Name : Yikwenmein Victor Magheng
Personal number: 19950218T614
course code: ET2595(assignment 2)

Introduction.

This is a report on a system integrity verifier(SIV) assignment.Changes to configurations, files and file attributes across the IT infrastructure are common, but hidden within a large volume of daily changes can be the few that impact file or configuration integrity. These changes can also reduce security posture and in some cases may be leading indicators of a breach in progress. The role of a system integrity verifier is to to effectively keep track of any of these changes. Generally, values monitored for unexpected changes to files or configuration items include but not limited to:

- Credentials
- Privileges and Security Settings
- Content
- Core attributes and size
- Hash values
- Configuration values

In reality, this is an application that performs the act of validating the integrity of a system file using a verification method between the current file state and a known good baseline state. In this assignment, the following criteria are used to make comparisons on a file's current state against a certain known good state

- File/directory path
- File /directory size
- File/directory permissions
- Name of user and group owning file/directory

- File’s checksum message digest

Some possible applications include;

- An intrusion detection environment
- Hybrid Cloud Security, Embedded Security, Security for Linux, Security for Windows Server
- Tripwire products

Design and Implementation.

In this system integrity verifier, the information which the program monitors for unexpected changes on files or directories include:

- File/directory path
- File /directory size
- File/directory permissions
- Name of user and group owning file/directory
- File’s checksum message digest

Python 3 has been used to develop this integrity verifier and it compiles on python 3 compilers. First things first, the program uses the `os.walk()` to iteratively parse through a monitored_directory pass to the `os.walk()` function as argument. os.walk() returns *three* items,

- The root directory,
- A list of directories (dirs) immediately below the current root and
- A list of files found in those directories. The documentation will give more information. “dirs” will contain a list of directories just below root, and files will contain a list of all the files found in those directories. In the next iteration, each

directory of those in the previous dirs list will take on the role of root in turn and the search will continue from there, going down a level only after the current level has been searched

1. File/directory path. To get this information, the program uses the os.walk(). For each file in files returned by the os.walk(), the file path is gotten with

```
File path = root directory + os.sep + file
```

For each subdirectory in dirs returned by os.walk(),the directory path is given by

```
Directory path = root directory + os.sep + subdirectory
```

2. Name of user and group owning file/directory. To get this information, the program uses the os. stat() function. See os.stat. It gives you st_uid which is the user id of the owner and st_gid which is the group id of owner. Then the program converts it to a more friendly name. To do that, the program uses getpwuid() and getgrgid()
3. File /directory size. The program uses the os.path.getsize() function. The function takes as argument file/directory path and returns its size in bytes. We can as well get this information from the os.stat() function.
4. File/directory last modification date: The program compute the file/directory last modification date with os.path.getmtime()function. We can as well get this information from the os.stat().st_ctime. To format the output and render it more human readable, we pass the output of os.path.getmtime() to time.ctime()

5. File/directory permissions: The program uses the st_mode returned by os.stat() function and to get the octal value of the permission, we pass it to the oct() function as follows oct(os.stat().st_mode)
6. File message digest: To compute message digest for a given hash function say md5,

```
hasher = hashlib.md5()
with open('filepath', 'rb') as afile:
    buf = afile.read()
    hasher.update(buf)
    message_digest = hasher.hexdigest())
```

For each file/directory path, the program adds it into a dictionary as a dictionary key and as it's dictionary value, the program adds another dictionary containing another dictionary containing above attributes. The dictionaries for file and directories are respectively as follows;

```
file_info[filepath] ={"Path to file":filepath,"Size of file":size,"User owning
file":owner,"Group owning file":group,"File permissions":mask,\
                          "File's last modification
date":last_modified,"Hashing_function":argument.hash_function,"message_digest":
message_digest}
```

```
file_info[directory_path] ={"Path to file":directory_path,"Size of
file":size,"User owning file":owner,"Group owning file":group,"File
permissions":mask,"File's last modification date":last_modified}
```

Finally, these dictionaries are loaded into a json file and witten into a verification text file.

```
json_file = json.dumps(file_info, indent=4)
with open(argument.verification_file, "w") as fp:
       fp.write(json_file)
```

At high level, the following pseudo code depicts how the SIV system operates.

```
parse command line arguments
if program mode is initialization

   if monitored directory exist
       check if verification and report files exist
       if True
           ask if user wants to override files
           if True
               loop through files and directories in monitored directory:
                   count files
                   count directories
                   get file/directory path
                   get file /directory size
                   get file/directory permissions
                   get name of user and group owning file/directory
                   compute file's checksum message digest
               for each file/directory;
                   write the following to the verification file
                       file/directory path
```

```
                    file /directory size
                    file/directory permissions
                    name of user and group owning file/directory
                    file's checksum message digest
                write the following to the report
                    path to monitored directory
                    path to the verification
                    number of directories parsed
                    number of files parsed
                    time taken to complete the


            if False
                exit program
        if False
        create verification and report files


    Else
    Exit program


if program mode is verification


    if monitored directory exist
        check if verification file exist and not empty
```

```
       if True

          check if report file exist

           if True;

               ask if user wants to override;

               if True

                   loop through files and directories in monitored
directory :

                       count files

                       count directories

                       get file/directory path

                       get file /directory size

                       get file/directory permissions

                       get name of user and group owning file/directory

                       compute file's checksum message digest

                   for each file/directory;

                       write a warning to the verification file if

                           file/directory path is different as on the
verification file

                           file /directory size is different as on the
verification file

                           file/directory permissions is different as on
the verification file

                           name of user and group owning file/directory is
different as on the verification file

                           file's checksum message digest is different as
on the verification file
```

```
                        count warnings issued

                    write the following to the report

                        path to monitored directory

                        path to the verification

                        number of directories parsed

                        number of files parsed

                        number of warnings issued

                        time taken to complete the

            if False:

                exit program

        if False

            exit program

    if False

    create verification and report files


Else

Exit program
```

Usage

1. **Initialization mode usage:** The following command for example runs the siv program in the initialization mode.

```
python siv.py -i -D /home/yikwenmein/Desktop/ass1 -V verification.txt -R
report.txt -H SHA-1
```

2. **Verification mode usage:** The following command for example runs the siv program in the initialization mode.

```
python siv.py -v -D /home/yikwenmein/Desktop/ass1 -V verification.txt -R
report.txt
```

Limitations

- The major limitation of the system is that it only detects changes but doesn't provide a realtime mechanism to contact intruder attempts. This SIV system will also fail in dealing with special characters in file structures.
- The system does not work in realtime. Changes will only be detected after executing the siv program. The system can become compromised but the administrator will only become aware when he/she executes the program.
- Another limitation is that the program cannot compute a message digest on folders but only on files. To compute folders, we need to install an additional library.