are many ways to utilize FOR loop functionality. In the following steps, you will explore a few different applications.

14. To run a script that will check if a user exists, type the following command:

```
if id -u sysadmin &> /dev/null; then printf "User Exists\n";  
else printf "User Does Not Exist\n"; fi
```

```
sysadmin@mintos:~$ if id -u sysadmin &> /dev/null; then printf "User Exists\n";  
else printf "User Does Not Exist\n"; fi  
User Exists
```

15. The results returned a positive because sysadmin exists. To trigger the response when the script returns a 1 (failure/false) rather than a 0 (success/true), type the command again using the username **student**:

```
if id -u student &> /dev/null; then printf "User Exists\n";  
else printf "User Does Not Exist\n"; fi
```

```
sysadmin@mintos:~$ if id -u student &> /dev/null; then printf "User Exists\n";  
else printf "User Does Not Exist\n"; fi  
User Does Not Exist
```

16. To create a script to check whether the number of lines in a file exceeds 50, type the following script:

```
if [ $(wc -l /etc/passwd | cut -d " " -f 1) -gt "50" ]; then printf "Is > 50\n";  
else printf "Not > 50\n"; fi
```

```
sysadmin@sysadmin-virtual-machine:~$ if [ $(wc -l /etc/passwd | cut -d " " -f  
1) -gt "50" ]; then printf "Is > 50\n"; else printf "Not > 50\n"; fi  
Not > 50
```

17. To see an example of a script that will test if you are the root, type the following command. As you are not currently logged in with root privileges, the command will return an exit status of **1**, indicating **False**.

```
if [ $(whoami) != "root" ]; then printf "I am not root\n"; else printf "I am  
root\n"; fi
```

```
sysadmin@sysadmin-virtual-machine:~$ if [ $(whoami) != "root" ]; then printf  
"I am not root\n"; else printf "I am root\n"; fi  
I am not root
```

18. Close the terminal session on the *MintOS* computer.

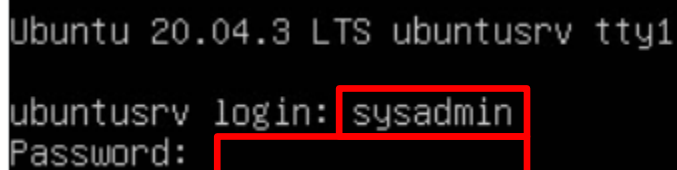
2 Web Server Log Files

Web browsers can provide a great deal of feedback about their activities, performance, and problems using comprehensive logging capabilities. The Web Master is charged with looking at all of these logs to glean statistical information from the web server.

The security analyst has a much more specific reason to analyze the log files, detecting abnormal, sometimes malicious, traffic.

2.1 Collecting Log Files

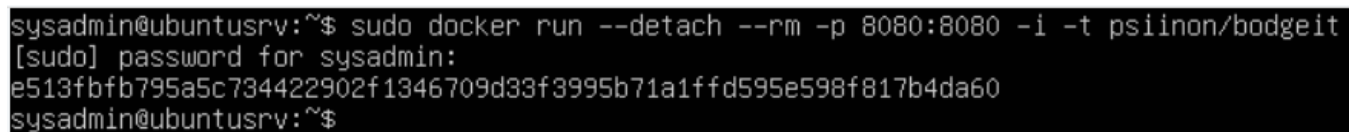
1. Set the focus to the **UbuntuSRV** computer.
2. Log in as sysadmin using the password: NDGLabpass123!



```
Ubuntu 20.04.3 LTS ubuntu:~$ ssh ubuntu:~$  
ubuntu login: sysadmin  
Password: 
```

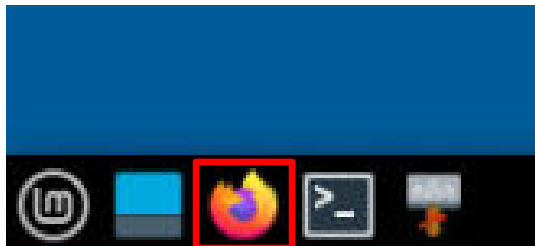
3. Begin by starting the *Bodgeit* website inside a *Docker* container. A Docker container is a form of virtualization that utilizes the OS in order to allow software to run inside of an isolated, virtual instance in any Linux environment. In order to start the **Bodgeit Docker** container, type the following command, using the password NDGLabpass123! when prompted:

```
sudo docker run --detach --rm -p 8080:8080 -i -t psiinon/bodgeit
```

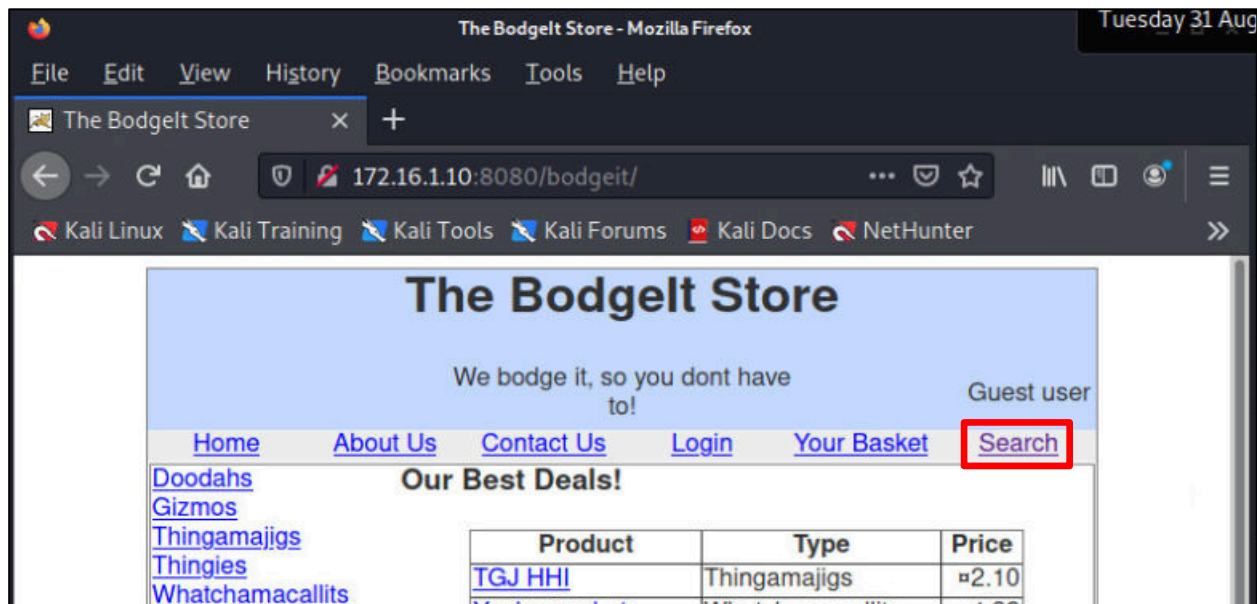


```
sysadmin@ubuntu:~$ sudo docker run --detach --rm -p 8080:8080 -i -t psiinon/bodgeit  
[sudo] password for sysadmin:  
e513fbfb795a5c734422902f1346709d33f3995b71a1ffd595e598f817b4da60  
sysadmin@ubuntu:~$
```

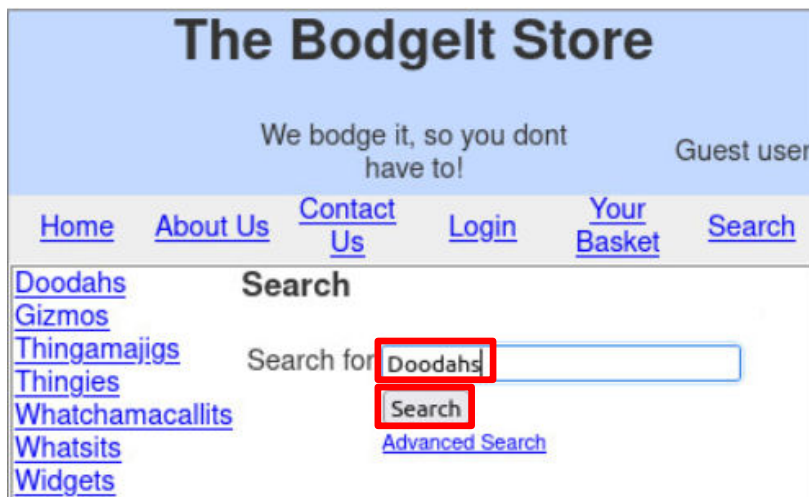
4. Set the focus back to the **MintOS** computer.
5. Open the **Web Browser** application on the taskbar.



6. Navigate to **http://172.16.1.10:8080/bodgeit/**. Confirm that the website has successfully loaded and click the **Search** button.



7. In the search box, enter Doodahs and click the **Search** button.



The list of *Doodahs* will be shown.

| Doodahs | Search |
|----------------------------------|---|
| Gizmos | You searched for: Doodahs |
| Thingamajigs | |
| Thingies | |
| Whatchamacallits | |
| Whatsits | |
| Widgets | |
| | |
| Product | Description |
| Zip a dee doo dah | Y oaghhh dufbq xrn apcbbn screnr umy nwpewf jcd nu fe ia rsodl cakggn. lvm apnspm u o. Q fpw uef lcvuvq hkbixmn plctu rbpq laiv maa |

8. Click the **Search** box again. This time search for *Gizmos*.

The Bodgelt Store

We bodge it, so you dont have to!

Guest user

[Home](#)
[About Us](#)
[Contact Us](#)
[Login](#)
[Your Basket](#)
[Search](#)

Search

Search for

[Advanced Search](#)

[Doodahs](#)
[Gizmos](#)
[Thingamajigs](#)
[Thingies](#)
[Whatchamacallits](#)
[Whatsits](#)
[Widgets](#)

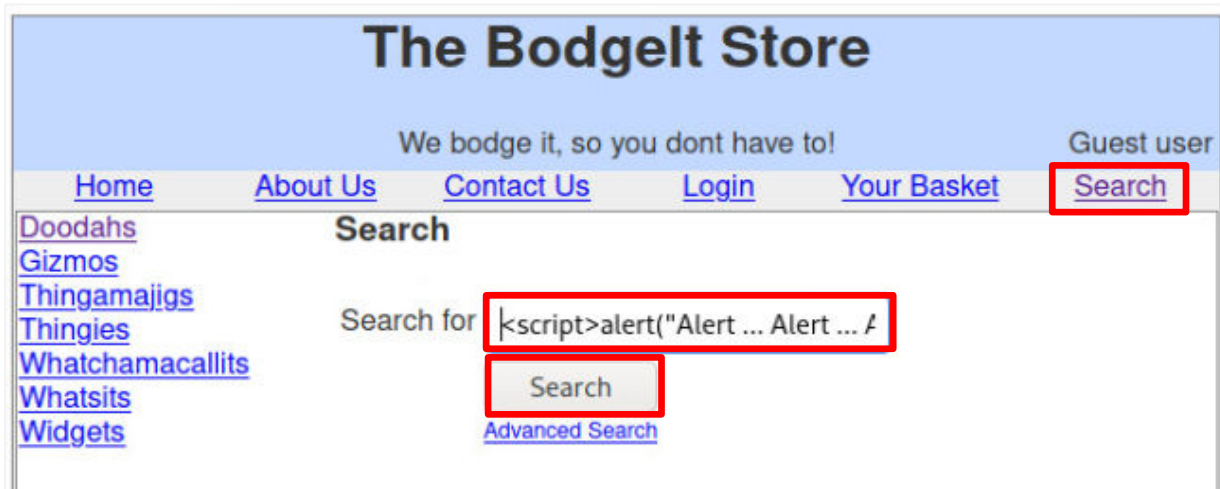
The list of *Gizmos* will be shown.

| Doodahs | Search |
|----------------------------------|--|
| Gizmos | You searched for: Gizmos |
| Thingamajigs | |
| Thingies | |
| Whatchamacallits | |
| Whatsits | |
| Widgets | |
| | |
| Product | Description |
| GZ XT4 | Kyjfev mlrahq gvaq n co mul vgpisc adxrm lxgxi. lrgkul m dyu rtxm eugv |

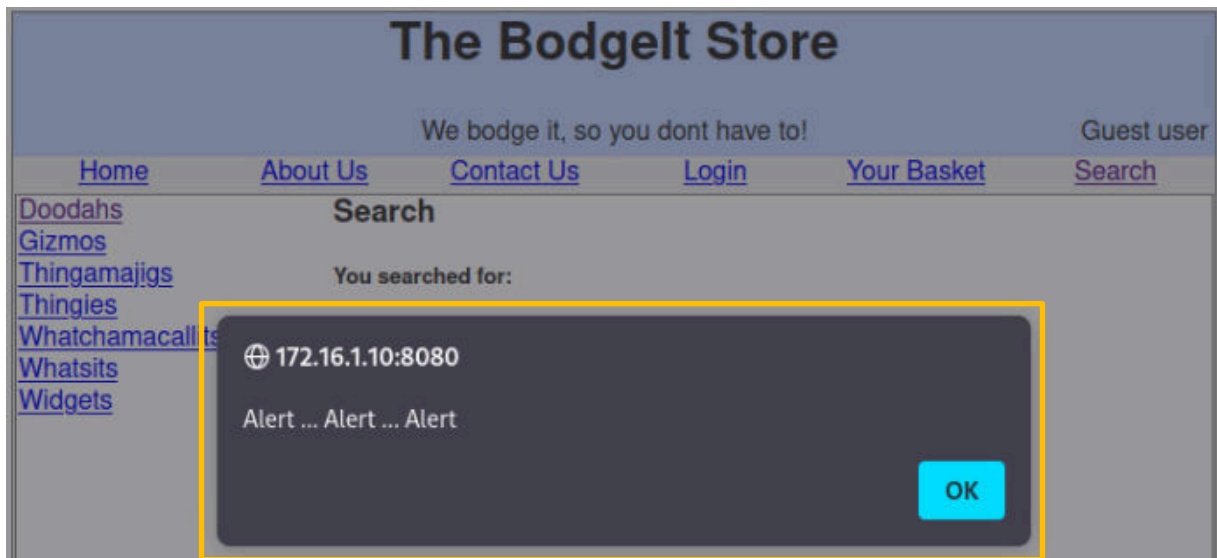
9. This website has a Cross-Site Scripting vulnerability. To execute a malicious script, inject a simple popup alert into the search function. Click the **Search** button in the top-right and type into the *Search For* field:

```
<script>alert("Alert ... Alert ... Alert")</script>
```

Then, click the **Search** button below the *Search For* box to execute the Cross-Site Script.



The response to the search is the *Cross-Side Script* popup box.



10. Close the web browser.
11. Since the vulnerable web server is running under *Docker*, you will need to access the underlying *Docker* shell. Return to the *UbuntuSRV* computer.
12. First, you will need to establish the *Docker* **Container ID**. At the **UbuntuSRV** prompt, type the following command:

```
sudo docker ps
```

If asked for the **[sudo]** password for **sysadmin**, type: **NDGlabpass123!**

```
sysadmin@ubuntusrv:~$ sudo docker ps
[sudo] password for sysadmin:
CONTAINER ID   IMAGE          COMMAND                  CREATED
2a7654dee4cf   psiinon/bodgeit "catalina.sh run"       16 minutes ago
8080/tcp, :::8080->8080/tcp   goofy_mayer
```

Make a note of the *Container ID*.



The Container ID will be different every time you start the *Docker* container

13. Type the container's shell by typing the command **sudo docker exec -it <Container ID>**:

```
sudo docker exec -it 2a7654dee4cf /bin/bash
```

```
sysadmin@ubuntusrv:~$ sudo docker exec -it 2a7654dee4cf /bin/bash
root@2a7654dee4cf:/usr/local/tomcat#
```

14. Note that the prompt now indicates that you are in the Docker container's shell. Change to the **logs** subdirectory by typing the command:

```
cd logs
```

```
root@2a7654dee4cf:/usr/local/tomcat# cd logs
```

15. List the files in the logs directory by typing the command:

```
ls -l
```

```
root@2a7654dee4cf:/usr/local/tomcat/logs# ls -l
total 16
-rw-r--r-- 1 root root 6721 Dec 31 00:14 catalina.2021-12-31.log
-rw-r--r-- 1 root root    0 Dec 31 00:14 host-manager.2021-12-31.log
-rw-r--r-- 1 root root  418 Dec 31 00:14 localhost.2021-12-31.log
-rw-r--r-- 1 root root  791 Dec 31 00:21 localhost_access_log.2021-12-31.txt
-rw-r--r-- 1 root root    0 Dec 31 00:14 manager.2021-12-31.log
```

The access log files will be the file name that begins with **localhost_access_log**. The remaining part of the name is the date that the *Docker* container's web server was started.

16. Display the log file with the following command (the **date stamp** at the end of the file name will be different):

```
cat localhost_access_log.2021-12-31.txt
```

```
root@2a7654dee4cf:/usr/local/tomcat/logs# cat localhost_access_log.2021-12-31.txt
203.0.113.2 - - [31/Dec/2021:00:16:42 +0000] "GET /bodgeit/ HTTP/1.1" 200 3177
203.0.113.2 - - [31/Dec/2021:00:16:42 +0000] "GET /bodgeit/style.css HTTP/1.1" 200 475
203.0.113.2 - - [31/Dec/2021:00:16:48 +0000] "GET /bodgeit/search.jsp HTTP/1.1" 200 2216
203.0.113.2 - - [31/Dec/2021:00:17:10 +0000] "GET /bodgeit/search.jsp?q=Doodahs HTTP/1.1" 200 2783
203.0.113.2 - - [31/Dec/2021:00:19:34 +0000] "GET /bodgeit/search.jsp HTTP/1.1" 200 2216
203.0.113.2 - - [31/Dec/2021:00:19:46 +0000] "GET /bodgeit/search.jsp?q=Gizmos HTTP/1.1" 200 2805
203.0.113.2 - - [31/Dec/2021:00:20:57 +0000] "GET /bodgeit/search.jsp HTTP/1.1" 200 2216
203.0.113.2 - - [31/Dec/2021:00:21:32 +0000] "GET /bodgeit/search.jsp?q=%3Cscript%3Ealert%28%22Alert
+...+Alert+...+Alert%22%29%3C%2Fscript%3E HTTP/1.1" 200 2056
```

Note the last request on the list shows a request to execute a JSP command to display an alert box... which is the Cross-Site Script.

17. Exit from the *Docker* container shell by typing the following command:

```
exit
```

```
root@2a7654dee4cf:/usr/local/tomcat/logs# exit
exit
sysadmin@ubuntusrv:~$ _
```

18. You need to copy the access log file to the working directory, which is represented by a period, on the *UbuntuSRV* system.

Use the **cp** (copy) command to copy the file from within the *Docker* container. The format of the file in the Docker container is

<Container ID>:/usr/local/tomcat/logs/localhost_access_log.<date stamp>.txt .

The name of the log file will be different based on the date.

If asked for the **[sudo]** password for **sysadmin**, use: **NDGlabpass123!**

```
sudo docker cp 2a7654dee4cf:/usr/local/tomcat/logs/localhost_access_log.2021-12-31.txt .
```

The **date stamp** at the end of the file name will be different, and there is a period at the end of the command, which will copy the file to the working directory on the *UbuntuSRV* system.

```
sysadmin@ubuntusrv:~$ sudo docker cp 2a7654dee4cf:/usr/local/tomcat/logs/localhost_access_log.2021-12-31.txt .
```

19. To confirm the file has been copied, type the following command:

```
ls -l
```

```
sysadmin@ubuntusrv:~$ ls -l
total 4
-rw-r--r-- 1 root root 791 Dec 31 00:21 localhost_access_log.2021-12-31.txt
```

20. Remain on the *UbuntuSRV* computer and continue to the next task.

2.2 Use Bash Scripting to Examine the Log Files

A web server's access log can have hundreds, even thousands of entries. Obviously, it could take many hours to scour the logs one line at a time, so by using the instructions on Bash scripting from the first part of the lab, a security analyst can leverage Bash commands to more quickly examine and report on attacks by bad actors exploiting vulnerabilities, such as Cross-Site Scripting. With this information, corrective actions can be taken.

Even with a small log, like the one generated in the previous step, you can still use Bash commands to identify suspicious activity in the log.

1. Display the log file you copied over with the following command (the **date stamp** at the end of the file name will be different):

```
cat localhost_access_log.2021-12-31.txt
```

```
root@2a7654dee4cf:/usr/local/tomcat/logs# cat localhost_access_log.2021-12-31.txt
203.0.113.2 - - [31/Dec/2021:00:16:42 +0000] "GET /bodgeit/ HTTP/1.1" 200 3177
203.0.113.2 - - [31/Dec/2021:00:16:42 +0000] "GET /bodgeit/style.css HTTP/1.1" 200 475
203.0.113.2 - - [31/Dec/2021:00:16:48 +0000] "GET /bodgeit/search.jsp HTTP/1.1" 200 2216
203.0.113.2 - - [31/Dec/2021:00:17:10 +0000] "GET /bodgeit/search.jsp?q=Doodahs HTTP/1.1" 200 2783
203.0.113.2 - - [31/Dec/2021:00:19:34 +0000] "GET /bodgeit/search.jsp HTTP/1.1" 200 2216
203.0.113.2 - - [31/Dec/2021:00:19:46 +0000] "GET /bodgeit/search.jsp?q=Gizmos HTTP/1.1" 200 2805
203.0.113.2 - - [31/Dec/2021:00:20:57 +0000] "GET /bodgeit/search.jsp HTTP/1.1" 200 2216
203.0.113.2 - - [31/Dec/2021:00:21:32 +0000] "GET /bodgeit/search.jsp?q=%3Cscript%3Ealert%28%22Alert%22%29%3C%2Fscript%3E HTTP/1.1" 200 2056
```

There are several things you can immediately discover. In the *URL* string *search.jsp*, there is *?q=* query string that is passed from the text typed into the *Search* field, which is the value in the database from the web page. In the case of a normal query, a string is passed, in this case, **Doodahs** and **Gizmos**. But in the highlighted box, you can see it was the injected script. In many cases, a **Cross-Site Script** command will begin with a **%** followed by a number, which represents the hexadecimal value of special characters. In this case **%3C** represents a **<** sign.



"So what do you look for? Anything out of the ordinary. How do you know what is ordinary? By checking your logs frequently to establish what is ordinary."

Here is one sample web access log entry that is a sign of an XSS attack.

```
192.168.0.252 -- [05/Aug/2009:15:16:42 -0400] "GET /%27%27;!--%22%3CXSS%3E=&{}} HTTP/1.1" 404 310 "-" "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.0.12)Gecko/2009070812 Ubuntu/8.04 (hardy) Firefox/3.0.12"
```

*The part to look for is the **GET /%27%27** command (there are several variants)."*

*<https://security.tcnj.edu/resources-tips/resources-for-server-administrators-and-developers/detecting-cross-site-scripting-attacks/>
Referenced from: ha.ckers.org and BeginLinux.com blog*

- At the **sysadmin@ubuntusrv:~\$** prompt, use the Bash command to find specific instances of the text **?q=%** by typing the following (the **date stamp** at the end of the file name will be different):

```
grep ?q=% localhost_access_log.2021-12-31.txt
```

```
sysadmin@ubuntusrv:~$ grep ?q=% localhost_access_log.2021-12-31.txt
203.0.113.2 - - [31/Dec/2021:00:21:32 +0000] "GET /bodgeit/search.jsp?q=%3Cscript%3Ealert%28%22Alert+...+Alert+...+Alert%22%29%3C%2Fscript%3E HTTP/1.1" 200 2056
```

The **grep** command (*globally search for a regular expression and print matching line*) looks for instances of text in a given file (or files) and displays the results. So, focusing on the discovered text, you will see:

```
"GET /bodgeit/search.jsp?q=%3Cscript%3Ealert
```

If there were multiple instances of the text, they would all be displayed.



In the early days of Unix (starting with Version 7), several variants of the **grep** command were implemented, namely, **egrep** and **fgrep**.

*"The **"egrep"** variant supports an extended regular expression syntax added by Alfred Aho after Ken Thompson's original regular expression implementation. The **"fgrep"** variant searches for any of a list of fixed strings using the Aho–Corasick string matching algorithm. Binaries of these variants persist in most modern systems, usually linking to **grep**; however, using these binaries has been deprecated in favor of using the **-E** and **-F** command-line switches of **grep** instead."*

<https://en.wikipedia.org/wiki/Grep>

- From a security administrator's perspective, it would be more productive to instance the log file and filter the **grep** text search on the file. This can easily be done using the **|** (pipe) operator. This operator takes the output of one Bash statement and redirects it into another Bash statement. In the case of our example, you can take the results of the **cat** command (the list of log entries) and then pipe them through **grep** to find the text matches. Type the following statement (the **date stamp** at the end of the file name will be different):

```
cat localhost_access_log.2021-12-31.txt | grep ?q=%
```

```
sysadmin@ubuntusrv:~$ cat localhost_access_log.2021-12-31.txt | grep ?q=%  
203.0.113.2 - - [31/Dec/2021:00:21:32 +0000] "GET /bodgeit/search.jsp?q=%3Cscript
```

- The next step is to redirect the output of the **grep** search to a file where it can be used for analysis, mitigation, investigation, and reporting. Type the following command to **cat** the log file, **grep** for the text and then redirect the output to a file called **xss_found.txt**:

```
cat localhost_access_log.2021-12-31.txt | grep ?q=% > xss_found.txt
```

```
sysadmin@ubuntusrv:~$ cat localhost_access_log.2021-12-31.txt | grep ?q=% > xss_found.txt
```

- Run a directory list to see if the file has been created

```
dir
```

```
sysadmin@ubuntusrv:~$ dir  
localhost_access_log.2021-12-31.txt  xss_found.txt
```

6. List out the contents of the **xss_found.txt** file.

```
cat xss_found.txt
```

```
sysadmin@ubuntu:~$ cat xss_found.txt
203.0.113.2 - - [31/Dec/2021:00:21:32 +0000] "GET /bodgeit/search.jsp?q=%3Cscript%3Ealert%28%22Alert+...+Alert+...+Alert%22%29%3C%2Fscript%3E HTTP/1.1" 200 2056
```

7. The next task will be done on the *Kali* machine.

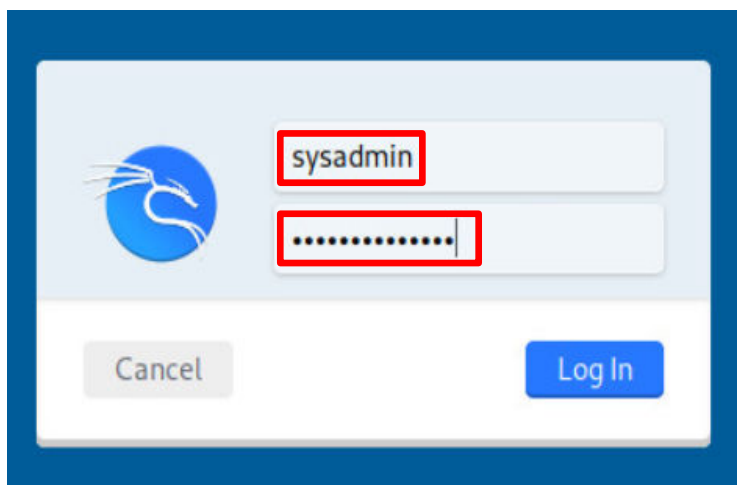
2.3 Examining a Squid Log Using Bash Scripting

Squid is a cache and forwarding web proxy. It was released in 1996 under the GNU General Public License, and it can be easily installed on Unix/Linux systems. It also has been incorporated into many firewall appliances, such as *pfSense*. Its main function is to improve website access by caching documents from websites and providing access for repeated requests resulting in faster browser response and lessening internet traffic.

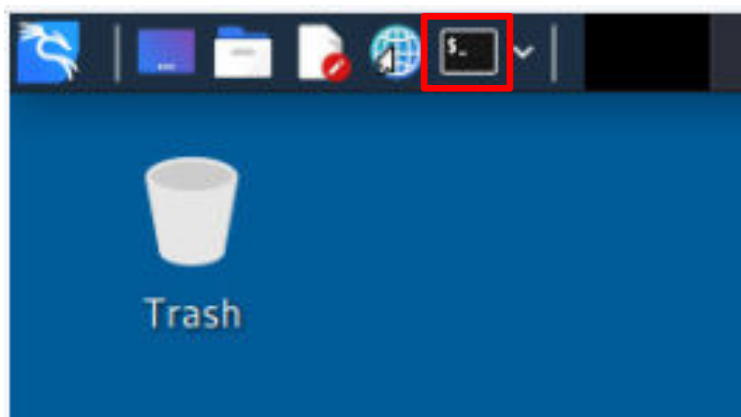
It also keeps a detailed log of the websites that are requested from hosts through their browsers. This information can be very valuable, not only to monitor what websites are being accessed and by whom but it is also used by a Security Administrator to track down the sources of malicious activity.

In this section, you will use the *Bash* command **grep** to analyze a *Squid* Log.

1. Set the focus to the **Kali** computer.
2. Log in as **sysadmin** using the password: **NDGLabpass123!**



3. Open the **Terminal** icon on the taskbar.



4. A *Squid* log has already been saved in the *LabFiles* folder for analysis. In the terminal window, type the following command to view the Squid Log:

```
cat Desktop/LabFiles/Logs/access.log | more
```

```
(sysadmin@kali)-[~]
$ cat Desktop/LabFiles/Logs/access.log | more
1157689312.049 5006 10.105.21.199 TCP_MISS/200 19763 CONNECT login.yahoo.com 4
43 badeyek DIRECT/209.73.177.115 -
1157689320.327 2864 10.105.21.199 TCP_MISS/200 10182 GET http://www.goonernews
.com/ badeyek DIRECT/207.58.145.61 text/html
1157689320.343 1357 10.105.21.199 TCP_REFRESH_HIT/304 214 GET http://www.goone
rnews.com/styles.css badeyek DIRECT/207.58.145.61 -
1157689321.315 1 10.105.21.199 TCP_HIT/200 1464 GET http://www.goonernews.c
om/styles.css badeyek NONE/- text/css
```


The log shows all of the information about the web request/response. There are several fields that are of importance to a security analyst:

- The first field is the timestamp (the number of seconds, with millisecond resolution since the Unix epoch, which is Thu Jan 1 00:00:00 UTC 1970). This can be converted to local time in many ways.
 - The second field is the IP address of the host that made the request. Using this information, the Security analyst can identify which host made the request (unfortunately, it cannot identify the user, but that could be gleaned from cross-referencing user access logs.)
 - The third field is the website that was requested. *This is the first thing to be checked.*
5. Press the **Spacebar** to advance to the next screen of log entries. Continue advancing and take a quick look at each screen. Since there are 1,643,250 lines, it would be very impractical to find interesting information in this way. The **grep** command would be a good way of filtering this traffic. Press **q** to exit from the **cat** function.

Windows is constantly calling back to Microsoft to check for updates, sending telemetry, reporting on web activity, etc. A Security Analyst would want to know how often and what kind of traffic is sent back. Using the **grep** command on the access log and **pipe** it to a file that could then be analyzed would be a good way to start.

6. In the *Kali* terminal, use the **cat** command to list the file and **|** (pipe) the log entries through **grep** and then **>** (redirect) the output to a file by typing the following command. It should only take a few seconds for the processing of the access log.


```
cat Desktop/LabFiles/Logs/access.log | grep microsoft.com > Desktop/MS-Activity.txt
```



```
(sysadmin@kali)-[~]  
$ cat Desktop/LabFiles/Logs/access.log | grep microsoft.com > Desktop/MS-Activity.txt
```

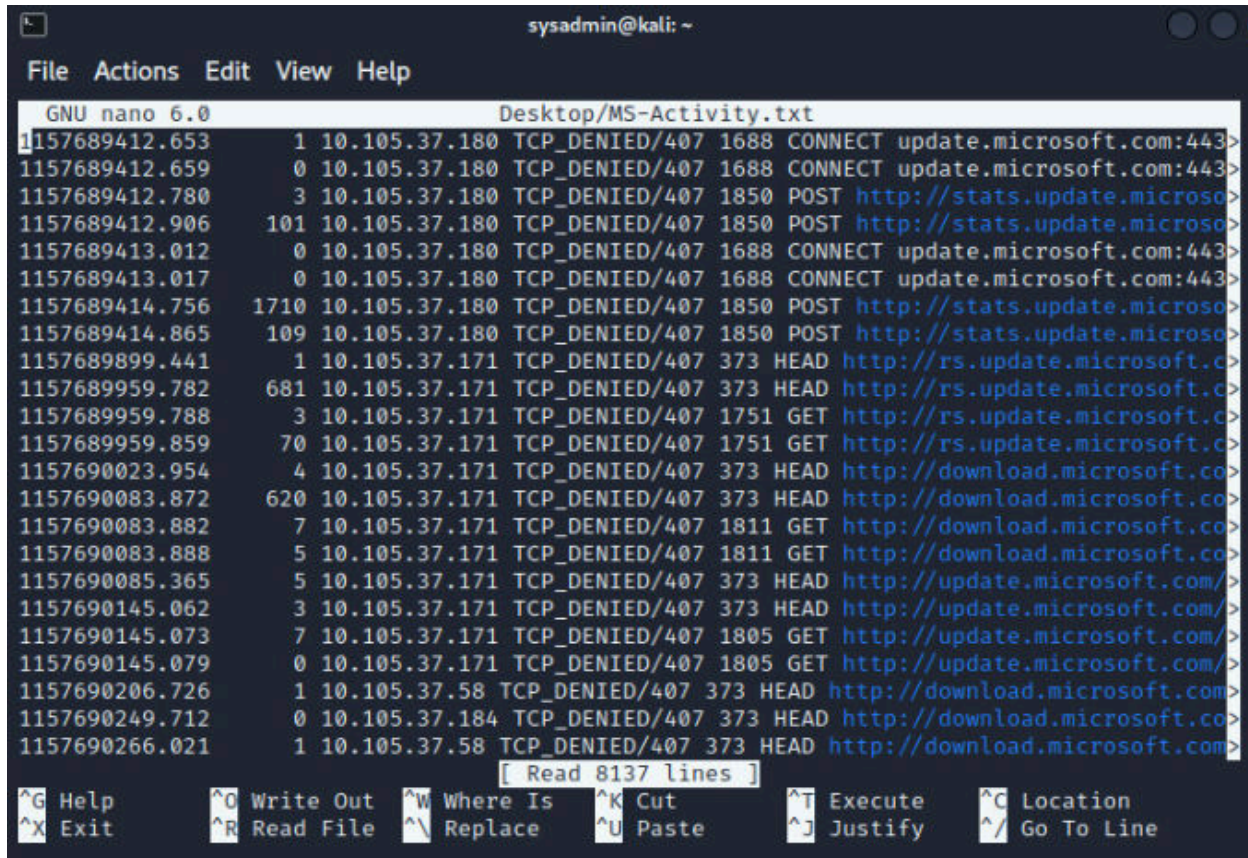
7. On the desktop, you should see the file *MS-Activity.txt*. Open this file in the *nano* editor by typing the command:

```
nano Desktop/MS-Activity.txt
```



```
(sysadmin@kali)-[~]  
$ nano Desktop/MS-Activity.txt
```


You can see that there are 8,137 references.



```

sysadmin@kali: ~
File Actions Edit View Help
GNU nano 6.0 Desktop/MS-Activity.txt
1157689412.653 1 10.105.37.180 TCP_DENIED/407 1688 CONNECT update.microsoft.com:443>
1157689412.659 0 10.105.37.180 TCP_DENIED/407 1688 CONNECT update.microsoft.com:443>
1157689412.780 3 10.105.37.180 TCP_DENIED/407 1850 POST http://stats.update.microso>
1157689412.906 101 10.105.37.180 TCP_DENIED/407 1850 POST http://stats.update.microso>
1157689413.012 0 10.105.37.180 TCP_DENIED/407 1688 CONNECT update.microsoft.com:443>
1157689413.017 0 10.105.37.180 TCP_DENIED/407 1688 CONNECT update.microsoft.com:443>
1157689414.756 1710 10.105.37.180 TCP_DENIED/407 1850 POST http://stats.update.microso>
1157689414.865 109 10.105.37.180 TCP_DENIED/407 1850 POST http://stats.update.microso>
1157689899.441 1 10.105.37.171 TCP_DENIED/407 373 HEAD http://rs.update.microsoft.c>
1157689959.782 681 10.105.37.171 TCP_DENIED/407 373 HEAD http://rs.update.microsoft.c>
1157689959.788 3 10.105.37.171 TCP_DENIED/407 1751 GET http://rs.update.microsoft.c>
1157689959.859 70 10.105.37.171 TCP_DENIED/407 1751 GET http://rs.update.microsoft.c>
1157690023.954 4 10.105.37.171 TCP_DENIED/407 373 HEAD http://download.microsoft.co>
1157690083.872 620 10.105.37.171 TCP_DENIED/407 373 HEAD http://download.microsoft.co>
1157690083.882 7 10.105.37.171 TCP_DENIED/407 1811 GET http://download.microsoft.co>
1157690083.888 5 10.105.37.171 TCP_DENIED/407 1811 GET http://download.microsoft.co>
1157690085.365 5 10.105.37.171 TCP_DENIED/407 373 HEAD http://update.microsoft.com/>
1157690145.062 3 10.105.37.171 TCP_DENIED/407 373 HEAD http://update.microsoft.com/>
1157690145.073 7 10.105.37.171 TCP_DENIED/407 1805 GET http://update.microsoft.com/>
1157690145.079 0 10.105.37.171 TCP_DENIED/407 1805 GET http://update.microsoft.com/>
1157690206.726 1 10.105.37.58 TCP_DENIED/407 373 HEAD http://download.microsoft.com>
1157690249.712 0 10.105.37.184 TCP_DENIED/407 373 HEAD http://download.microsoft.co>
1157690266.021 1 10.105.37.58 TCP_DENIED/407 373 HEAD http://download.microsoft.com>
[ Read 8137 lines ]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line

```

This text file could be parsed and processed into useful information, for example, programming the firewall to block access to and from websites.

8. This concludes the lab. You may now end the reservation.