DFRWS USA 2023

DFRWS
DIGITAL FORENSIC RESEARCH CONFERENCE

Learning Linux Forensic Analysis and Why it Matters

By Ali Hadi & Mariam Khader

Thanks to our sponsers:
- Champlain College
- Cyber 5W
- Leahy Center

Our Sponsors

Compromised HDFS Cluster!

Case Brief

You have been called to analyze a compromised Linux Hadoop Cluster. The cluster includes one Name Node (master) and two Data Nodes (Slaves). There is a suspicion that they all have been compromised, but no proof to that. The activity has been noticed to happen between Oct. 5th, 2019 and Oct. 8th, 2019.

Deliverable(s)

1. How did the threat actor gain access to the system?
2. What privileges were obtained and how?
3. What modifications were applied to the system?
4. What persistent mechanisms on each compromised system were being used?
5. Could this system be cleaned/recovered?
6. Recommendations

Outcome(s)

At the end of this lab, you will have the required skills to deal with a compromised Linux system, were you will be capable of doing:

1. Listing the volumes and mounting a forensic case image
2. Searching through the FHS
3. Search in log files
4. Understanding system services and how they work
5. Use TSK tools to list info of the image and deal with EXT4 fs
6. Use debugfs, EXT4 journal and ext4magic to recover deleted files
7. Generate and filter a super timeline

Table of Contents

Task #0: Environment Preparation

0.0 Connecting to the Playground

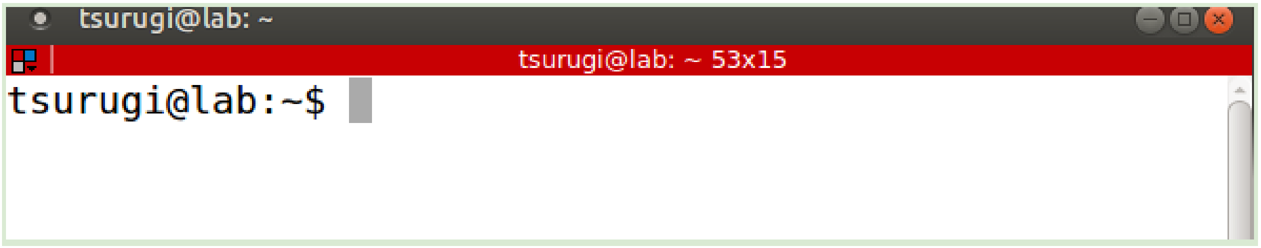
Please use your browser to connect to the following IP address to access your lab Virtual Machine (VM).

Playground Credentials		
Server	https://192.168.1.10____	
Username	user___	
Password	workshop	
Virtual Machine Credentials		
Username	**tsurugi**	
Password	**tsurugi**	

0.1 Preparing Working Environment

From the top panel click on the red icon with a cross inside. Or from the menu as seen in figure 1.

Applications → System Tools → MATE Terminal



```
tsurugi@lab:~$
```

Figure 1 - Mate Terminal

Create a directory named "**cases**"

```
$ mkdir cases
```

Then change your working directory to the newly created directory, as follows:

```
$ cd cases
```

This will be where we will store all our cases, but for now let us create another directory for our **hdfs** case as following:

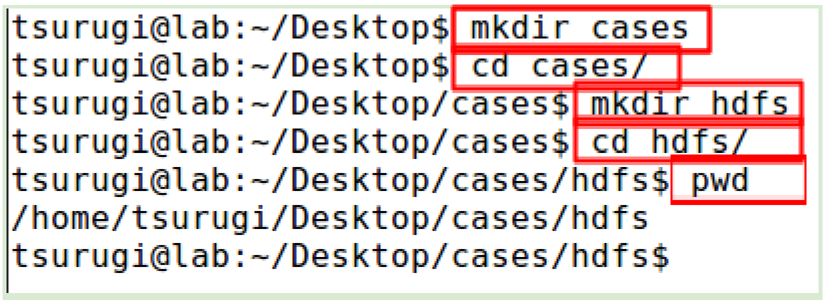
$ mkdir hdfs

Again, change your working directory to the newly created directory, as follows:

$ cd hdfs

Make sure you're inside the **hdfs** directory. This could be done with the '**pwd**' command as follows and as seen in figure 2.

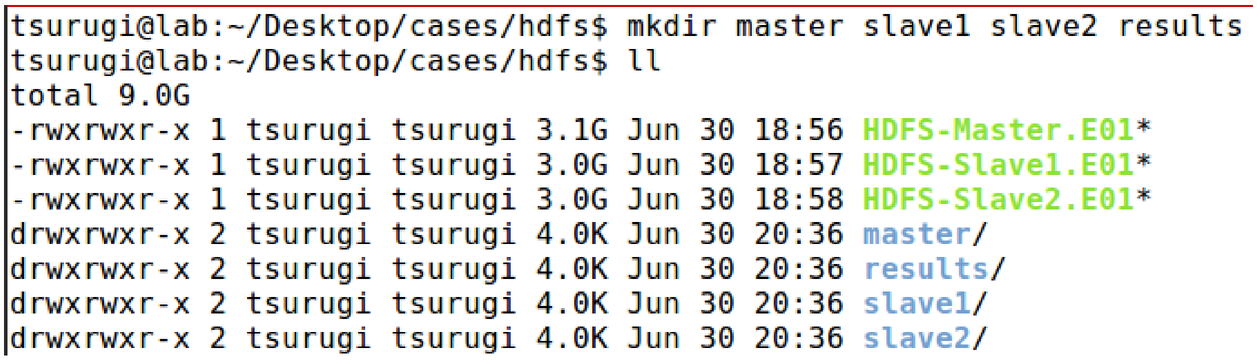
$ pwd



```
tsurugi@lab:~/Desktop$ mkdir cases
tsurugi@lab:~/Desktop$ cd cases/
tsurugi@lab:~/Desktop/cases$ mkdir hdfs
tsurugi@lab:~/Desktop/cases$ cd hdfs/
tsurugi@lab:~/Desktop/cases/hdfs$ pwd
/home/tsurugi/Desktop/cases/hdfs
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 2 - Preparing Evidence Environment

Now let's create some mount points to be used later to mount our forensic images. Don't worry, more on that later and also a directory to hold our results that we pull from these forensic images. This can be seen in figure 3.



```
tsurugi@lab:~/Desktop/cases/hdfs$ mkdir master slave1 slave2 results
tsurugi@lab:~/Desktop/cases/hdfs$ ll
total 9.0G
-rwxrwxr-x 1 tsurugi tsurugi 3.1G Jun 30 18:56 HDFS-Master.E01*
-rwxrwxr-x 1 tsurugi tsurugi 3.0G Jun 30 18:57 HDFS-Slave1.E01*
-rwxrwxr-x 1 tsurugi tsurugi 3.0G Jun 30 18:58 HDFS-Slave2.E01*
drwxrwxr-x 2 tsurugi tsurugi 4.0K Jun 30 20:36 master/
drwxrwxr-x 2 tsurugi tsurugi 4.0K Jun 30 20:36 results/
drwxrwxr-x 2 tsurugi tsurugi 4.0K Jun 30 20:36 slave1/
drwxrwxr-x 2 tsurugi tsurugi 4.0K Jun 30 20:36 slave2/
```

Figure 3 - Creating Mount Points

Hint: If you ever want to check why a command was used this way or what the options being used mean? Then use one of the following:

1. **man command-name**
2. **command --help**
3. **command -h**
    a. **Example: $ man mmls**

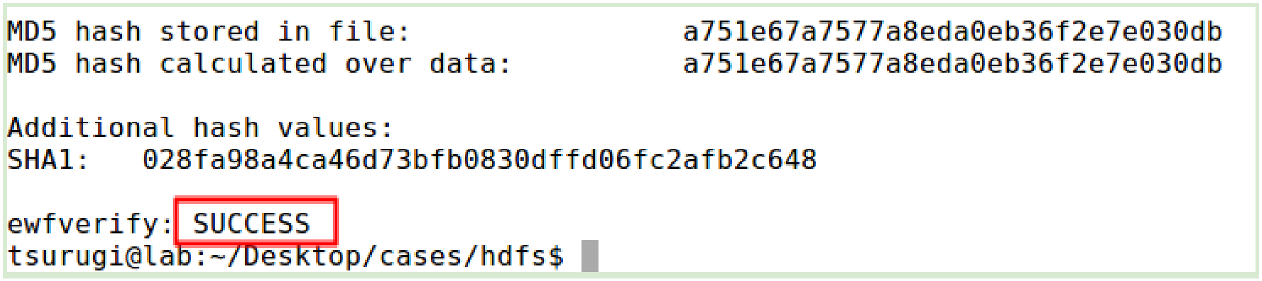
Task #1: Verification and Mounting

1.1 Verifying the Evidence

Before doing anything, let's verify the case image, which could be done as seen below:

```
$ ewfverify HDFS-Master.E01
```

Make sure you get a success message for all three forensic images (this may take some time depending on the specs of your VM, so please be patient). An example can be seen in figure 1.1.

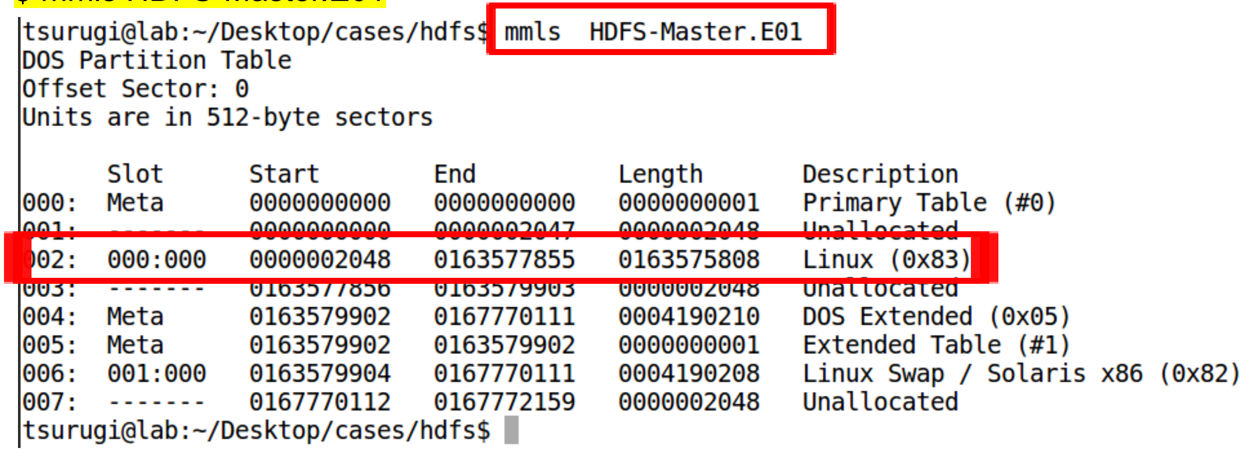


```
MD5 hash stored in file:              a751e67a7577a8eda0eb36f2e7e030db
MD5 hash calculated over data:        a751e67a7577a8eda0eb36f2e7e030db

Additional hash values:
SHA1:   028fa98a4ca46d73bfb0830dffd06fc2afb2c648

ewfverify: SUCCESS
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.1 - Verifying the Master Server Forensic Image

Now, let us check our drives and what volumes does each one of them include. Let's start with the Master and the results can be seen in figure 1.2.

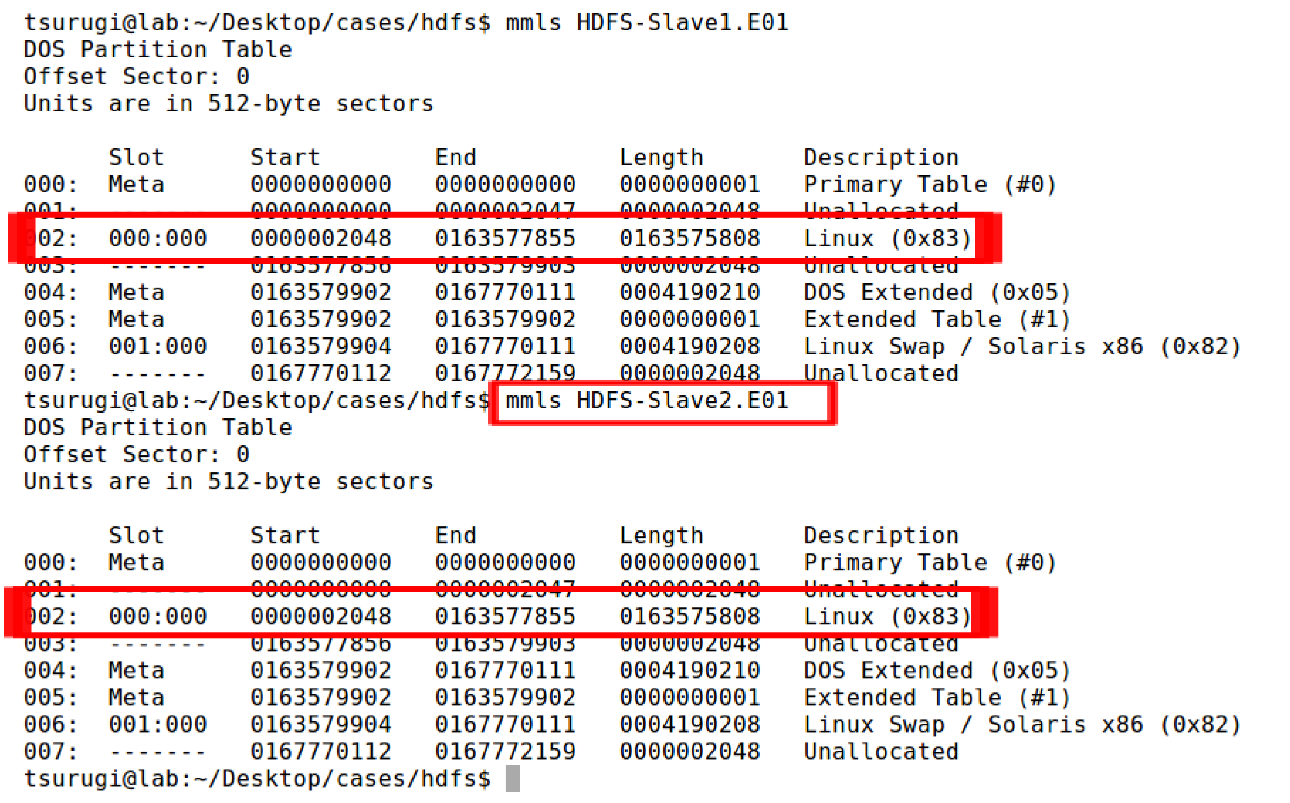
```
$ mmls HDFS-Master.E01
```



```
tsurugi@lab:~/Desktop/cases/hdfs$ mmls  HDFS-Master.E01
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0163577855   0163575808   Linux (0x83)
003:  -------   0163577856   0163579903   0000002048   Unallocated
004:  Meta      0163579902   0167770111   0004190210   DOS Extended (0x05)
005:  Meta      0163579902   0163579902   0000000001   Extended Table (#1)
006:  001:000   0163579904   0167770111   0004190208   Linux Swap / Solaris x86 (0x82)
007:  -------   0167770112   0167772159   0000002048   Unallocated
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.2 - Partition and Volume Layout for Master Server

We can see the results for Slave 1 and Slave 2 in figure 1.3.



```
tsurugi@lab:~/Desktop/cases/hdfs$ mmls HDFS-Slave1.E01
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0163577855   0163575808   Linux (0x83)
003:  -------   0163577856   0163579903   0000002048   Unallocated
004:  Meta      0163579902   0167770111   0004190210   DOS Extended (0x05)
005:  Meta      0163579902   0163579902   0000000001   Extended Table (#1)
006:  001:000   0163579904   0167770111   0004190208   Linux Swap / Solaris x86 (0x82)
007:  -------   0167770112   0167772159   0000002048   Unallocated
tsurugi@lab:~/Desktop/cases/hdfs$ mmls HDFS-Slave2.E01
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0163577855   0163575808   Linux (0x83)
003:  -------   0163577856   0163579903   0000002048   Unallocated
004:  Meta      0163579902   0167770111   0004190210   DOS Extended (0x05)
005:  Meta      0163579902   0163579902   0000000001   Extended Table (#1)
006:  001:000   0163579904   0167770111   0004190208   Linux Swap / Solaris x86 (0x82)
007:  -------   0167770112   0167772159   0000002048   Unallocated
tsurugi@lab:~/Desktop/cases/hdfs$
```

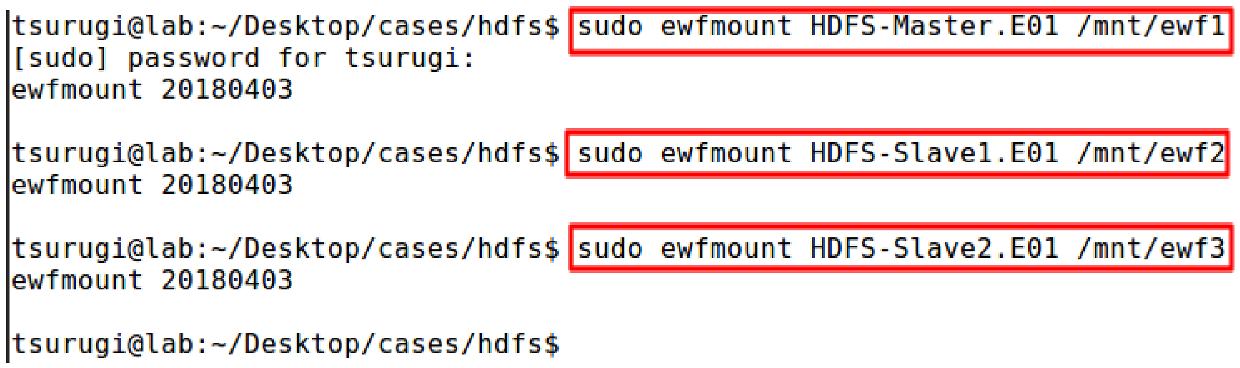
Figure 1.3 - Partition and Volume Layout for both Slave1 and Slave2 Servers

The volumes we are interested in are the ones with index number 002. As you can see, these volumes are all described as Linux (0x83), so most probably they hold a Linux file system. Another thing to note is that they all start at a sector offset of 2048. This will be needed later when we get to the mounting part of the workshop.

1.2 Mounting the Evidence

First, let us mount the E01 file for each one of them, which could be done using the command below and as seen in figure 1.4.

```
$ sudo ewfmount HDFS-Master.E01 /mnt/ewf1/
```

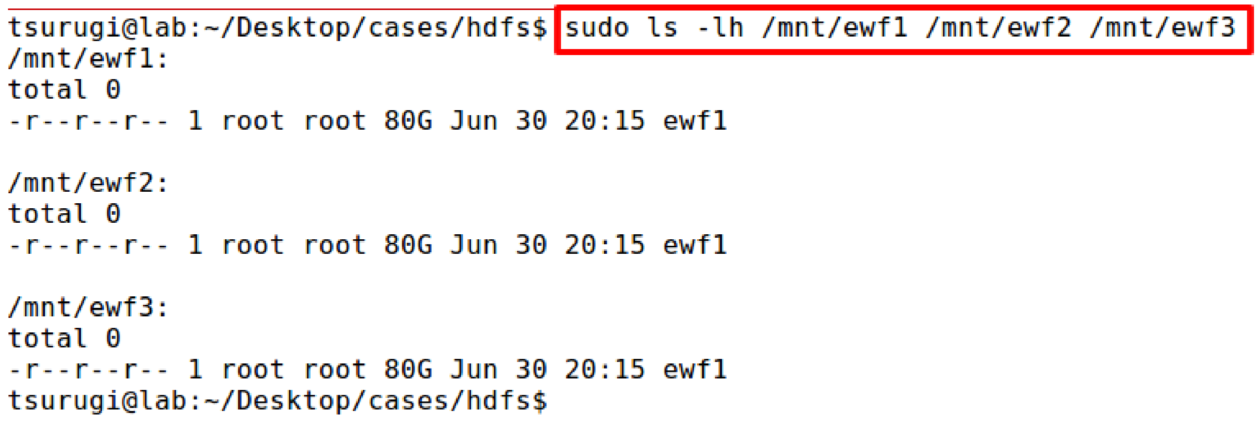


```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo ewfmount HDFS-Master.E01 /mnt/ewf1
[sudo] password for tsurugi:
ewfmount 20180403

tsurugi@lab:~/Desktop/cases/hdfs$ sudo ewfmount HDFS-Slave1.E01 /mnt/ewf2
ewfmount 20180403

tsurugi@lab:~/Desktop/cases/hdfs$ sudo ewfmount HDFS-Slave2.E01 /mnt/ewf3
ewfmount 20180403

tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.4 - Mounting Forensic Images using ewfmount

If you list the contents of the ewf1 directory, you should see a file named "**ewf1**". You'll see a similar file in both ewf2 and ewf3 as seen in figure 1.5.

```
$ sudo ls -lh  /mnt/ewf1   /mnt/ewf2   /mnt/ewf3
```



```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo ls -lh /mnt/ewf1 /mnt/ewf2 /mnt/ewf3
/mnt/ewf1:
total 0
-r--r--r-- 1 root root 80G Jun 30 20:15 ewf1

/mnt/ewf2:
total 0
-r--r--r-- 1 root root 80G Jun 30 20:15 ewf1

/mnt/ewf3:
total 0
-r--r--r-- 1 root root 80G Jun 30 20:15 ewf1
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.5 - Checking ewf Directories After Mount

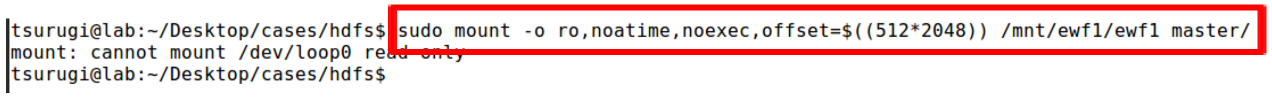
Whenever you need help or more info. on a command, just check the man pages 😉

Great, so we have everything prepared, now let us get into business! Make sure you are within the **hdfs** directory "**/home/tsurugi/cases/hdfs**", you can double check your current location using **pwd** 😉

Before we mount the volumes we will need the offset to the volume of interest, which we saw in both figure 1.2 and figure 1.3 to be 2048. Now this value is in sectors and at the beginning of both of those figures, we can see a line saying "Units are in 512-byte sectors". So we need to multiply the offset by 512.

Mounting the forensic images can be done using the mount command as seen below and the result seen in figure 1.6.

sudo mount -o ro,noexec,noatime,offset=$((512*2048)) /mnt/ewf1/ewf1 master



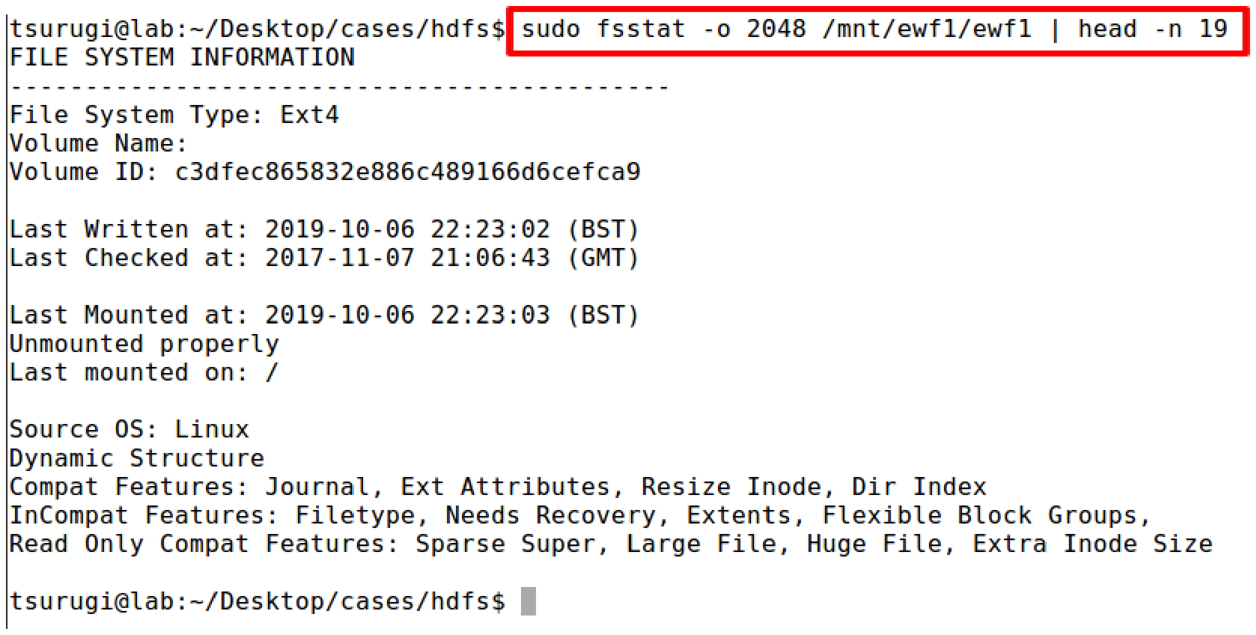
```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo mount -o ro,noatime,noexec,offset=$((512*2048)) /mnt/ewf1/ewf1 master/
mount: cannot mount /dev/loop0 read-only
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.6 - Trying to Mount Linux Volume on Master Server

Q1.1: Was the command successful or not and why?

Let us do some checking first using **fsstat** to see why that happened. We will need the offset to the volume of interest, but **fsstat** can deal directly with sectors, so we just need to pass the offset as seen below and seen in figure 1.7.

$ sudo fsstat -o 2048 /mnt/ewf1/ewf1 | head -n 19



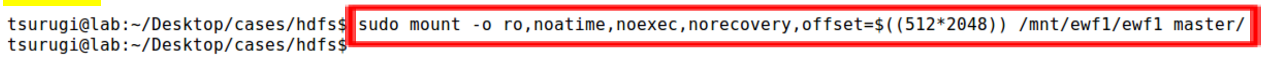
```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo fsstat -o 2048 /mnt/ewf1/ewf1 | head -n 19
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext4
Volume Name:
Volume ID: c3dfec865832e886c489166d6cefca9

Last Written at: 2019-10-06 22:23:02 (BST)
Last Checked at: 2017-11-07 21:06:43 (GMT)

Last Mounted at: 2019-10-06 22:23:03 (BST)
Unmounted properly
Last mounted on: /

Source OS: Linux
Dynamic Structure
Compat Features: Journal, Ext Attributes, Resize Inode, Dir Index
InCompat Features: Filetype, Needs Recovery, Extents, Flexible Block Groups,
Read Only Compat Features: Sparse Super, Large File, Huge File, Extra Inode Size

tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.7 - Checking File System Info

Please read all the details, they are important, but for being as brief as possible here, check the line under "***Last Mounted at***". It says that it was not unmounted properly, and this might happen when the system was not shutdown properly. Therefore, this could mean that there is some data in the journal that was not written to volume, which will usually happen once the volume comes back online.

Okay, enough talking, let's adjust our command with the noload/norecovery option which can be seen in figure 1.8.

sudo mount -o ro,noexec,noatime,norecovery,offset=$((512*2048)) /mnt/ewf1/ewf1 master



```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo mount -o ro,noatime,noexec,norecovery,offset=$((512*2048)) /mnt/ewf1/ewf1 master/
tsurugi@lab:~/Desktop/cases/hdfs$
```

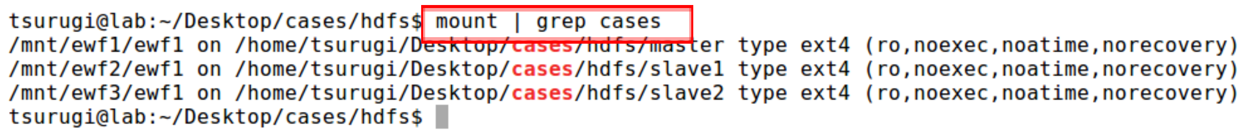
Figure 1.8 - Successfully Mounting Linux Volume on Master Server

Super! No errors this time! Therefore, make sure to repeat the same for all of the other volumes on the Slave1 and Slave2 forensic images.

1.3 Checking Status of Mounted Evidence

In order to check the status of the mounted volumes, we can use the mount command again but with no options. This can be seen in figure 1.9.

$ mount | grep cases

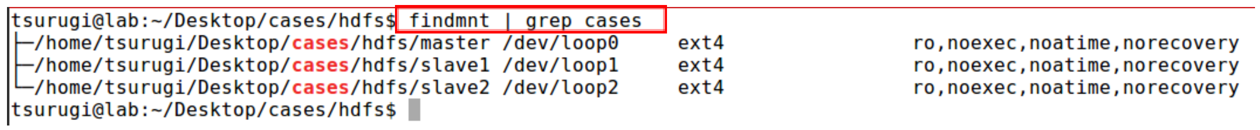


```
tsurugi@lab:~/Desktop/cases/hdfs$ mount | grep cases
/mnt/ewf1/ewf1 on /home/tsurugi/Desktop/cases/hdfs/master type ext4 (ro,noexec,noatime,norecovery)
/mnt/ewf2/ewf1 on /home/tsurugi/Desktop/cases/hdfs/slave1 type ext4 (ro,noexec,noatime,norecovery)
/mnt/ewf3/ewf1 on /home/tsurugi/Desktop/cases/hdfs/slave2 type ext4 (ro,noexec,noatime,norecovery)
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.9 - Checking Status of Mounted Volumes

Let us do one more check to find which loop device are these forensic images connected to. This can be done using the findmnt command as seen in figure 1.10.

$ findmnt | grep cases



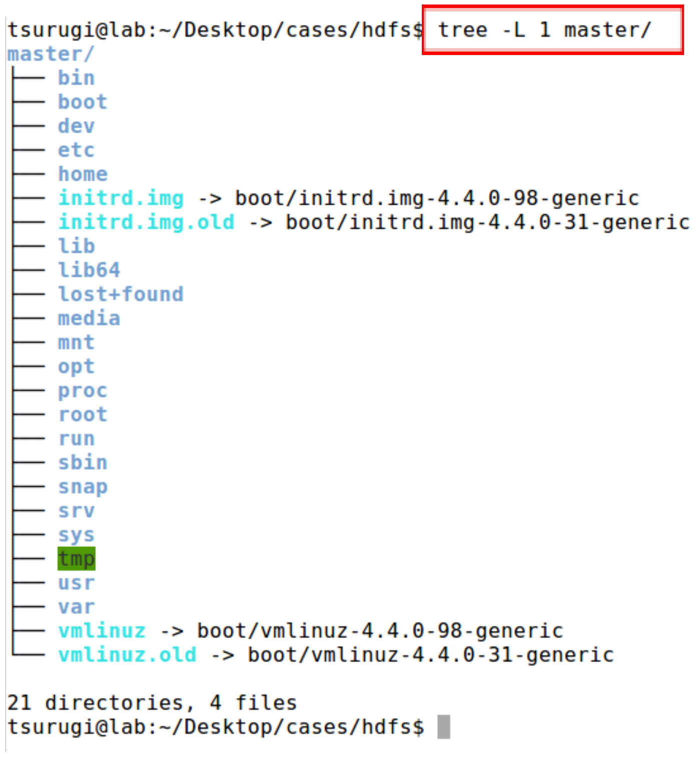
```
tsurugi@lab:~/Desktop/cases/hdfs$ findmnt | grep cases
├─/home/tsurugi/Desktop/cases/hdfs/master /dev/loop0     ext4             ro,noexec,noatime,norecovery
├─/home/tsurugi/Desktop/cases/hdfs/slave1 /dev/loop1     ext4             ro,noexec,noatime,norecovery
└─/home/tsurugi/Desktop/cases/hdfs/slave2 /dev/loop2     ext4             ro,noexec,noatime,norecovery
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.10 - Finding Which Loop Device is Attached to Each Volume

So from the results we can see that the **master** case is mounted on /dev/loop0, **slave1** to /dev/loop1, and **slave2** to /dev/loop2. This will be very useful later when we get to use TSK in task #4.

Now let's see what we have now inside the "**master**" directory. I'm going to use "tree" this time to do that, but feel free to use other stuff, such as "ls". Make sure to do this for the other two directories we have (slave1 and slave2). This can be seen in figure 1.11.

$ tree -L 1 master



```
tsurugi@lab:~/Desktop/cases/hdfs$ tree -L 1 master/
master/
├── bin
├── boot
├── dev
├── etc
├── home
├── initrd.img -> boot/initrd.img-4.4.0-98-generic
├── initrd.img.old -> boot/initrd.img-4.4.0-31-generic
├── lib
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── snap
├── srv
├── sys
├── tmp
├── usr
├── var
├── vmlinuz -> boot/vmlinuz-4.4.0-98-generic
└── vmlinuz.old -> boot/vmlinuz-4.4.0-31-generic

21 directories, 4 files
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 1.11 - Using Tree to List the Directory Content

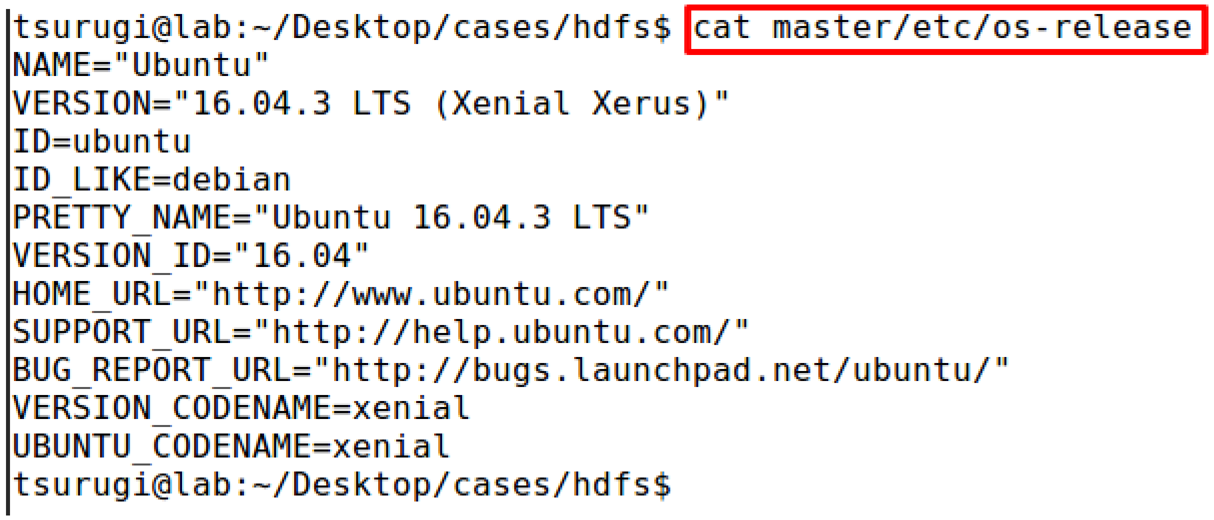
Task #2: Gathering General System Information

This task will be divided into multiple sections, to make sure we go over as much information as possible to help us with our investigation.

2.1 System Navigation

PLEASE (all uppercase) take some time to navigate and understand the file hierarchy standard (FHS) before proceeding. Understanding the hierarchy of the system is very important, especially if you are new to Linux systems. It is an excellent time to ask yourself questions about the directories under root and what could be found under each one of them.

Let us also verify the Linux flavor we are dealing with. This can be done by checking the /etc/os-release file as seen in figure 2.1.



```
tsurugi@lab:~/Desktop/cases/hdfs$ cat master/etc/os-release
NAME="Ubuntu"
VERSION="16.04.3 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.3 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 2.1 - Checking Linux Flavor

As you can see from the results of our master server, we are dealing with an Ubuntu Linux 16.04.3 LTS code name Xenial.

Please do the same to both slave servers before moving further.

2.2 Timezone Information

An important part of an investigation is also verifying the timezone used on the system. I know some might say that this is already provided to us. You're correct about that BUT "trust, but verify" is how we should go with investigations…

Let's check the timezone found on all of the systems:

```
$ cat master/etc/timezone slave1/etc/timezone slave2/etc/timezone
```

Q2.1: What did you find?

You can double check the results by examining the /etc/localtime file for the timezone. So when checking it using the file command, you should find the path to the zone information being used on the system.

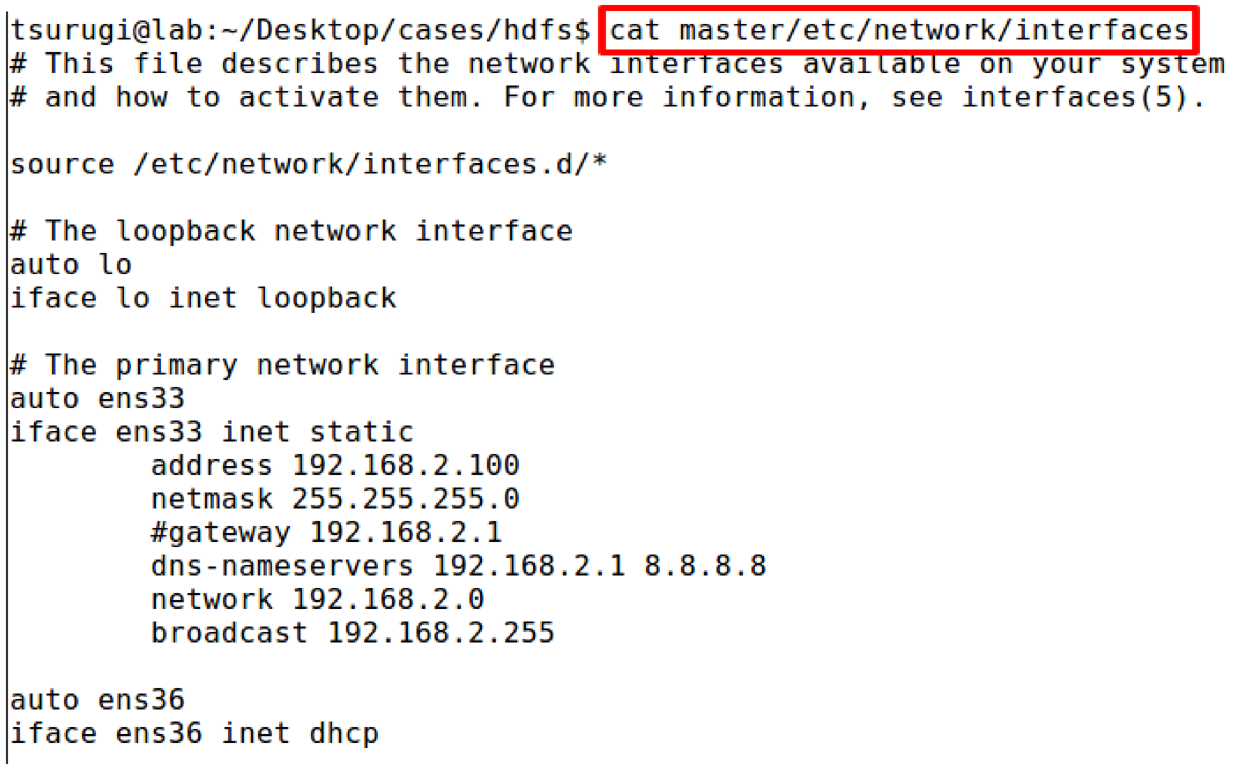
Q2.2: What type of file is /etc/localtime?

We are going to use this information later, especially when generating our timeline, so make sure you document it on your technical notes document.

PLEASE READ ME: not all of the workshop has screenshots for a reason, which is we need you to pay attention and ask questions and not just copy and paste commands. Therefore, we will purposely leave questions and commands unanswered, but they will be together, so enjoy the ride 😉

2.3 Network Information

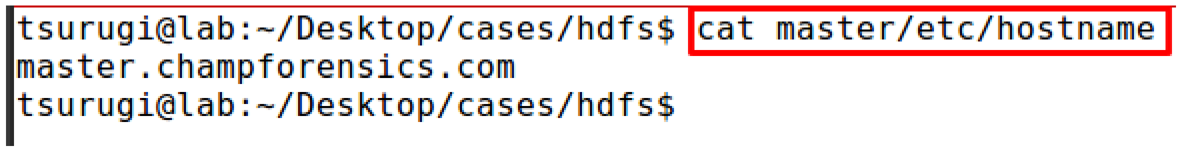
Let us gather information about the network configurations used on these three servers. Now, since we knew that this is an Ubuntu server, we know that the network settings could be found in either the /etc/network or /etc/netplan directory. Others could exist, but these are the most commonly seen locations. The results of this check can be seen in figure 2.2.



```
tsurugi@lab:~/Desktop/cases/hdfs$ cat master/etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens33
iface ens33 inet static
        address 192.168.2.100
        netmask 255.255.255.0
        #gateway 192.168.2.1
        dns-nameservers 192.168.2.1 8.8.8.8
        network 192.168.2.0
        broadcast 192.168.2.255

auto ens36
iface ens36 inet dhcp
```

Figure 2.2 - Checking Network Configuration

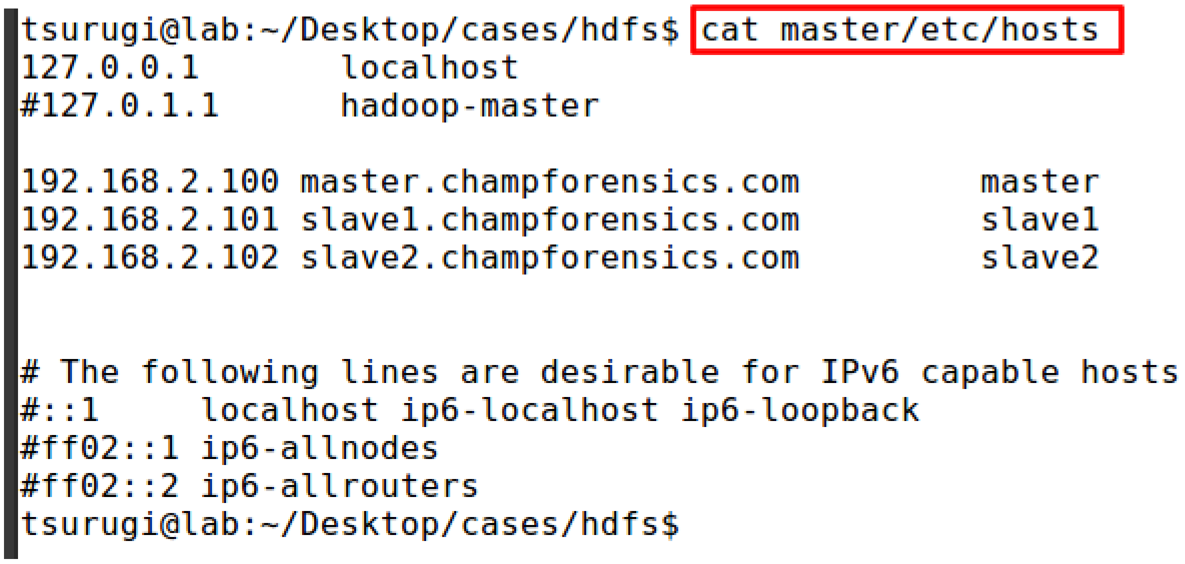
Let's check the hostname of the server which can be found in the /etc/hostname file as seen in figure 2.3.



```
tsurugi@lab:~/Desktop/cases/hdfs$ cat master/etc/hostname
master.champforensics.com
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 2.3 - Checking Hostname of System

Another important location to check would be the /etc/hosts file, which could be used to define static host lookups (static dns settings). Using cat again, we can see some cool results in figure 2.4.



```
tsurugi@lab:~/Desktop/cases/hdfs$ cat master/etc/hosts
127.0.0.1       localhost
#127.0.1.1      hadoop-master

192.168.2.100 master.champforensics.com         master
192.168.2.101 slave1.champforensics.com         slave1
192.168.2.102 slave2.champforensics.com         slave2


# The following lines are desirable for IPv6 capable hosts
#::1     localhost ip6-localhost ip6-loopback
#ff02::1 ip6-allnodes
#ff02::2 ip6-allrouters
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 2.4 - Checking Static DNS Lookups

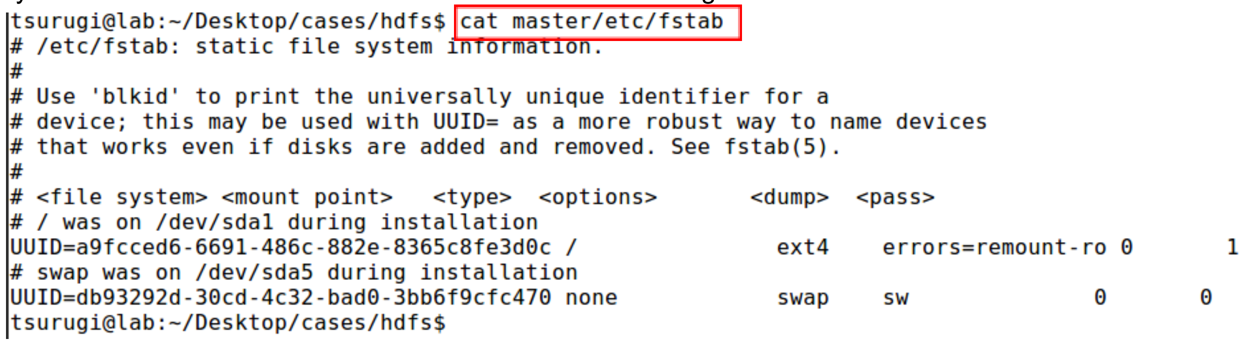
Based on the settings found from the master server configuration files (make sure you check slave1 and slave2 too), we can conclude all the results into Table 2.1.

Table 2.1 - Cluster Network Settings

Hostname	FQDN	IP Address	Subnet	Gateway	DNS
master	master.champforensics.com	192.168.2.100	/24	192.168.2.1	192.168.2.1 and 8.8.8.8
slave1	slave1.champforensics.com	192.168.2.101			
slave2	slave2.champforensics.com	192.168.2.102			

2.4 Drive Information

We can find useful information about the drives, volumes, and file systems available from the /etc/fstab file and also by directly checking the drive as we saw in task 1, when we used mmls. Let's check the file system information found on the master server as seen in figure 2.5.



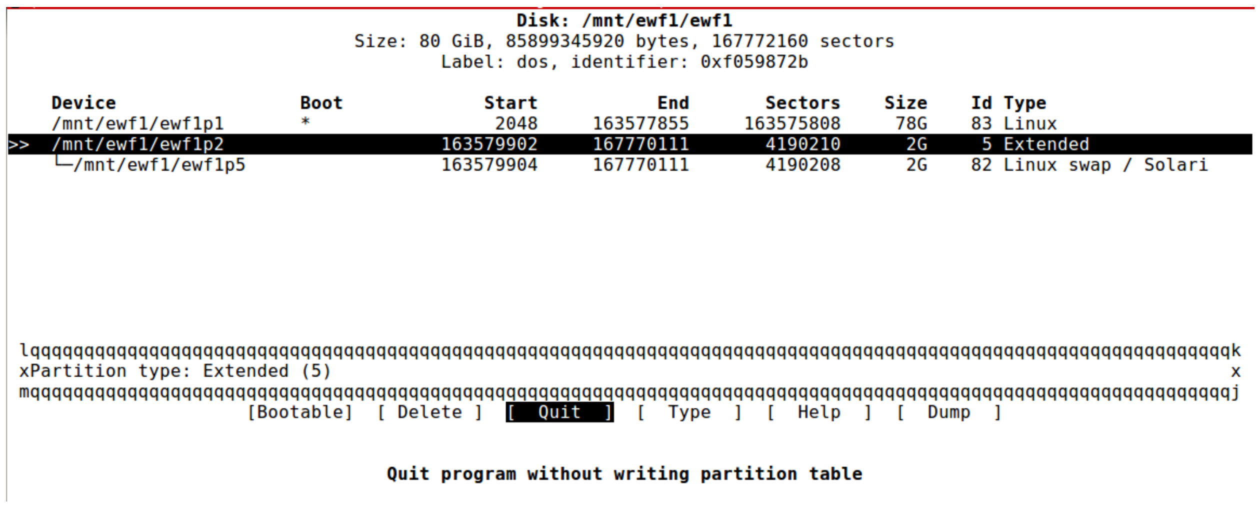
```
tsurugi@lab:~/Desktop/cases/hdfs$ cat master/etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=a9fcced6-6691-486c-882e-8365c8fe3d0c /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=db93292d-30cd-4c32-bad0-3bb6f9cfc470 none            swap    sw              0       0
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 2.5 - Checking File System Info.

It seems that the system is configured to use UUIDs instead of directly using the volume numbers, which is a great way to configure Linux systems (ask your instructor for more information if you want). So we need to find a way to map the UUIDs to the drives and the volumes on them. Now, since we are actually investigating a forensic image, we cannot use the mount command or the /etc/mtab file because it will be empty, plus we cannot use the /proc/mounts pseudo file either. Therefore we will be using the cfdisk tool to find the UUIDs of each partition.

Use the command as seen below and then use the arrows to find the partition with the UUID of the ext4 file system as seen in figure 2.6. This will be the root partition of this system.

```
$ sudo cfdisk /mnt/ewf1/ewf1
```



```
                                   Disk: /mnt/ewf1/ewf1
                     Size: 80 GiB, 85899345920 bytes, 167772160 sectors
                           Label: dos, identifier: 0xf059872b

    Device              Boot          Start        End     Sectors   Size  Id Type
    /mnt/ewf1/ewf1p1    *              2048  163577855   163575808    78G  83 Linux
>>  /mnt/ewf1/ewf1p2             163579902  167770111     4190210     2G   5 Extended
    └─/mnt/ewf1/ewf1p5           163579904  167770111     4190208     2G  82 Linux swap / Solari




 lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
 xPartition type: Extended (5)                                                                    x
 mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
                    [Bootable]  [ Delete ]  [  Quit  ]  [  Type  ]  [  Help  ]  [  Dump  ]


                              Quit program without writing partition table
```

Figure 2.6 - Using cfdisk to Find Volume UUID

Make sure you follow the same process for the other servers.

Note: on live systems, you can check the /dev/disk directory, where you can list the info by **id**, **label**, **path**, and by **UUID**.

Task #3: Users, Groups, and Home Directories

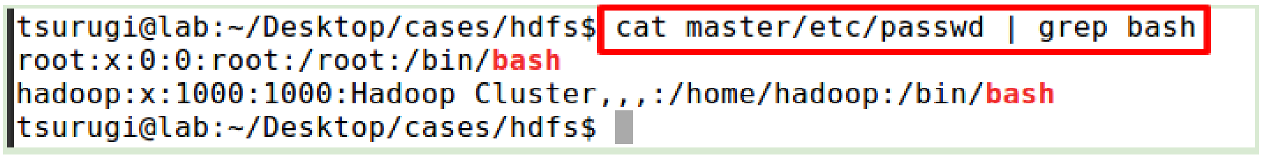
This task will also be divided into multiple sections, to make sure we go over as much information as possible to help us with our learning process and investigation.

3.1 User Information

On a Linux system, the user information is found in the /etc/passwd file, so make sure you check it out before proceeding.

Now, let's check it out using the command below. As you can see in figure 3.1, this will list only the users that have a **bash** shell configured to use at login. Please note that on some systems this might be **sh** or **dash**, etc. So make sure you know the environment you are investigating.

```
$ cat master/etc/passwd | grep bash
```



```
tsurugi@lab:~/Desktop/cases/hdfs$ cat master/etc/passwd | grep bash
root:x:0:0:root:/root:/bin/bash
hadoop:x:1000:1000:Hadoop Cluster,,,:/home/hadoop:/bin/bash
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 3.1 - Accounts with Bash Shells

From the results we can see that there are only two users who have shell access, root and hadoop.

Q3.1) Is this the same for all of the other systems? If the answer is NO, which system has a different output and what did you find?

Q3.2) Explain what you found that is not following the standard Linux FHS?

Let's check if they have passwords to login. This can be found in the shadow file, which is found under the /etc directory. So use the information you found to check the /etc/shadow file for each system. You can do that as seen below. **Note:** make sure to replace username with the username you found in the previous steps.

```
$ cat master/etc/shadow | grep username
```

In the shadow file, the second column (: is the separator), if you find a * it means no password is there, but if you find a long string? Then a password exists for that user.

Q3.3) List who has a password.

3.2 Group Information

We need to check the group info that these users belong to. The group info is found in /etc/group file.

Q3.4) What are the groups that the user hadoop belongs to?

Q3.5) Who has sudo access, or in other words, is in the sudo group?

Use all the information you gathered to list on each system which user was found, their corresponding user ID, home directory, and finally the groups they belong to. Add all this to the table 3.1 found below. An example has already been made for you to follow.

Table 3.1 - Usernames, Groups, Home Directories, etc

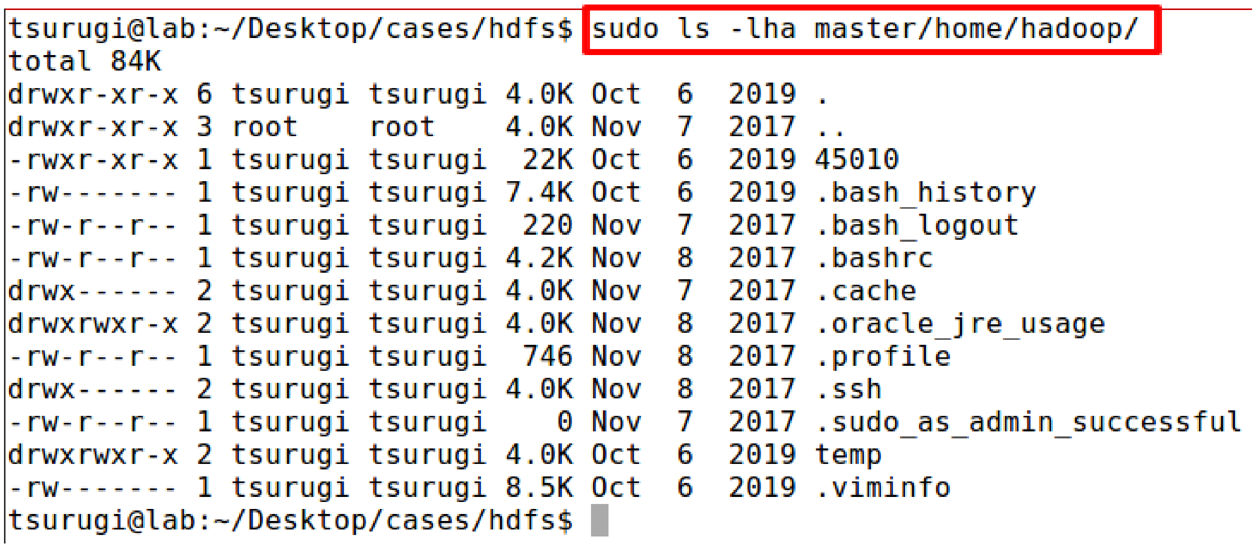
Host	Username	User ID	Home Directory	Groups
master	hadoop	1000	/home/hadoop	adm, cdrom, sudo, dip, plugdev, lxd, lpadmin, sambashare, and hadoop
slave1				
slave2				

3.3 Home Directories

We need to explore each user and their home directory. Use the information we found in the previous step to help find and investigate each user. Let's start with the master server.

Use the command below to list all the contents in the user hadoop's home directory. Make sure to use the correct command line options to list all files/directories even those that are hidden as seen in figure 3.2.

$ sudo ls -lha master/home/hadoop/



```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo ls -lha master/home/hadoop/
total 84K
drwxr-xr-x 6 tsurugi tsurugi 4.0K Oct  6  2019 .
drwxr-xr-x 3 root    root    4.0K Nov  7  2017 ..
-rwxr-xr-x 1 tsurugi tsurugi  22K Oct  6  2019 45010
-rw------- 1 tsurugi tsurugi 7.4K Oct  6  2019 .bash_history
-rw-r--r-- 1 tsurugi tsurugi  220 Nov  7  2017 .bash_logout
-rw-r--r-- 1 tsurugi tsurugi 4.2K Nov  8  2017 .bashrc
drwx------ 2 tsurugi tsurugi 4.0K Nov  7  2017 .cache
drwxrwxr-x 2 tsurugi tsurugi 4.0K Nov  8  2017 .oracle_jre_usage
-rw-r--r-- 1 tsurugi tsurugi  746 Nov  8  2017 .profile
drwx------ 2 tsurugi tsurugi 4.0K Nov  8  2017 .ssh
-rw-r--r-- 1 tsurugi tsurugi    0 Nov  7  2017 .sudo_as_admin_successful
drwxrwxr-x 2 tsurugi tsurugi 4.0K Oct  6  2019 temp
-rw------- 1 tsurugi tsurugi 8.5K Oct  6  2019 .viminfo
tsurugi@lab:~/Desktop/cases/hdfs$ 
```

Figure 3.2 - Contents of the Hadoop User on Master Server

Explore the directory and its contents, especially those bash files, the ssh directory, temp, the viminfo and definitely that file which is named 45010!

Q3.6) Did you find anything useful? Explain…

Spend some time checking the contents of the **.bash_history** (a file that is used to store a history of all the commands used on a Linux system). The **dot** at the beginning of the file, denotes that this is a hidden file. I'm sure you found something interesting! 🙂

Q3.7) Can you correlate that with the contents found on the slave servers and did you find anything weird?

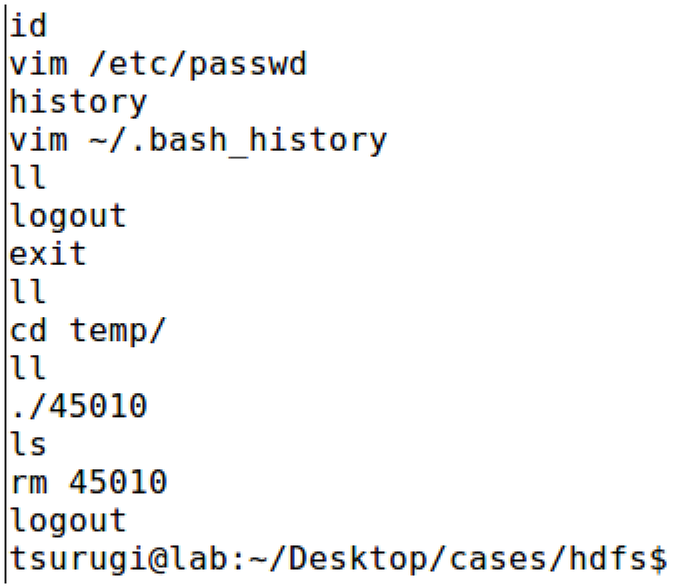
**Move on to investigate the root user and document your findings.

Use the command below to show the contents of the .bash_history file of the **hadoop** user on the **slave1** server, but make sure to use the tee command with it too as seen below.

```
$ sudo cat slave1/home/hadoop/.bash_history | tee results/slave1-hadoop-bash_history.txt
```

The tee command allows us to copy the content we dumped on the terminal to a file at the same time. Do the same for all other users to gather information about each one of them and what they did on those systems.

From the bash history for the user **hadoop** on **slave1**, it seems the user edited the passwd file using vim (totally weird!!) and rm a file of interest before logging out of the system. A snippet of what has been found can be seen in figure 3.3.



```
id
vim /etc/passwd
history
vim ~/.bash_history
ll
logout
exit
ll
cd temp/
ll
./45010
ls
rm 45010
logout
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 3.3 - Snippet from Bash History of Hadoop User on Slave1

Use the history to dig deeper with your investigation, since they could be a good map to check what and where to look for activity. After that move to the next task.

Task #4: Working with The Sleuth Kit (TSK)

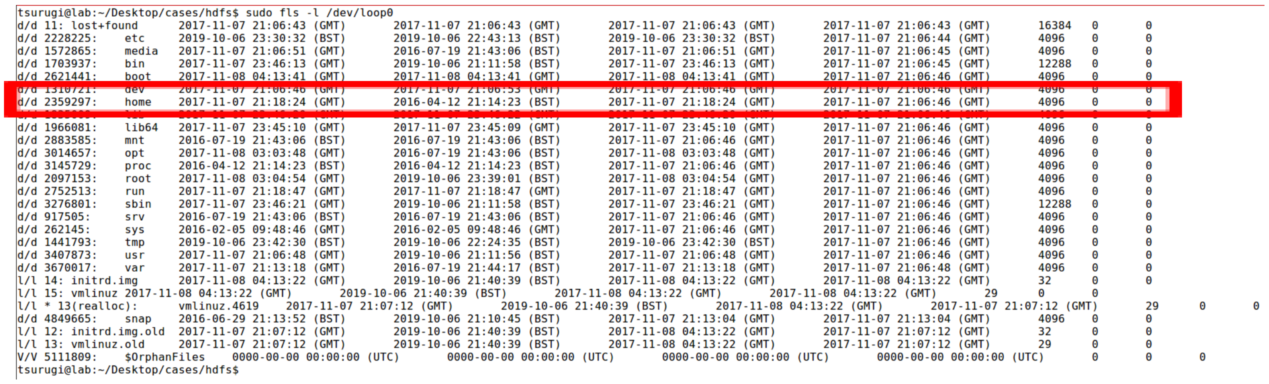
The idea of this task is to get familiar with basic TSK tools available. Let us start by listing the contents of the hadoop user directory found on the master server. There are two ways to do that, let's start with the hard way 😀

4.1 Listing Files

We first need to get the inode number for the home directory and then use that to get the inode number for the hadoop directory. But before you start, TSK deals with volumes/disks not with volumes that are mounted! In other words, we cannot use TSK directly with files in our mountpoint directories (master, slave1, and slave2). So we will be using the loop devices we saw in task #1 section 1.3.

Using the info. as we previously found, we can now use the fls command to list the files and directories within the root of the file system mounted to /dev/loop0 as you can see in figure 4.1.

$ sudo fls -l /dev/loop0



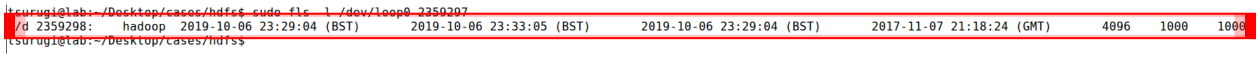
```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo fls -l /dev/loop0
d/d 11: lost+found      2017-11-07 21:06:43 (GMT)       2017-11-07 21:06:43 (GMT)       2017-11-07 21:06:43 (GMT)       2017-11-07 21:06:43 (GMT)       16384   0       0
d/d 2228225:    etc     2019-10-06 23:30:32 (BST)       2019-10-06 22:43:13 (BST)       2019-10-06 23:30:32 (BST)       2017-11-07 21:06:44 (GMT)       4096    0       0
d/d 1572865:    media   2017-11-07 21:06:51 (GMT)       2016-07-19 21:43:06 (BST)       2017-11-07 21:06:51 (GMT)       2017-11-07 21:06:45 (GMT)       4096    0       0
d/d 1703937:    bin     2017-11-07 23:46:13 (GMT)       2019-10-06 21:11:58 (BST)       2017-11-07 23:46:13 (GMT)       2017-11-07 21:06:45 (GMT)       12288   0       0
d/d 2621441:    boot    2017-11-08 04:13:41 (GMT)       2017-11-08 04:13:41 (GMT)       2017-11-08 04:13:41 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 1310721:    dev     2017-11-07 21:06:46 (GMT)       2017-11-07 21:06:53 (GMT)       2017-11-07 21:06:46 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 2359297:    home    2017-11-07 21:18:24 (GMT)       2016-04-12 21:14:23 (BST)       2017-11-07 21:18:24 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d [illegible]:    lib     [illegible]
d/d 1966081:    lib64   2017-11-07 23:45:10 (GMT)       2017-11-07 23:45:09 (GMT)       2017-11-07 23:45:10 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 2883585:    mnt     2016-07-19 21:43:06 (BST)       2016-07-19 21:43:06 (BST)       2017-11-07 21:06:46 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 3014657:    opt     2017-11-08 03:03:48 (GMT)       2016-07-19 21:43:06 (BST)       2017-11-08 03:03:48 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 3145729:    proc    2016-04-12 21:14:23 (BST)       2016-04-12 21:14:23 (BST)       2017-11-07 21:06:46 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 2097153:    root    2017-11-08 03:04:54 (GMT)       2019-10-06 23:39:01 (BST)       2017-11-08 03:04:54 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 2752513:    run     2017-11-07 21:18:47 (GMT)       2017-11-07 21:18:47 (GMT)       2017-11-07 21:18:47 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 3276801:    sbin    2017-11-07 23:46:21 (GMT)       2019-10-06 21:11:58 (BST)       2017-11-07 23:46:21 (GMT)       2017-11-07 21:06:46 (GMT)       12288   0       0
d/d 917505:     srv     2016-07-19 21:43:06 (BST)       2016-07-19 21:43:06 (BST)       2017-11-07 21:06:46 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 262145:     sys     2016-02-05 09:48:46 (GMT)       2016-02-05 09:48:46 (GMT)       2017-11-07 21:06:46 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 1441793:    tmp     2019-10-06 23:42:30 (BST)       2019-10-06 22:24:35 (BST)       2019-10-06 23:42:30 (BST)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 3407873:    usr     2017-11-07 21:06:48 (GMT)       2019-10-06 21:11:56 (BST)       2017-11-07 21:06:48 (GMT)       2017-11-07 21:06:46 (GMT)       4096    0       0
d/d 3670017:    var     2017-11-07 21:13:18 (GMT)       2016-07-19 21:44:17 (BST)       2017-11-07 21:13:18 (GMT)       2017-11-07 21:06:48 (GMT)       4096    0       0
l/l 14: initrd.img      2017-11-08 04:13:22 (GMT)       2019-10-06 21:40:39 (BST)       2017-11-08 04:13:22 (GMT)       2017-11-08 04:13:22 (GMT)       32      0       0
l/l 15: vmlinuz 2017-11-08 04:13:22 (GMT)       2019-10-06 21:40:39 (BST)       2017-11-08 04:13:22 (GMT)       2017-11-08 04:13:22 (GMT)       29      0       0
l/l * 13(realloc):      vmlinuz.4619    2017-11-07 21:07:12 (GMT)       2019-10-06 21:40:39 (BST)       2017-11-08 04:13:22 (GMT)       2017-11-07 21:07:12 (GMT)       29      0       0
d/d 4849665:    snap    2016-06-29 21:13:52 (BST)       2019-10-06 21:10:45 (BST)       2017-11-07 21:13:04 (GMT)       2017-11-07 21:13:04 (GMT)       4096    0       0
l/l 12: initrd.img.old  2017-11-07 21:07:12 (GMT)       2019-10-06 21:40:39 (BST)       2017-11-08 04:13:22 (GMT)       2017-11-07 21:07:12 (GMT)       32      0       0
l/l 13: vmlinuz.old     2017-11-07 21:07:12 (GMT)       2019-10-06 21:40:39 (BST)       2017-11-08 04:13:22 (GMT)       2017-11-07 21:07:12 (GMT)       29      0       0
V/V 5111809:    $OrphanFiles    0000-00-00 00:00:00 (UTC)       0000-00-00 00:00:00 (UTC)       0000-00-00 00:00:00 (UTC)       0000-00-00 00:00:00 (UTC)       0       0       0
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 4.1 - Listing Volume Content using fls

The inode number is the number you will find in the 2nd column. The inode number for the "**lost+found**" directory is **11** and for the boot directory is **2621441**, and so on and so forth.

Now, from the results we can see that the home directory has the inode number **2359297**. We can now use that with the fls command to list the contents of that directory as seen in figure 4.2.

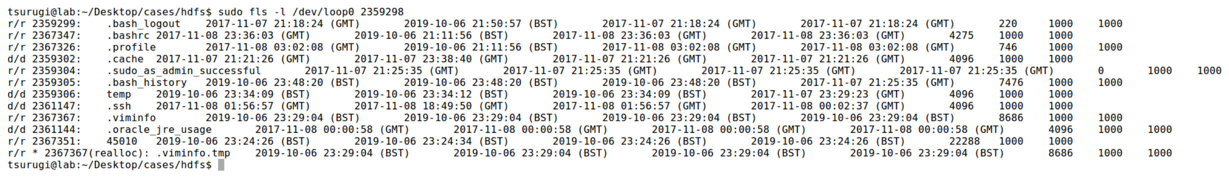
$ sudo fls -l /dev/loop0 2359297



```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo fls -l /dev/loop0 2359297
/d 2359298:     hadoop  2019-10-06 23:29:04 (BST)       2019-10-06 23:33:05 (BST)       2019-10-06 23:29:04 (BST)       2017-11-07 21:18:24 (GMT)       4096    1000    1000
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 4.2 - Listing Directory Content using fls

One more time using the inode number of the hadoop directory which is 2359298 to list the content of the directory, which can be seen in figure 4.3.



```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo fls -l /dev/loop0 2359298
r/r 2359299:    .bash_logout    2017-11-07 21:18:24 (GMT)       2019-10-06 21:50:57 (BST)       2017-11-07 21:18:24 (GMT)       2017-11-07 21:18:24 (GMT)       220     1000    1000
r/r 2367347:    .bashrc 2017-11-08 23:36:03 (GMT)       2019-10-06 21:11:56 (BST)       2017-11-08 23:36:03 (GMT)       2017-11-08 23:36:03 (GMT)       4275    1000    1000
r/r 2367326:    .profile        2017-11-08 03:02:08 (GMT)       2019-10-06 21:11:56 (BST)       2017-11-08 03:02:08 (GMT)       2017-11-08 03:02:08 (GMT)       746     1000    1000
d/d 2359302:    .cache  2017-11-07 21:21:26 (GMT)       2017-11-07 23:38:40 (GMT)       2017-11-07 21:21:26 (GMT)       2017-11-07 21:21:26 (GMT)       4096    1000    1000
r/r 2359304:    .sudo_as_admin_successful       2017-11-07 21:25:35 (GMT)       2017-11-07 21:25:35 (GMT)       2017-11-07 21:25:35 (GMT)       2017-11-07 21:25:35 (GMT)       0       1000    1000
r/r 2359305:    .bash_history   2019-10-06 23:48:20 (BST)       2019-10-06 23:48:20 (BST)       2019-10-06 23:48:20 (BST)       2017-11-07 21:25:35 (GMT)       7476    1000    1000
d/d 2359306:    temp    2019-10-06 23:34:09 (BST)       2019-10-06 23:34:12 (BST)       2019-10-06 23:34:09 (BST)       2017-11-07 23:29:23 (GMT)       4096    1000    1000
d/d 2361147:    .ssh    2017-11-08 01:56:57 (GMT)       2017-11-08 18:49:50 (GMT)       2017-11-08 01:56:57 (GMT)       2017-11-08 00:02:37 (GMT)       4096    1000    1000
r/r 2367367:    .viminfo        2019-10-06 23:29:04 (BST)       2019-10-06 23:29:04 (BST)       2019-10-06 23:29:04 (BST)       2019-10-06 23:29:04 (BST)       8686    1000    1000
d/d 2361144:    .oracle_jre_usage       2017-11-08 00:00:58 (GMT)       2017-11-08 00:00:58 (GMT)       2017-11-08 00:00:58 (GMT)       2017-11-08 00:00:58 (GMT)       4096    1000    1000
r/r 2367351:    45010   2019-10-06 23:24:26 (BST)       2019-10-06 23:24:34 (BST)       2019-10-06 23:24:26 (BST)       2019-10-06 23:24:26 (BST)       22288   1000    1000
r/r * 2367367(realloc): .viminfo.tmp    2019-10-06 23:29:04 (BST)       2019-10-06 23:29:04 (BST)       2019-10-06 23:29:04 (BST)       2019-10-06 23:29:04 (BST)       8686    1000    1000
tsurugi@lab:~/Desktop/cases/hdfs$
```

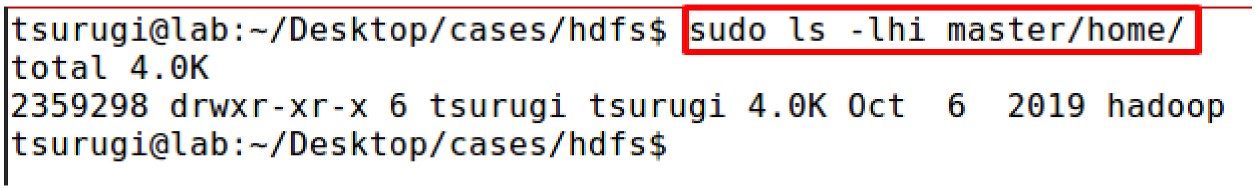
Figure 4.3 - Listing Home Directory Content using fls

Q4.1) What was the inode number for the file named “known_hosts” which is found within the .ssh directory? Use the same method as we have done so far.

The easiest way to find the inode of a file is to use the “-i” option with the ls command as seen in the command below.

$ sudo ls -lhi master/home/

How easy is that? See the results in figure 4.4 😀



```
tsurugi@lab:~/Desktop/cases/hdfs$ sudo ls -lhi master/home/
total 4.0K
2359298 drwxr-xr-x 6 tsurugi tsurugi 4.0K Oct  6  2019 hadoop
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 4.4 - Listing inode Number of Hadoop Directory

4.2 Finding Files

Now, you might be thinking, how can we check if those inodes we found truly belong to those files or directories? Well, the good thing is, that TSK comes with a command named "ffind" where we can use the inode number to find which file it is pointing to. Let's check the inode no of the hadoop directory:

```
$ sudo ffind   /dev/loop0   2359298
```

Now I want you to go back and look at the contents of the **hadoop** user home directory but focus on the .viminfo.tmp file. Use can use the pipe "|" with the grep command to help narrow down your search as seen below.

```
$ sudo fls -l /dev/loop0 2359298 | grep viminfo
```

Q4.2) How many files did you find that have that name and why? Could you explain?

Q4.3) What does **realloc** mean here (README)?

Use the inode# of that file and search for what file does it belong to, using the same approach above.

Q4.4) What did you find the inode number 2367367 of the (realloc) file truly belongs to?

4.3 Extracting Files

We can extract or dump a file from the volume or whatever you want to call it, using the TSK's icat command as seen below.

```
$ sudo icat /dev/loop0 2367367  | tee results/master-hadoop-viminfo.txt
```

Reminder: the command above will concatenate the output of the file to standard output (stdout) and using the tee command we can also copy the output to a file of our name, which was master-hadoop-viminfo.txt in the command above.

Let's do another example but this time for the 45010 file. This time we will be using the > to redirect the output instead of the tee command since this is a binary file and we won't benefit from dumping it's content to the terminal. Most of the content will be not human readable, so let's do it as seen below.

```
$ sudo icat /dev/loop0 2367351  > results/master-hadoop-45010.bin
```

Now check the file's type and it's strings content with both the file and strings commands:

```
$ file results/master-hadoop-45010.bin
```

Q4.5) What type of file did you find this file to be?

Then run the following command:

```
$ strings results/master-hadoop-45010.bin
```

Q4.6) Did you find anything referring to this file being an exploit? Show proof.

Let's compare the hashes for the file we just extracted and the file found directly in the master/home/hadoop/ directory. This can be done using the following command:

```
$ md5sum results/master-hadoop-45010.bin master/home/hadoop/45010
```

Show proof that both files match each other.

4.4 Deleted Files

You are on your own with this task. You are required to use what we have learned so far to check for the file we saw deleted in task #3 section 3.3. Yes, we are referring to the **45010** file that we saw was deleted from both servers **slave1** and **slave2**.

Show all your steps to the following:

1. Getting the inode number of the temp directory
2. What is the inode number of the 45010 file
3. Using the inode number for the file, does it map back to the 45010 file or not?
4. Use the icat command to extract the contents of the file and save it to results/slave1-hadoop-45010.bin
5. Check the file type and explain your findings
6. Check the file content using strings and explain your findings
7. What do you think is going on here?

Please use the same methods mentioned above to do other experiments and make sure you're comfortable with using the TSK commands we've covered until now. Do not move forward without understanding everything before this message. In the end, our goal is to learn, not just to run commands and find the answer to the questions!

Task #5: Data Recovery / File Carving

In this task we want to recover the files that have been deleted, especially the file that was deleted based on the commands we found being used in the **.bash_history** file.

The BAD NEWS, unfortunately on an EXT4 file system, once the file is deleted, the metadata that points to the file is zeroed out and there are no longer any pointers pointing back to the volume.

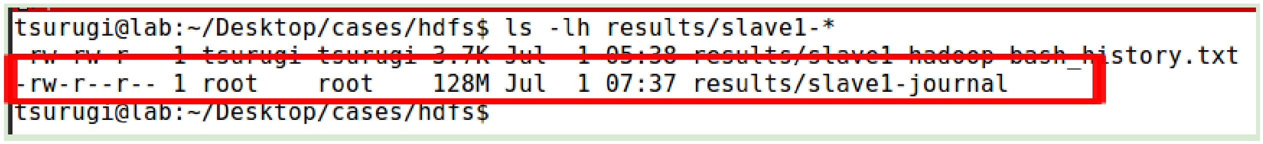
5.1 Dumping EXT4 Journal

The GOOD NEWS, is let us assume that you manage to get your hands on the system before any of the deleted files metadata was overwritten, then we might be able to recover that data with the help of the file system's journal. If this method does not work, let's say because you arrived at the crime scene late or this was an operation that happened a couple of months ago, then we still could probably apply file carving techniques to extract the deleted files, as long as they have not been overwritten.

Therefore, let us go with option (a) and use the journal to help us recover the files. To extract the journal, we will be using debugfs and asking it to dump the file with the inode **#8**, which is the inode number for the file system's journal. This can be done as:

```
$ sudo debugfs -R 'dump <8> ./results/slave1-journal' /dev/loop1
```

You should end up with a **128MB** file (size of the EXT4 journal) as seen in figure 5.1.



```
tsurugi@lab:~/Desktop/cases/hdfs$ ls -lh results/slave1-*
-rw-rw-r-- 1 tsurugi tsurugi 3.7K Jul  1 05:38 results/slave1-hadoop-bash_history.txt
-rw-r--r-- 1 root    root    128M Jul  1 07:37 results/slave1-journal
tsurugi@lab:~/Desktop/cases/hdfs$
```

Figure 5.1 - Verifying Size of File System Journal

Now we want to search for files that were deleted between October 5th 2019 and October 8th 2019 based on the case brief that was given to us. Therefore, let us define a variable with that value:

```
AFTER=$(date -d"2019-10-05 00:00:00" +%s)

BEFORE=$(date -d"2019-10-08 00:00:00" +%s)
```

5.2 Targeted Data Recovery

Before attempting the recovery step, I would like you to check or list what files actually we can recover with the help of the journal for example from the /home/hadoop directory. This can be done using the command below (please adjust the values AFTER & BEFORE with the corresponding numbers from the commands above):

```
$ sudo ext4magic /dev/loop1 -a $AFTER -b $BEFORE -f /home/hadoop -j results/slave1-journal -l
```

Q5.1) What was the path "/home/hadoop" used in the options above for? (hint: man ext4magic)

Now, let us perform the recovery step itself instead of just listing the files that are recoverable, which could be done as seen below:

```
$ sudo ext4magic /dev/loop1 -a $AFTER -b $BEFORE -f /home/hadoop -j results/slave1-journal -r -d results/slave-recovery1/
```

Q5.2) Were you able to recover the 45010 file? Show proof with validation that the recovery was successfully done. (hint: use sha256sum)

5.3 Try to Recover All Deleted Files

Another approach would be to try recovering everything. This could also be done using ext4magic, but by providing the **-m** option, which will try and recover all the deleted files on the volume, as seen below:

```
$ sudo ext4magic /dev/loop1 -a $AFTER -b $BEFORE -f /home/hadoop -j results/slave1-journal -m -d results/slave-recovery2
```

Use:

```
$ sudo tree -L 1 results/slave-recovery2/
```

Please check the man pages for the ext4magic tool, this is truly an excellent tool with so many more features/capabilities, so what are you waiting for? Go check them out!

5.4 Deleted Exploit

Now that we have been able to find the 45010 file and we noticed inside it's content a string referring to it being an exploit.

Q5.4) Can you search and find out what this file is? (hint: use the search "45010 exploit" phrase). Explain your findings.

Now that you have found out what it is. We need another proof which is to find if this exploit has truly been executed on these systems or not! 😕

We are going to leave this to you to figure out. Use all the knowledge you have learned so far to find the answer.

Task #6: Finding the Persistence Mechanism

From the previous investigations we found the 45010 file and we also found out that it is a Linux Kernel exploit that has been used to gain **root** access. Let us now use it as a reference to search for other files that might have been added or modified on these systems.

6.1 Searching Based on Reference

Because we know the 45010 file still exists on the master server, we can use it as a reference to search for other files. First let's search for files that have been modified and use the 45010 as a reference. This can be done using the find command but with the **-newer** option as seen below.

```
sudo find master/ -type f -newer master/home/hadoop/45010
```

Use either the tee command or the **>** to redirect the output to a file named master-find1.txt in the results directory.

We can also search for what files have been accessed after the 45010 file using the **-anewer** option as seen below.

```
sudo find master/ -type f -anewer master/home/hadoop/45010
```

Again save the output to your results directory in a file named master-find2.txt.

6.2 Searching Based on Date Range

Now, since the file 45010 on the slave systems is actually deleted and not accessible directly without date recovery, we cannot use the previous approach. Therefore, this time we will be performing the search using a date range. Please note that we could still apply the previous approach on both **slave1** and **slave2**, but we will need to find a file that can be used as a reference and falls within the time frame of the incident.

We know that the incident happened between October the 5th and the 8th, we can use the find command again, but this time with the **-newermt** option as seen below.

```
sudo find slave1/ -type f \( -newermt "2019-10-05" -and ! -newermt "2019-10-08" \) | tee
results/slave1-find.txt
```

The brackets and the **-and** are to make sure that we search for anything newer than October 5th and October 8th. Make sure you repeat the same steps for **slave2** and you spend some time on your findings before continuing your investigation.

6.3 Checking Files Content

One of the entries that stood up in the results of the **master** server, was the cluster.php file found under the "/usr/local/hadoop/bin/" directory.

If we check its content, we will find that this file is dealing with sockets. Spend some time doing some Googling and asking questions and then answer the question below.

Q6.1) What is this file and what is it doing? Explain in detail.

Use the debugfs command to find when this file was created. An example of how to use it can be seen below. Make sure you replace the word **inode#** with the inode number for the cluster.php file.

```
$ sudo debugfs -R ‘stat <inode#>’ /dev/loop0
```

Now, check the other find results for both slave1 and slave2. Assuming you found another file that had the word "cluster" in it.

Q6.2) Which system did you find it on and what was the content of this file?

Do some research and then explain your findings and what is this doing or achieving before you move on to the next task.

Note: the instructors are also here to help you 😉

Task #7: Checking System Logs

In this task we move our focus completely on logs and log analysis. We will be using very simple techniques, so don't worry 🙂. Log files on a Linux system are found under the /var/log directory. This directory contains many different log files depending on the system you are investigating, but in general on a Linux Ubuntu flavor they will include the following:

- Main log file is syslog (named messages on other systems)
- User activity logs can be found in auth.log, wtmp, btmp, lastlog, faillog, etc
- Kernel logs can be found in kern.log (dmesg on live systems)
- Others depending on what systems/applications are installed on the system you may find Apache, MySQL, PHP, Mail, etc log files

7.1 Installed Packages

Now since we know this is a Ubuntu system, we also know that whenever a package is installed using the apt or **dpkg** package manager, it will leave a log entry within dpkg.log file and the history of the installation activity will be in the history.log file under the apt/ log directory found within the log/ directory itself. So let us check it out…

We can use the tail command to show us the last couple of entries within the file as seen in the command below.

```
$ sudo tail master/var/log/dpkg.log
```

If we add the **-n20** or **-n30** option, we can even see the last 20 line entries or last 30 line entries, depending on the number of entries you choose. If you examine the results, you will see that the **php** and its related packages were installed. Before we move on to another file, explore the contents using the less command and see if you can find any other activity within our time period. This can be done as seen below and then using the up and down arrows to scroll through the file.

```
$ sudo less master/var/log/dpkg.log
```

Q7.1) Did you find any weird installations happening during the investigation time period?

Now check the history file using the command below.

```
sudo tail -n20 master/var/log/apt/history.log
```

Q7.2) When was the php package installed and what other packages were installed with it (list at least three of them)?

Make sure you check the same locations on the other systems, don't skip those, these steps might just show you one part, so don't miss the others.

7.2 User Login Activity

First log we are going to check is the wtmp and btmp. so first let us do it this way:

```
$ sudo last -f master/var/log/wtmp
```

Now again but with head (hope you notice the difference):

```
$ sudo last -f master/var/log/wtmp | head -n 10
```

Q7.3) Who was the last user to login to the system?

Q7.4) From where did the login happen?

Now the btmp file (failed login attempts). Do the same as before without using head and with sudo powers. Please spend some time carefully going through this log file, it is very important 😊 .

```
$ sudo last -f master/var/log/btmp | head -n 20
```

Q7.5) Why are there so many failed login attempts, what do you think is happening? Explain your answer.

Check the auth.log file and explain what happened:

```
$ sudo cat master/var/log/auth.log
```

Q7.6) does it match the activity that you saw in the previous log (btmp)?

Q7.7) Was the user successful in obtaining access using this method? Explain with proof.

Search for the line that has the text “Failed password for invalid user magnos”. (README)

Find out what happened immediately after that and see if it all makes sense now.

Q7.8) Explain what happened and at what time.

Let us also check the lastlog file using the “strings” command:

```
$ strings master/var/log/lastlog
```

Q7.9) What IP Address did you find?

Q7.10) Does it match the IP Address you saw in any of the previous log files? Please document all files that you found that IP address in, you will need them for your final report.

We are unaware of a tool to read the lastlog file offline, but use the binary structure for the lastlog file to read its records. The structure can be found below:

4 bytes	→	timestamp
32 bytes	→	terminal
256 bytes	→	hostname

By now, you should have found how this threat actor gained access to the master system and hopefully into other systems too. Let’s move on and generate a timeline to make sure this all can be correlated together and makes sense!

Task #8: Creating a Timeline

This task could be done at the beginning or the end, it depends on how you approach your cases. I am not going to say which is good and which is bad, just use the approach you feel more comfortable with.

8.1 Generating a Super Timeline

Now, to generate a super timeline for our case, we will be using the log2timeline.py framework. This could be seen in the command below:

```
$ sudo log2timeline.py -z UTC -t / --parse linux,apache_access,apt_history master.timeline master/
```

Q8.1: What does the linux parser used in the command above search for?

Q8.2: Why did we add the apache_access parser?

8.2 Filtering Your Timeline

Finally, let’s filter our timeline and sort it using the psort.py tool, which can be done as seen below.

Master:

```
sudo psort.py -z UTC -o L2tcsv -w master.csv master.timeline "date > '2019-10-05 00:00:00' AND date < '2019-10-08 00:00:00'"
```

Slave1:

```
sudo psort.py -z UTC -o L2tcsv -w slave1.csv slave1.timeline "date > '2019-10-05 00:00:00' AND date < '2019-10-08 00:00:00'"
```

Slave2:

```
sudo psort.py -z UTC -o L2tcsv -w slave2.csv slave2.timeline "date > '2019-10-05 00:00:00' AND date < '2019-10-08 00:00:00'"
```

8.3 One Timeline to Rule Them All

Make sure you remove the header from two files and then use the command line below to combine all of them.

```
cat master.csv slave1.csv slave2.csv >> timeline.csv
```

Note(s):

1. Spaces were added to the command above to help you understand it, otherwise it is not needed.
2. Use whatever tool or spreadsheet application to go through your timeline. For quick checks, I usually use Eric Zimmerman’s Timeline Explorer, but it’s up to you.

Deliverables:

1. How did the threat actor(s) gain access to the system?
   ANSWER:

2. What privileges were obtained and how?
   ANSWER:

3. What are the modifications that have been applied to the system?
   ANSWER:

4. What persistent mechanisms did the threat actor(s) use?
   ANSWER:

5. What needs to be done to restore the cluster to a fully cleaned and working environment?
   ANSWER:

6. Notes and recommendations
   ANSWER:

Reference(s)

- Linux DFIR Team, https://linuxdfir.ashemery.com