Introduction

In this assignment, you will implement a very simple system integrity verifier (SIV) for a Linux system. The goal of the SIV is to detect file system modifications occurring within a directory tree. The SIV outputs statistics and warnings about changes to a report file specified by the user.

The SIV can be run either in initialization mode or in verification mode.

Initialization mode

In initialization mode, the SIV program requires the user to enter a path to a monitored directory, another path to a verification file (outside the monitored directory), a third path
to a report file and a hash function (e.g., the SIV must support at least MD-5 and SHA-1 message digests). The program will do the following:
a) Verify that the specified monitored directory exists
b) Verify that the specified location of the verification file and the report file are outside the monitored directory
c) Verify that the specified hash function is supported by your SIV
d) If the verification file and/or report file exists already then they will be overwritten.
e) Recursively iterate through the directory contents (of arbitrary depth level) and for each file/directory found record the following information in the verification file (it is up to you decide the format used to store records in the verification file):
I. Full path to file/directory, including filename
II. Size of the file
III. Name of user owning the file/directory
IV. Name of group owning the file/directory
V. Access rights to the file/directory (either octal or symbolic)
VI. Last modification date
VII. Computed message digest with specified hash function over file contents
f) Write to the report file (this must be a text file) a summary of your findings:
I. Full pathname to monitored directory
II. Full pathname to verification file
III. Number of directories parsed
IV. Number of files parsed
V. Time to complete the initialization mode

Verification mode

In verification mode, the user provides the path of the verification file and a path to a report
file. The program will do the following:

a) Verify that the specified verification file exists and, if true, begin parsing the file
b) Verify that the specified verification file is outside the monitored directory
c) Verify that the specified report file is outside the monitored directory
d) If the report file exists already then it shall be overwritten by the new report.
e) Recursively iterate through the directory contents (of arbitrary depth level) and verify that it matches exactly the information from the verification file. A warning must be printed to the report file for each entry that diverges from the records in the verification file, along with information about what is different:
I. New or removed files/directories
II. Files with a different size than recorded (show old size and new size)
III. Files with a different message digest than computed before (show old digest and new digest)
IV. Files/directories with a different user/group (show old user/group and new user/group)
V. Files/directories with modified access rights (show old access rights and new access rights)
VI. Files/directories with a different modification date (show old modification date and new modification date)
f) Write to the report file (this must be a text file) a summary of your findings
I. Full pathname to monitored directory
II. Full pathname to verification file
III. Full pathname to report file
IV. Number of directories parsed
V. Number of files parsed
VI. Number of warning issued
VII. Time to complete the verification mode

The program is not allowed to change the verification file when executing in verification
mode.

User interface

The program must be runnable from the command-line and accept the following command-
line arguments:

siv <-i|-v|-h> –D <monitored_directory> -V <verification_file> -R

<report_file> -H
<hash_function>

The options -i (indicating initialization mode), -v (indicating verification mode), and –h (indicating help mode) are mutually exclusive.

The options -V and -H are mutually exclusive, meaning that you specify the hash function only when you create the verification file (in initialization mode). In verification mode, the hash function must be recovered from the verification file. The allowable values for <hash_function> are sha1 for SHA-1 digests and md5 for MD-5 digests. Please, use exactly these strings for the hash function.

When the help option (-h) is given, the program will print the accepted command-line arguments with a short explanation for each, and show an example how to run the program in initialization mode and verification mode respectively. In addition, all supported hash functions must be listed with the syntax required to specify them on the command-line.

Your program will be executed automatically using the command-line arguments list above. It must totally non-interactive, that is it may not prompt for user input. If the program fails because it does not support the required arguments or because is waiting for input from the user, the project will not be approved.

Example 1: Initialization mode with valid syntax (siv in the current working directory) ./siv -i -D important_directory -V verificationDB -R my_report.txt -H sha1
Example 2: Verification mode with valid syntax (siv in the current working directory) ./siv -v -D important_directory -V verificationDB -R my_report2.txt

In some cases (Java for example) you need to wrap the full java command line in a shell script in order to follow the syntax above. If you are in this situation read about the use of shebang https://en.wikipedia.org/wiki/Shebang_(Unix).

Implementation

You are free to use Bash shell scripting or one of the following programming languages to implement the SIV (or a combination thereof): C, C++, Java, Perl, or Python (only Python 3).
The programming language and any additional libraries or extensions you may require must be installable with apt, apt-get or pip --user. Solutions that require manually downloading the source code or binary, or manual

configuration will not be accepted. Using an already existing SIV system (such as Tripwire) is not allowed either. “The submission will be tested on server A will be tested make sure it works.”

It is strongly suggested that you consult the man pages for stat(1) and stat(2) shown in references. Stat(1) is useful to use from a shell script and stat(2) is useful from within a C-program. Other programming languages may provide simplified interfaces to stat(2) or their own API.

Please note that the SIV must not require superuser (root) access, but rather must be able to run with regular user permissions.

Testing

On GitHub https://github.com/dragos-bth/siv_tester you find the test framework. Instructions are also available there on how to install the SIV tester on Server A and run it.

“The code must be tested and it should be executed”

The best way to stress test your SIV is to download the complete Linux kernel source from http://www.kernel.org. You can uncompress the source code XZ archive with the command:

```
tar xvfJ <filename.tar.xz>
```

You should try changing files as outlined above under section “Verification mode” and check
that your SIV can find the changes. Make sure all the required types of changes can be
detected

Deliverables and evaluation

You will deliver 1) a lab report (as PDF
file) and 2) the complete SIV source code as a tar.gz archive.

The delivered source code will be compiled to an executable if necessary. It is therefore imperative that your code compiles (unless it is a script) and that you document which compiler/interpreter version must be used. If your code requires libraries or extension, then provide instructions how to install them (remember, only things installable with apt, apt-get or pip --user). Make sure your program can execute on serverA. Specific sets of data will be used to test the functionality of the code. You will not have access to these sets of data during code development.

It is expected that authors can explain every line in the submitted program as well as the format of the verification file(doc strings).

The report must have the following structure:
1. Introduction.
Here you explain the goal of the project and the possible applications of the SIV
2. Design and implementation
Here you explain how you detect each type of change (all 6 types) that can occur to the records. I expect detailed algorithm descriptions with pseudo-code. For example, if your program can detect the last modification time for a file, I expect that you describe the function or the utility you use to obtain that information from. Also, I expect you to specify in detail the format of the verification file. The level of detail must be such that one should be able write a program that interprets the contents of the verification file without guessing or looking into your code. For example, if each record (i.e., file-related data) occupies one row in the verification file, describe the meaning of each field including the appropriate data type for it (e.g., string terminated by zero, integer, floating point, hex string etc.). Explain also which programming language you have chosen and why. If there are any software dependencies required to run your program, make sure you provide instruction how to install them in serverA, so I can prepare a similar environment when testing your program. Remember, only things installable with apt, apt-get or pip –user.

3. Usage
A short manual how to use your SIV, including examples for the most common operations.
4. Limitations
Explain if your program has any limitations, if any, in terms of the tasks it must perform. These limitations must be limited special cases (e.g., special characters in file names) as your program must implement all functionality described in this document .

References

1. stat(1): man 1 stat
http://linux.die.net/man/1/stat
2. stat(2): man 2 stat, or man fstat
http://linux.die.net/man/2/stat
3. md5sum(1): man 1 md5sum
http://linux.die.net/man/1/md5sum
4. evp_digestinit(3): man 3 evp_digestinit
http://linux.die.net/man/3/evp_digestinit

5. sha1sum(1): man 1 sha1sum
http://linux.die.net/man/1/sha1sum
6. sha1(3): man 3 sha1
http://linux.die.net/man/3/sha1
7. Bourne Shell Programming, Steve Parker
http://steve-parker.org/sh/sh.shtml
8. Bourne Shell Programming, Andrew Arensburger
http://www.ooblick.com/text/sh
9. UBUNTU Hypertext Man Pages
http://manpages.ubuntu.com/
10. Linux Shell Scripting Tutorial: A Beginner's handbook
http://www.freeos.com/guides/lsst/
11. The Linux Documentation Project
http://www.tldp.org/
12. Advanced Bash-Scripting Guide
http://tldp.org/LDP/abs/html/
13. A collection of tutorials
http://www.grymoire.com/Unix/
14. The Bash Manual
http://www.gnu.org/software/bash/manual/bash.html