Host-based Intrusion Detection System in Shell Script

3809ICT: Applied Network Security

Zane Keeley

s2960235

1 Introduction

The purpose of this project is to develop an Intrusion Detection System (IDS) which can detect unauthorized modifications within a given database. Being able to detect intrusions within a database or system are extremely important for the purposes of security – as it can provide insights regarding where your system is vulnerable and trigger appropriate responses to protect the information of the system, which may include automatic software responses (shutting down a system, rehashing, detecting patterns to prevent a potential threat etc).

The most common types of IDS are as follows - Network Based Intrusion Prevention Systems (NPIS), Wireless Intrusion Prevention Systems (WIPS), Network Behavior Analysis (NBA) and Host Based Intrusion Detection System (HIPS). The NPIS monitors an entire network through analyzing the traffic (protecting multiple connected workstations), an NBA monitors a network for unusual traffic using pattern detections (such as DDoS attacks), a WIPS monitors the traffic of wireless networks and a HIPS monitors the traffic within a particular workstation.

For my IDS, I developed a HIPS based system which can detect file permission changes, content changes, modifier changes (such as the last date a file was modified, the owner of a file or the size of a file) and any files which have been created or deleted within the working directory being maintained. When an intrusion is detected, the user is made aware of the category of intrusion, and any of the relevant files which were impacted within each category.

Through completing this project, I learnt a lot regarding how IDS systems work – such as storing a state and using encryption that utilizes the same algorithm to compare unencrypted fields. I also learnt a lot more surrounding shell / Linux based programming, and how it can be used to process files and directories, and benefit the security of a system.

2 Platform and Tools

The programming / development for my IDS was completed using the bash scripting language, where I utilized the following commands:

- Awk – retrieving a particular argument of an output. This was useful for processing the output of commands and retrieving specific information (such as the file names or permissions of a file).

- Grep – matching outputs. Grep was utilized to compare strings formed within the current state to the clean state, where inconsistencies indicate something has changed between the two states. Grep was also very useful for retrieving specific information – as I used Grep to search for identifiers held within my clean state to quickly find the required information.

- Openssl sha1 – The bash command for SHA1 was used to create hash values of the fields I wanted to compare. I used the SHA1 hash function on the permissions, modifiers and file contents within my IDS working directory, and stored them within my clean state. Later – I used the SHA1 hash function to create hash values for verification, which could be compared to my clean state to detect intrusion.

- Ls – the ls command was used to store a record of the files / directories held within the directory being maintained, and collecting their relevant information using the –l option.

Other basic built ins operations within shell such as looping, redirection and case checking were also very helpful within the structure of program.

3 Implementation and Testing

Verification (-c option) :

The first section of my program is built around developing a 'clean state' of the current working directory, which uses the –c option (**c**lean state), and acts as a record for the current files / directories. My clean state stores the file / directory names within the current directory and the hash values for the permissions, modifiers and contents of each.

The clean state and IDS program are kept outside of the directory being maintained – to simulate a workbench using external or purchased software operating on one of their directories.

To create a clean state, my program completes the following stages:

- **Step 1**: Deletes the existing content of the previous clean state

- **Step 2**: Stores a record of the files / directories currently stored within the working directory, which will be used later to detect files which have been created or deleted.

- **Step 3**: Create an identifier and hash function for the permissions of each file. This is done by appending 'p:' to the start of each file or directory, before the result is hashed. This hides which identifier corresponds to which file and can be used to easily find and compare the permissions for the file. If the identifier is present within the clean state file – it indicates that permissions for that file are present. Next, I retrieve the permissions for the relevant file, before they are hashed and appended to hashed identifier for the file permissions. I added permissions to the clean state separately (I.e not as part of the modifiers) – as detecting permission changes is quite important and doing it separately allowed me to specifically identify permission changes with files (as opposed to the umbrella option of detecting a file or directory has changed).

- **Step 4**: Create an identifier and hash function for the modifiers of each file. The modifiers of each file include the links, owner, group, file size and the last date / time when the file was modified. Permissions were not included as they were done separately above. The identifier was created using the same process as in step 1, with 'm:' being added to the front of the file name. The 'm:' was added to be able to efficiently find and compare the modifiers of files within the clean state, and detect that modifiers for a specific file are present within the clean state. Once the identifiers were completed, I collected the modifiers for each file using ls –l and removed the filename / permissions, before hashing the result and printing it next to the identifier within the clean state.

- **Step 5**: Create an identifier and hash function for the contents of each file. The identifier was created the same way as in Step 1, but 'c:' was added to the front of the file name. The 'c:' was added to be able to easy identify files within the clean state related to the contents of a file. Once the identifier was completed, I hashed the contents of each file using SHA1 and printed the output next to the files relevant identifier. Comparing the hash values of file contents was added as an extra layer of protection (some file changes will be detected within the modifiers).

Intrusion Detection (-v option):

The second section of my program is designed around detected changes / intrusions since the clean state was created, and can be completed using the –v option of my program (<u>v</u>erify clean state). The verify option will create new records of all the files / directories within the current directory, and compare their hash values with those stored in the clean state. Any changes detected are listed and categorized based on whether the change was detected within permissions, the record of files (creation or deletion) or the contents / modifiers of a file.

To verify my clean state, my program completes the following stages:

- **Step 1**: Creates a variable for permissions, changes, deletion and creation. For each variable, any file / directory where intrusion is detected within the relevant category are added. If a variable is empty after our clean state is verified, it indicates no intrusion of that category took place (i.e. no permissions were changed, or no files were deleted).

- **Step 2**: Stores a record of each file within the current directory being maintained. The program then performs 2 loops:
  - Compare each record with those stored in the clean state. Any files / directories found within the records but not within the clean state indicate a file / directory which has been created, which is added to the relevant list created in Step 1.
  - Compare each of the files / directories stored in the clean state to the record retrieved. Any file / directory found within the clean state but not our retrieved records indicated a file / directory which has been deleted, which is added to the relevant list created in Step 1.

- **Step 3**: Create an identifier and hash function for the permissions of each file. The identifier is created through adding 'p:' to the start of each file and hashing the result. Next, the program retrieves the permission information of each file and appends the result to the identifier. If the hashed identifier is found within the clean state file – it indicates permissions for the relevant file are present. We ensure only 1 result is found – to prevent an intruder adding false adding false permissions and to make our program more robust (an intruder would still find it difficult to add false permissions as all values are hashed). If the hashed identifier is found, we compare the full hash (the identifier + permissions hash) to the relevant line, where if the identifier is found but the full hash is not – it indicates that the permissions for the file have been changed and the file is added to the relevant variable from Step 1.

- **Step 4**: Create an identifier and hash function for the modifiers of each file. The identifier is created through adding 'm:' to the start of each file and hashing the result. Next, the program retrieves the modifiers information of each file and appends the result to the identifier. If the hashed identifier is found within the clean state file – it indicates modifiers for the relevant file are present. We ensure only 1 result is found – to prevent an intruder adding false adding false modifiers (again, an intruder would have a hard time as all values are hashed, but this provides extra security). If the hashed identifier is found, we compare the full hash (the identifier + modifiers hash) to the relevant line, where if the identifier is found but the full hash is not – it indicates that the modifiers for the file have been changed and the file is added to the relevant variable from Step 1.

- **Step 5**: Create an identifier and hash function for the contents of each file. The identifier is created through adding 'c:' to the start of each file and hashing the result. Next, the program calculates the SHA1 hash value of the file contents and appends the result to the identifier. If the hashed identifier is found within the clean state file – it indicates contents for the relevant file are present. We ensure only 1 result is found.

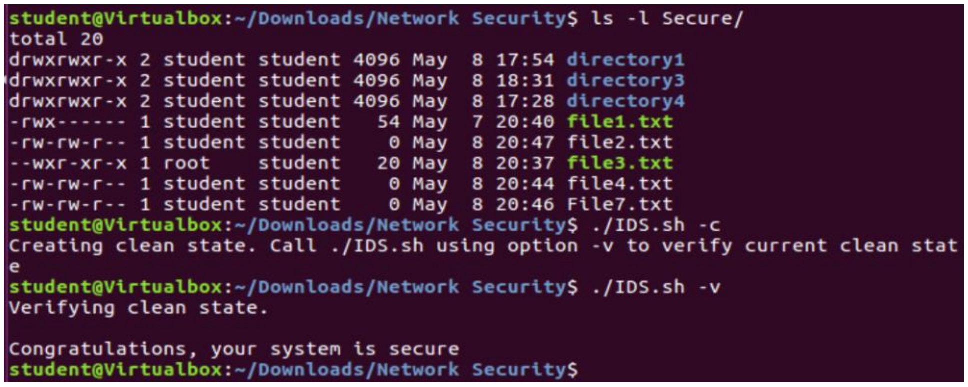
If the hashed identifier is found, we compare the full hash (the identifier + contents hash) to the relevant line, where if the identifier is found but the full hash is not – it indicates that the contents for the file have been changed and the file is added to the relevant variable from Step 1.

- **Step 6**: We analyze the variables created in Step 1 to determine which (if any) intrusions have occurred. For each category where intrusion is detected (namely, the associated variable is not empty), we print out the category and all the associated files / directories which were affected. If all variables are empty, we can assert that no intrusion has taken place – where we inform the user through printing to the terminal.

Testing:

For testing my IDS – I tested changing permissions, file contents, modifiers (I.e file owner), deleting and removing files, and making changes within the subdirectories (I.e deleting / creating files). Evidence of the test cases are provided below:

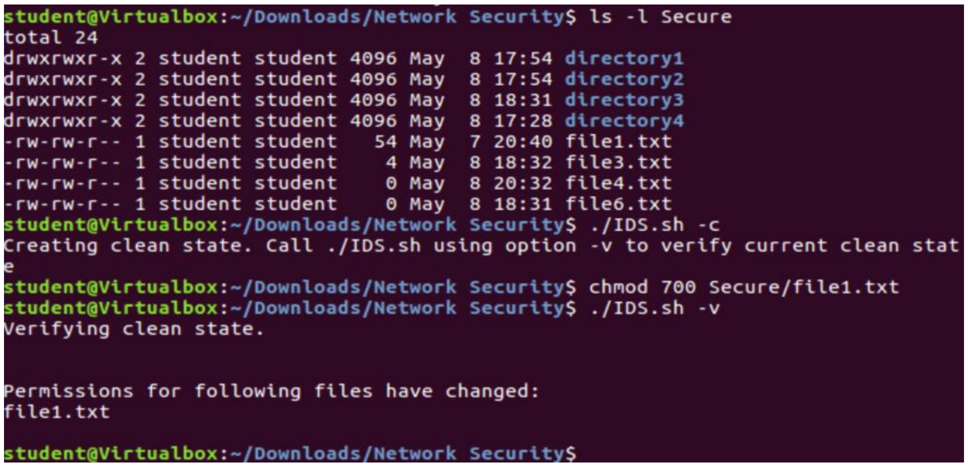
No Files Being Changed (verification of clean state is a success):



```
student@Virtualbox:~/Downloads/Network Security$ ls -l Secure/
total 20
drwxrwxr-x 2 student student 4096 May  8 17:54 directory1
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rwx------ 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 student student    0 May  8 20:47 file2.txt
--wxr-xr-x 1 root    student   20 May  8 20:37 file3.txt
-rw-rw-r-- 1 student student    0 May  8 20:44 file4.txt
-rw-rw-r-- 1 student student    0 May  8 20:46 File7.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.

Congratulations, your system is secure
student@Virtualbox:~/Downloads/Network Security$
```

Simulating an Unauthorized User Changing a File Permission:

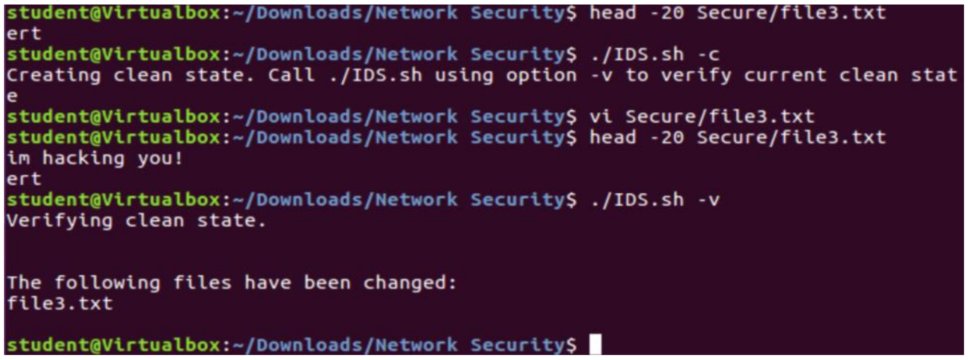


```
student@Virtualbox:~/Downloads/Network Security$ ls -l Secure
total 24
drwxrwxr-x 2 student student 4096 May  8 17:54 directory1
drwxrwxr-x 2 student student 4096 May  8 17:54 directory2
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rw-rw-r-- 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 student student    4 May  8 18:32 file3.txt
-rw-rw-r-- 1 student student    0 May  8 20:32 file4.txt
-rw-rw-r-- 1 student student    0 May  8 18:31 file6.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ chmod 700 Secure/file1.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.


Permissions for following files have changed:
file1.txt

student@Virtualbox:~/Downloads/Network Security$
```

Simulating an Unauthorized User Changing File Contents:

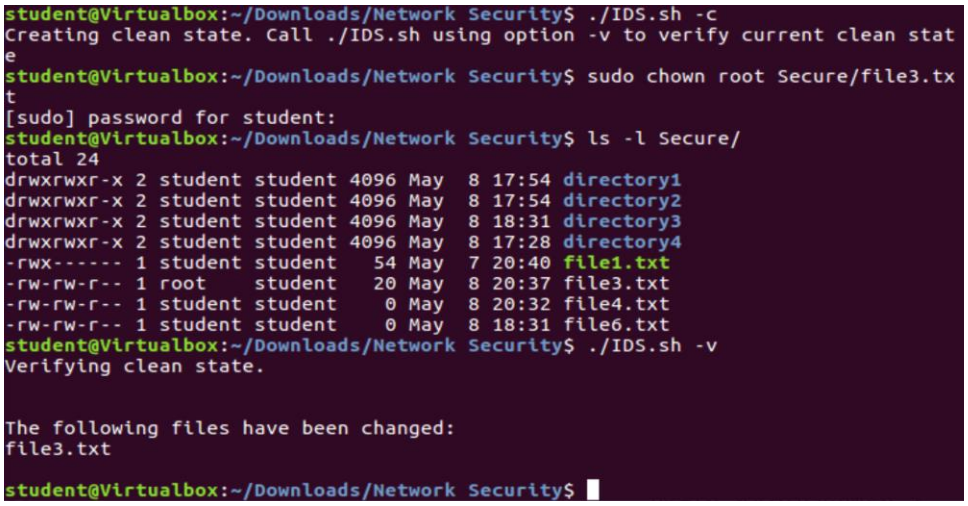


```
student@Virtualbox:~/Downloads/Network Security$ head -20 Secure/file3.txt
ert
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ vi Secure/file3.txt
student@Virtualbox:~/Downloads/Network Security$ head -20 Secure/file3.txt
im hacking you!
ert
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.


The following files have been changed:
file3.txt

student@Virtualbox:~/Downloads/Network Security$ 
```

Simulating an Unauthorized User Changing File Modifiers:

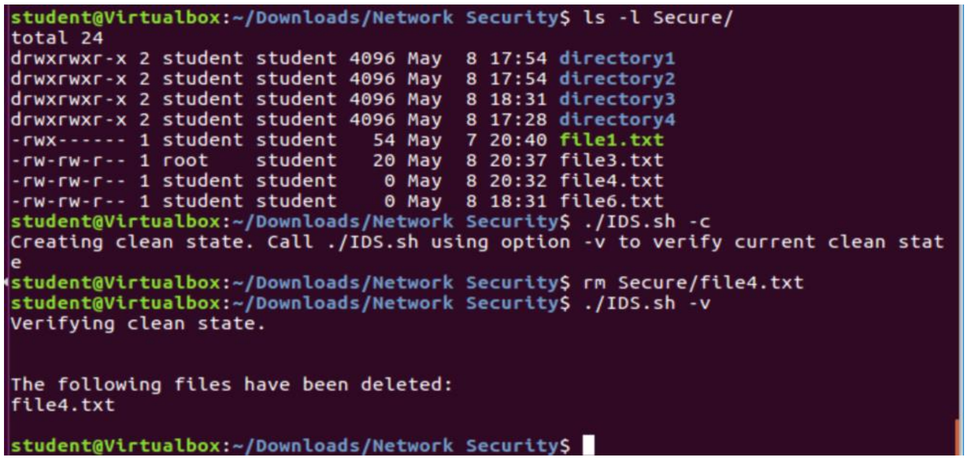


```
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ sudo chown root Secure/file3.tx
t
[sudo] password for student:
student@Virtualbox:~/Downloads/Network Security$ ls -l Secure/
total 24
drwxrwxr-x 2 student student 4096 May  8 17:54 directory1
drwxrwxr-x 2 student student 4096 May  8 17:54 directory2
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rwx------ 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 root    student   20 May  8 20:37 file3.txt
-rw-rw-r-- 1 student student    0 May  8 20:32 file4.txt
-rw-rw-r-- 1 student student    0 May  8 18:31 file6.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.


The following files have been changed:
file3.txt

student@Virtualbox:~/Downloads/Network Security$ 
```

Simulating an Unauthorized User Deleting a File:

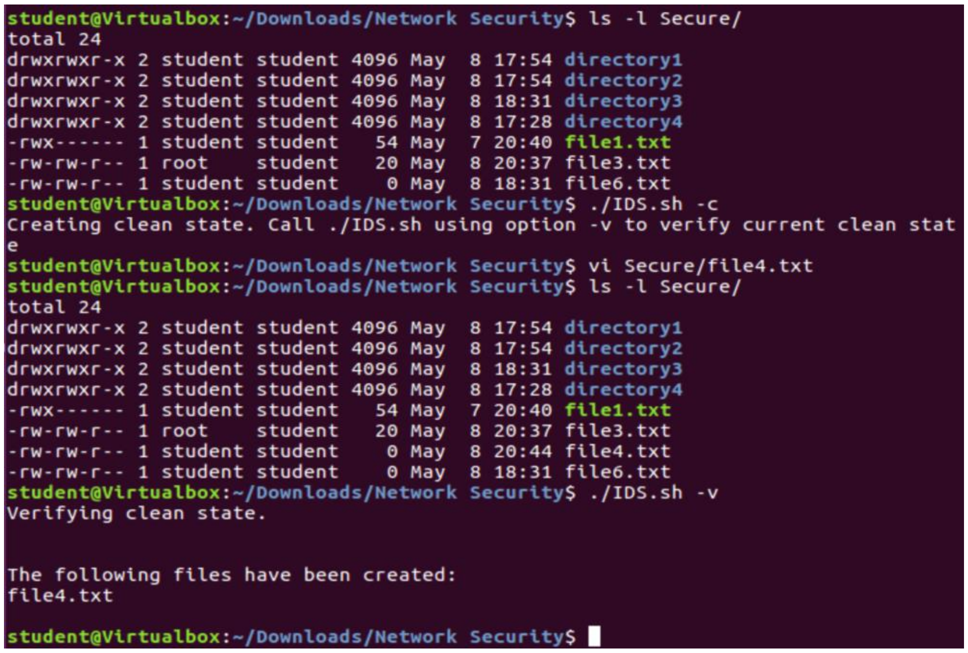


```
student@Virtualbox:~/Downloads/Network Security$ ls -l Secure/
total 24
drwxrwxr-x 2 student student 4096 May  8 17:54 directory1
drwxrwxr-x 2 student student 4096 May  8 17:54 directory2
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rwx------ 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 root    student   20 May  8 20:37 file3.txt
-rw-rw-r-- 1 student student    0 May  8 20:32 file4.txt
-rw-rw-r-- 1 student student    0 May  8 18:31 file6.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ rm Secure/file4.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.


The following files have been deleted:
file4.txt

student@Virtualbox:~/Downloads/Network Security$ 
```

Simulating an Unauthorized User Creating a New File:

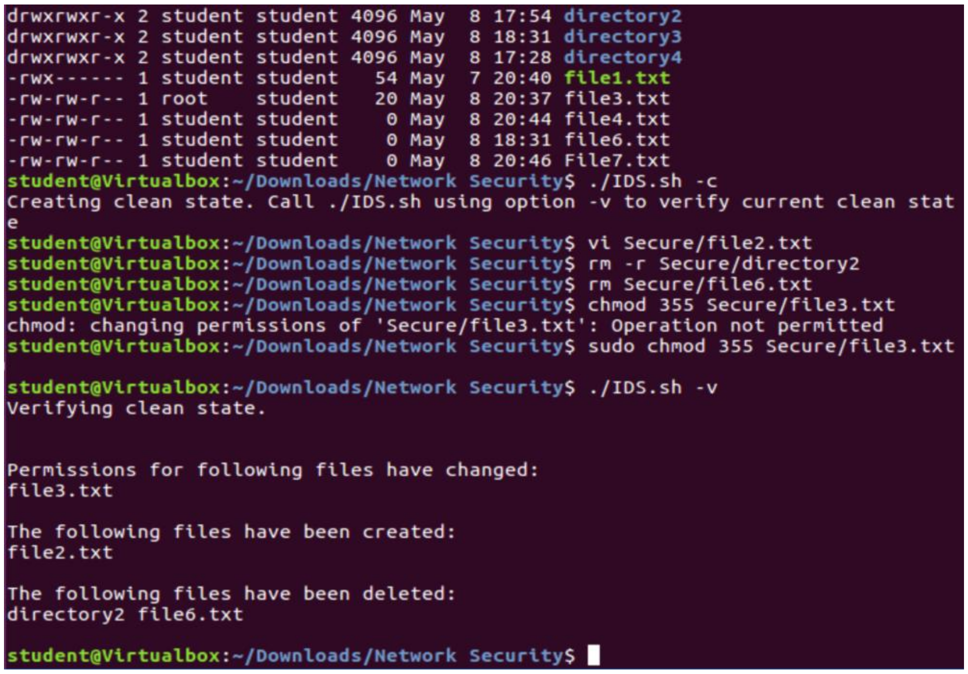


```
student@Virtualbox:~/Downloads/Network Security$ ls -l Secure/
total 24
drwxrwxr-x 2 student student 4096 May  8 17:54 directory1
drwxrwxr-x 2 student student 4096 May  8 17:54 directory2
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rwx------ 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 root    student   20 May  8 20:37 file3.txt
-rw-rw-r-- 1 student student    0 May  8 18:31 file6.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ vi Secure/file4.txt
student@Virtualbox:~/Downloads/Network Security$ ls -l Secure/
total 24
drwxrwxr-x 2 student student 4096 May  8 17:54 directory1
drwxrwxr-x 2 student student 4096 May  8 17:54 directory2
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rwx------ 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 root    student   20 May  8 20:37 file3.txt
-rw-rw-r-- 1 student student    0 May  8 20:44 file4.txt
-rw-rw-r-- 1 student student    0 May  8 18:31 file6.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.


The following files have been created:
file4.txt

student@Virtualbox:~/Downloads/Network Security$ 
```

Simulating an Unauthorized User Creating a New File, Deleting a File, Deleting a Directory and Changing a Files Permissions Simultaneously:



```
drwxrwxr-x 2 student student 4096 May  8 17:54 directory2
drwxrwxr-x 2 student student 4096 May  8 18:31 directory3
drwxrwxr-x 2 student student 4096 May  8 17:28 directory4
-rwx------ 1 student student   54 May  7 20:40 file1.txt
-rw-rw-r-- 1 root    student   20 May  8 20:37 file3.txt
-rw-rw-r-- 1 student student    0 May  8 20:44 file4.txt
-rw-rw-r-- 1 student student    0 May  8 18:31 file6.txt
-rw-rw-r-- 1 student student    0 May  8 20:46 File7.txt
student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -c
Creating clean state. Call ./IDS.sh using option -v to verify current clean stat
e
student@Virtualbox:~/Downloads/Network Security$ vi Secure/file2.txt
student@Virtualbox:~/Downloads/Network Security$ rm -r Secure/directory2
student@Virtualbox:~/Downloads/Network Security$ rm Secure/file6.txt
student@Virtualbox:~/Downloads/Network Security$ chmod 355 Secure/file3.txt
chmod: changing permissions of 'Secure/file3.txt': Operation not permitted
student@Virtualbox:~/Downloads/Network Security$ sudo chmod 355 Secure/file3.txt

student@Virtualbox:~/Downloads/Network Security$ ./IDS.sh -v
Verifying clean state.


Permissions for following files have changed:
file3.txt

The following files have been created:
file2.txt

The following files have been deleted:
directory2 file6.txt

student@Virtualbox:~/Downloads/Network Security$ 
```

4 Discussion

I would recommend using my system to provide a small level of security within a

I discovered the issue where it is very hard to keep a key created using shell commands secure within the working directory of a system. Common solutions involve keeping it on an external system or using an external drive to hold the key (so only the person with the drive has access to the key). However these still present issues where the external system / drive will need to keep the encryption key secure.

My system only works within 1 directory, but multiple instances could be instantiated to handle multiple directories. The program could also be expanded to work on all subdirectory files, however the amount of development required would be a very large undertaking a student participating within this course.

The clean state file and IDS can also easily be accessed by someone with access to the system where the IDS is working, so it would be beneficial if the clean state were encrypted after it is created, and decrypted when verifying the clean state – however this presents the issue of keeping the secure.

My recommendation for keeping a key secure would be a system where the key is typed in, and where it requires a password to run the system (create a clean state and verify the clean state). This would remove the need for any encryption key to be stored on the system, protect the clean state file and prevent unauthorized users creating false clean states / clean state files.

5 Conclusion

My implementation successfully compares the hash values of the permissions, content and modifiers of a file, in addition to comparing the records of the current directory with those held within the clean state to determine any file which has been added or deleted (a file within the clean state but not within the records retrieved when verifying the clean state indicates a file being deleted, whereas a file within the records retrieved when verifying the clean state but which is not within the clean state indicates a file which has been created).

My system ended up working extremely well for the tasks for the goals I set out (detecting changes in permissions, modifiers, content and files being created or deleted). I was very happy that my program was successful when several changes were taking place, and that it could also detect changes in subdirectories (such as files being deleted / created within a subdirectory) - which would be presented as a directory being changed when my program verifies the clean state.

I found the project to be very enjoyable and was very pleased to build on my not only IDS systems, but Network Security and shell programming as well.