

# Assignment: System Integrity Verifier (SIV)

---

## 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 or report file exists already then, the user will be asked for each existing file if he wants to overwrite it. If answer is “no”, the program terminates.
- e) Recursively iterate through the directory contents (of arbitrary depth level) and for each file 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, including filename
  - II. Size of the file
  - III. Name of user owning the file
  - IV. Name of group owning the file
  - V. Access rights to the file (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, the user will be asked if he wants to overwrite it. If answer is “no”, the program terminates.
- 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
  - III. Files with a different message digest than computed before
  - IV. Files/directories with a different user/group
  - V. Files/directories with modified access right
  - VI. Files/directories with a different 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.

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. If the program fails because it does not support the requirement arguments, the entire submission will be marked as fail.

*Example 1: Initialization mode*

```
siv -i -D important_directory -V verificationDB -R my_report.txt -H sha1
```

*Example 2: Verification mode*

```
siv -v -D important_directory -V verificationDB -R my_report2.txt
```

## 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++, Objective-C, Java, Perl, or Python. You are not allowed to use libraries of functions that are not included in the standard installation of the programming language. Using an already existing SIV system (such as Tripwire) is not allowed either. Your submission will be tested on serverA – make sure it works there!

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.

## Testing

The best way to 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.

## Deliverables and evaluation

This is an individual assignment with two deliverables. 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. Make sure your program can execute on serverA without extra additions. 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.

Authors who submit very similar code will be asked to explain in detail their programs in order to eliminate plagiarism suspicions. It is expected that authors can explain every line in the submitted program as well as the format of the verification file.

The report will be written with Microsoft Word (or compatible package) and formatted according to the ACM small standard format available from <https://www.acm.org/publications/acm-word-style-guide> (first template on that page). The report will be uploaded to Canvas in PDF format (**not Word**).

The report must have the following structure:

1. Introduction.  
Here you explain the goal of the assignment 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. 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 to 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 specify them, so I can prepare a similar environment when testing your program.
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 for warranting a passing grade.

## 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](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>