



HackerFrogs Afterschool Beginner Cybersecurity Skills

Linux Operation Basics with Bandit CTF: Part 2

| | |
|-----------------------|--|
| Class: | Linux OS Operations |
| Workshop Number: | AS-LIN-02 |
| Document Version: | 1.0 |
| Special Requirements: | Completion of previous workshop, AS-LIN-01 |

Table of Contents

| | |
|--|----|
| <u>Introduction</u> | 3 |
| <u>Workshop Guide Format Notes</u> | 3 |
| <u>Using the SSH Program to Connect to the CTF Game Server</u> | 3 |
| <u>Part 1 – Further File Search With Bandit 6</u> | 4 |
| <u>Part 2 – Searching Within File Contents With Bandit 7</u> | 6 |
| <u>Part 3 – Sorting And Ordering File Output With Bandit 8</u> | 7 |
| <u>Part 4 – Further File Contents Search With Bandit 9</u> | 10 |
| <u>Part 5 – The Base 64 Character Codec With Bandit 10</u> | 11 |
| <u>Summary</u> | 13 |
| <u>Extra Credit</u> | 13 |

Introduction

Welcome to HackerFrogs Afterschool! HackerFrogs Afterschool is a series of online workshops meant to teach cybersecurity skills and concepts to beginners who may not be familiar with the field of cybersecurity. This lesson is the second lesson on Linux OS (operating system) operations, which is an essential skill set, due to the heavy preference of Linux OS in the cybersecurity field, and among ethical hackers in particular. This lesson will cover the following learning objectives:

Learning Objectives:

- Interact with the Linux OS filesystem using basic commands
- Open a Command Line Interface (CLI) terminal on your computer
- Use the SSH program to start an interactive session between your computer and a remote server

Workshop Guide Format Notes

When following instructions in this document, there will be instructions to type specific commands in the CLI window or in a search bar. These commands will be displayed in **red**, and will have a different font applied to them, as well as a light grey background color. For example:

```
this is a sample command text line
```

If there are code snippets in this document, they look similar to the format above, except that the text will be black. For example:

```
this is a sample code snippet line
```

Using the SSH Program to Connect to the CTF Game Server

In the **Command Prompt** window, enter the following command to connect to the the Bandit CTF server using the Secure Shell (**SSH**) program (if instructions are required for opening the Command Prompt window, please see the previous workshop's guide document):

```
ssh bandit6@bandit.labs.overthewire.org -p 2220
```

When prompted if you want to continue connecting, enter **yes**, and when asked for a password, enter the password obtained from the previous workshop. When entering a password into the SSH program, you will not see any response or output while you are typing the password. This is normal. If you are pasting the password into the Command Prompt window, you will need to use the **Ctrl + Shift + V** keyboard shortcut.

IMPORTANT PASSWORD NOTE: The Bandit CTF level passwords are changed every few months, so it's possible that the passwords in this document are out of date. Please use the passwords we obtained in the last workshop in that case.

Upon a successful login to the Bandit server, you should see a screen similar to the one below:

```
--[ More information ]--  
  
For more information regarding individual wargames, visit  
http://www.overthewire.org/wargames/  
  
For support, questions or comments, contact us through IRC on  
irc.overthewire.org #wargames.  
  
Enjoy your stay!  
  
bandit6@bandit:~$
```

CONTEXT

The SSH program allows users to connect to other SSH servers on the network, i.e., computers that also have the SSH program installed and running in server mode. Once we have connected to the server, any commands that we run in the terminal are done in the context of the user account we've logged in as (in this case bandit6), and we are now interacting with the file system of the SSH server (at overthewire.org). Also, because the SSH server is running Linux, we now have to use Linux commands to interact with it (regardless of whether our own computer is running Windows or MacOS or something else).

Part 1 – Further File Search With Bandit 6

Step 1

Look at the level objectives for Bandit level 6 at the following URL:

<https://overthewire.org/wargames/bandit/bandit7.html>

Bandit Level 6 → Level 7

Level Goal

The password for the next level is stored somewhere on the server and has all of the following properties:

- owned by user bandit7
- owned by group bandit6
- 33 bytes in size

The info on the webpage indicates that the flag file for the level is in an unknown location on the server, and has all of the following characteristics:

- **owned by the user bandit7**
- **owned by the group bandit6**
- **33 bytes in size**

CONTEXT

We'll be continuing our use of the **find** command for this first exercise. Let's get started.

Step 2

Use the following **find** command to locate files with the criteria specified in the level description:

```
find / -user bandit7 -group bandit6 -size 33c 2>/dev/null
```

The output of the command should look like the screenshot below (target file underlined in red):

```
bandit6@bandit:~$ find / -user bandit7 -group bandit6 -size 33c 2>/dev/null
/var/lib/dpkg/info/bandit7.password
bandit6@bandit:~$
```

Let's go over the different parts of the command we used:

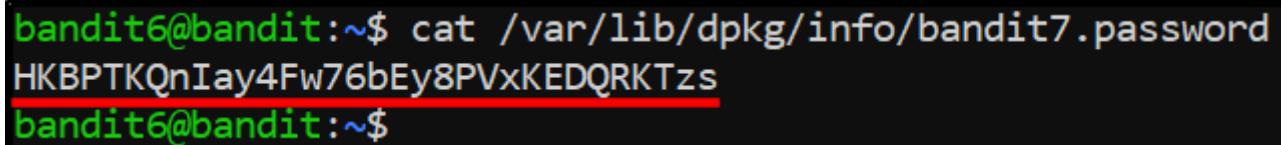
| | |
|-----------------------------|--|
| <code>find</code> | - The command itself |
| <code>/</code> | - The directory to search in (the whole filesystem) |
| <code>-user bandit7</code> | - The file's owner |
| <code>-group bandit6</code> | - The file's group |
| <code>-size 33c</code> | - The file's size |
| <code>2>/dev/null</code> | - Modifies output so that any error messages are not displayed |

Step 3

Now to read the flag file:

```
cat /var/lib/dpkg/info/bandit7.password
```

The output of the command should look like the screenshot below (flag string underlined in red):



```
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs
bandit6@bandit:~$
```

Part 2 – Searching Within File Contents With Bandit 7

Step 1

Login to level 7 of the Bandit CTF game with the following command:

```
ssh bandit7@bandit.labs.overthewire.org -p2220
```

Step 2

Look at the Bandit level 7 objectives at the following webpage:

<https://overthewire.org/wargames/bandit/bandit8.html>

Bandit Level 7 → Level 8

Level Goal

The password for the next level is stored in the file **data.txt** next to the word **millionth**

The focus for this level will be a little different because we'll be looking inside of file contents instead of looking for the files themselves.

Step 3

Use the **grep** command to search inside of the **data.txt** file:

```
grep -e millionth data.txt
```

The output should look like the output in the screenshot below (flag string underlined in red):

```
bandit7@bandit:~$ grep -e millionth data.txt
millionth      cvX2JJJa4CFALtqS87jk27qwqGhBM9p1V
bandit7@bandit:~$
```

CONTEXT

The Linux **grep** command is used to search the contents of files for user-defined words or patterns. Let's break down the grep command we used here:

| | |
|--------------|--------------------------------------|
| grep | - The command itself |
| -e millionth | - Search for the pattern 'millionth' |
| data.txt | - The file to be searched |

Part 3 – Sorting And Ordering File Output With Bandit 8

Step 1

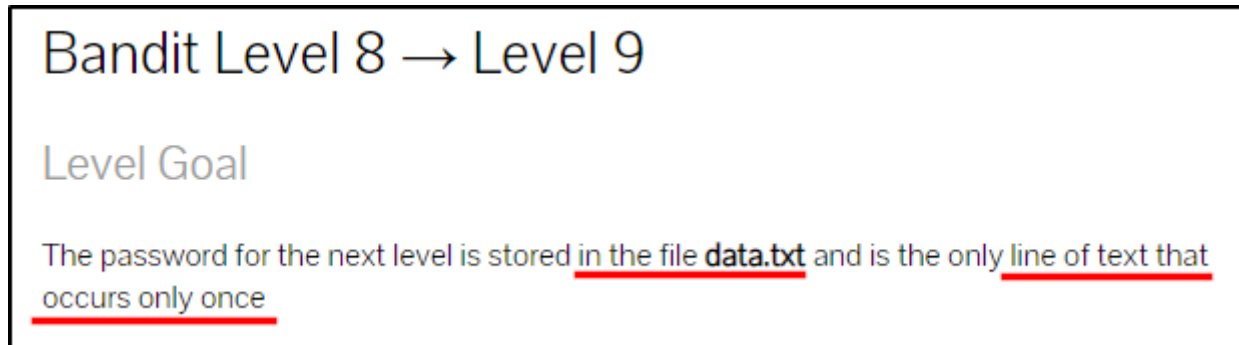
As usual, we start by logging into the Bandit server using SSH:

```
ssh bandit8@bandit.labs.overthewire.org -p 2220
```

Step 2

Then we look at the objectives for the level on associated webpage:

<https://overthewire.org/wargames/bandit/bandit8.html>



In this level, we will be using output redirection and sorting to find the flag.

Step 3

Let's try reading the **data.txt** file:

```
cat data.txt
```

The output for this is a great number of alphanumeric strings. The level instructions state that the flag is the only line that appears only once in the file. First we have to sort the data.

Step 4

Let's use the sort command to put the lines of text in **data.txt** in order:

```
cat data.txt | sort
```

The output of the previous command is show that all of the same lines of text have been grouped together.

CONTEXT

The command we passed in this step is actually two commands, **cat** and **sort**, with the pipe **|** character used to pass the output of one command to the input of another.

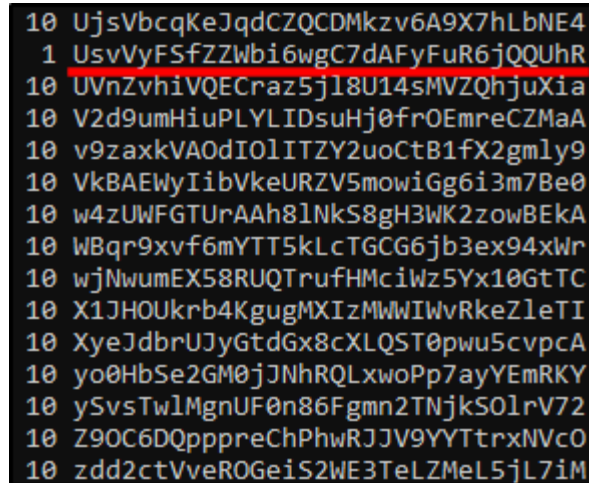
The **sort** command looks through different lines of input and groups any of those lines with the same values together. Next, we'll find the line that occurs only once.

Step 5

Enter the following command to the terminal:

```
cat data.txt | sort | uniq -c
```

The output of this command should look like the following screenshot (flag string underlined in red):



```
10 UjsVbcqKeJqdCZQCDMkv6A9X7hLbNE4
 1 UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhr
10 UVnZvhiVQECraz5j18U14sMVZQhjuXia
10 V2d9umHiuPLYLIDsuHj0fr0EmreCZMaA
10 v9zaxkVAOdIOlITZY2uoCtB1fX2gmly9
10 VkBAEWyIibVkeURZV5mowiGg6i3m7Be0
10 w4zUWFGTUrAAh8lNkS8gH3WK2zowBEkA
10 WBqr9xvf6mYTT5kLcTGCG6jb3ex94xWr
10 wjNwumEX58RUQTrufHMcIWz5Yx10GtTC
10 X1JHOUkrb4KgugMXIzMwWIWvRkeZleTI
10 XyeJdbrUJyGtdGx8cXLQST0pwu5cvpcA
10 yo0HbSe2GM0jJNhRQLxwoPp7ayYEmRKY
10 ySvsTwlMgnUF0n86Fgm2TNjkS0lrV72
10 Z9OC6DQpppreChPhwRJjV9YYTtrxNVcO
10 zdd2ctVveROGeiS2WE3TeLZMeL5jL7iM
```

We've identified the level's flag here, but we can clean up the output just a little bit more in the next step.

CONTEXT

As we can see by the output of the previous command, command piping can be used multiple times. In this case, the output of the **cat** command is passed to the **sort** command, and the **sort** command's output is passed to the **uniq** command.

By default, the **uniq** command returns only one instance of each of the lines in any given input (no matter how many times it appears in the input), with the **-c** flag applied, however, the number of times each line of input appears is also returned.

Step 6

For the sake of completeness, let's add an extra **sort** command to our chain:

```
cat data.txt | sort | uniq -c | sort
```

The expected output should look like the screenshot below (flag string underlined in red):

```
10 wjNwumEX58RUQTrufHMcIWz5Yx10GtTC
10 X1JHOUkrb4KgugMXIzMWwIWvRkeZleTI
10 XyeJdbrUJyGtdGx8cXLQST0pwu5cvpcA
10 yo0HbSe2GM0jJNhRQLxwoPp7ayYEmRKY
10 ySvsTwlMgnUF0n86Fgm2TNjkS0lrV72
10 Z9OC6DQpppreChPhwRJJV9YYTtrxNVc0
10 zdd2ctVveROGeiS2WE3TeLZMeL5jL7iM
1  UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR
bandit8@bandit:~$
```

Part 4 – Further File Contents Search With Bandit 9

Step 1

Login to the Bandit CTF server as the bandit9 user:

```
ssh bandit9@bandit.labs.overthewire.org -p2220
```

Step 2

Check the objectives of Bandit level 4 at the following webpage:

<https://overthewire.org/wargames/bandit/bandit10.html>

Bandit Level 9 → Level 10

Level Goal

The password for the next level is stored in the file data.txt in one of the few human-readable strings, preceded by several '=' characters.

This level deals with the concept of a file's binary **data** content vs **text** content.

Step 3

First let's try reading the contents of **data.txt**:

```
cat data.txt
```

The output that returns is mostly unintelligible. We can extract the human-readable portion of the contents with a specific command.

Step 4

Let's use the **strings** command to return only ASCII text from the **data.txt** file:

```
strings data.txt
```

The output from this command still doesn't give us much, but at least we can read these characters.

CONTEXT

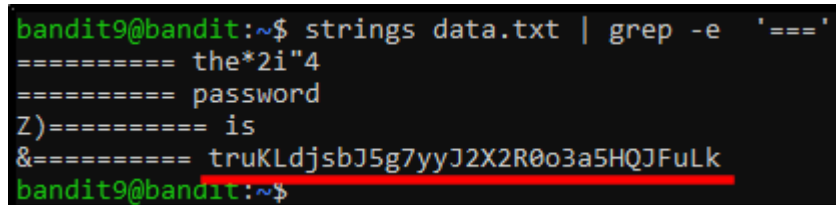
If the contents of a file contain all human-readable characters (in ASCII, UTF-8, etc), then that file's contents can be called **text**. The opposite of file text content is binary **data** content, which is meant to be read by machines (hardware or software). However, files with data content often also contain text, so we can use the **strings** command to extract the text of the file.

Step 5

We can pipe the output of the **strings** command into the **grep** command to obtain the flag for the level:

```
strings data.txt | grep -e '==='
```

The output of the command should look like the screenshot below (level flag string underlined in red):



```
bandit9@bandit:~$ strings data.txt | grep -e '==='  
===== the*2i"4  
===== password  
Z)===== is  
&===== truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk  
bandit9@bandit:~$
```

By passing the output of the **strings** command to the **grep** command, we were able to pull out only those lines of the output that contained the equals '=' characters, which included this level's flag.

Part 5 – The Base 64 Character Codec With Bandit 10

Step 1

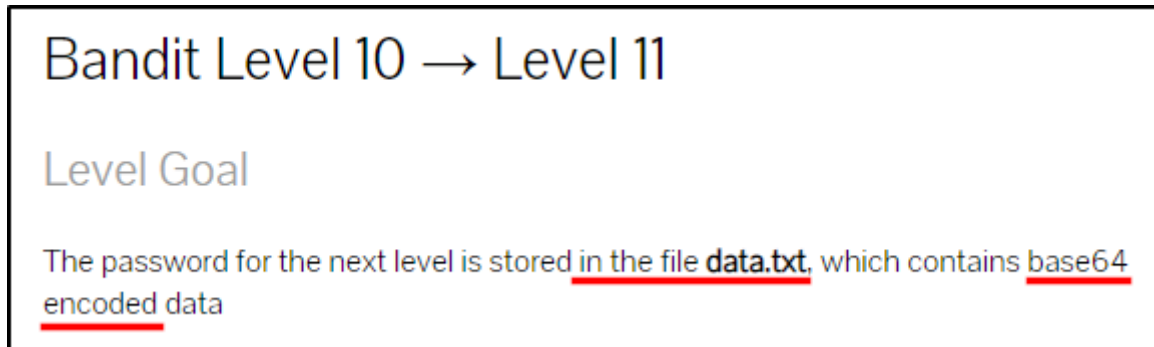
Login to Bandit level 10 with SSH:

```
ssh bandit10@bandit.labs.overthewire.org -p2220
```

Step 2

Check the level objectives at the following webpage:

<https://overthewire.org/wargames/bandit/bandit11.html>



This level of Bandit deals with character encoding, specifically base 64. Let's get started.

Step 3

First, let's try reading the **data.txt** file normally:

```
cat data.txt
```

The results of the command should look like the screenshot below:

```
bandit10@bandit:~$ cat data.txt
VGhlIHBhc3N3b3JkIGlzIElGdWt3S0dzRlc4TU9xM0lSRnFyeEUxaHhUTkViVBSCg==
bandit10@bandit:~$
```

CONTEXT

This text string here has (according to the level description page) been base 64 encoded. Character encoding is used to transform a string of characters into a different form. Character encoding is used for a variety of different reasons, but a common use of it is to transform data into a format for transport over the internet.

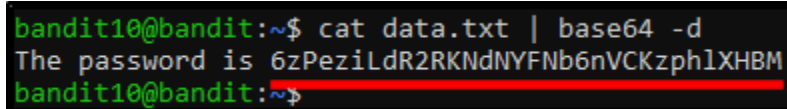
For example, a lot of web servers won't accept non-ASCII characters in HTTP requests, so in order to transfer data which contains UTF-8 characters (e.g., letters with umlauts or tildas), it has to be encoded into another format before transport.

Step 4

Let's run the Linux **base64** command to decode the string in the **data.txt** file:

```
cat data.txt | base64 -d
```

The output of this command should look like the following screenshot (password underlined in red):



```
bandit10@bandit:~$ cat data.txt | base64 -d
The password is 6zPezILdR2RKNdNYFNb6nVCKzphlXHBM
bandit10@bandit:~$
```

CONTEXT

By default, the **base64** command will encode whatever data is based to it into a base64 string. If we append the **-d** flag to it, the **base64** command will instead decode the data into hexadecimal bytes, which are then ASCII encoded to printable characters (if possible).

Summary

While working through the Bandit CTF levels 6 to 10, we learned to use The following commands to work inside of the Linux OS file system:

| | |
|----------------|--|
| find | <-- this command is used to search for files on a computer system |
| grep | <-- this command is used to search for strings in file contents |
| sort | <-- this command is used to sort lines in a file according to alphabetical / numerical order |
| uniq | <-- this command is used to filter out lines with identical content in files |
| base64 | <-- this command is used to encode / decode data using the base64 codec |
| strings | <-- this command is used to find human-readable text in files |

Extra Credit

The following is an extra exercise that is relevant to the material taught in this workshop:

TryHackMe – Linux Fundamentals Part 1

<https://tryhackme.com/room/linuxfundamentalspart1>

NOTE: We will need a registered account to complete this exercise.

In the next workshop, we'll be challenging our understanding of Linux commands with the TryHackMe platform.

Until next time, HackerFrogs!

