HAVUGIMANA
Léonce
Pers.nber: 860512_5494

2017-07-14

Assignment1: System Integrity
Verifier Report

1.Introduction

This is a report of assignment1, System Integrity Verifier, in the course Network Security(ET2540).The main goal of this work is to design and implement a simple application to detect changes which may occur in a supervised file system(or specific directory) and to understand concept of file integrity. The intention was to touch on programming language libraries designed to implement file manipulation and encryption algorithms such as SHA1, MD5,etc.. Moreover, it is important to have some practice about designing and developing integrity verification system at basic level, this gives some insights of File System architecture and security, which is the basic of security of data to be transmitted over a network.

2.Design and implementation:

This system Integrity Verifier(SIV) is built on capabilities provided by java API. The main class encompasses a bunch of methods which help to perform different tasks. The source code can be compiled by java compiler, Javac. The system has been designed according to the assignment requirements mainly the two modes of running the application: Initialization mode and verification mode. Basing on that, the problem is divided into two but interdependent blocks. Verification mode is dependent on initialization components. Command line arguments are defined using array. We didn't find any other java native API to help for command line options specification. All elegant class libraries available for this function are external(eg. Apache Commons CLI,...), they should be imported as third party API; and assignment instructions do not allow to use libraries which are not included in the standard installation of programming language.

The system conception intends to detect the following changes which may occur to a file in a given directory:

1. New or removed files/directory
2. Files with a different size than recorded
3. Files with a different message digest than computed before
4. Files with a different user group
5. Files with modified access right
6. Files with a different modification date

Before explaining changes detection, let's talk a bit about initialization. The most useful part of initialization, is a recursive method which iterates in monitored directory of any length, retrieving files stored in subdirectories, if any; and all files are stored in a list. Then, for every single file there is a call of java "Path" class static methods. Path class provides methods helping to identify file attributes(size,owner,etc…). Those attributes are recorded in a verification file. Verification file records will later play the role of baseline from which we can figure out if a file has undergone

any modification. File attributes are recorded in verification file(veriFile) as follows to facilitate retrieval:

-each line contains individual file attributes ending with the name of Hash Function

-each string in a line is a value of an attribute

Eg./home/leonce/Courses/Untitled1 998 leonce leonce rw-rw-r-- 2017-07-07T01:19:13.190915Z 4514b731b9891f10a409ce7cffc2b227 MD5

The order is: filename_size(in bytes)_owner_Group owner_access right_last modification date_Hashfunction

Algorithm statement:

- Retrieve all files in a monitored directory and store them in ArrayList<File>
- For each file : cast it toString, then add it to a Set collection (monitorSet)
- Read all the file names written in Verification File & store in another set(veriSet)
- Declare a 3rd set : newFilesSet=monitorSet\veriSet
- Use looping to print every element x of the set difference into report File **(New files identified)** **,**numberOfWarnings++
- Declare a 4th set: removedFilesSet= veriSet\monitorSet
- Use loop to print every element x of the set difference into report File**(Removed files identified)** **,**numberOfWarnings++
- Declare a list of string and store in it the first line record strings of the veriFile (this will help to recover Hash Function ,in last index of the list)
- For each file(string) in monitorSet do the following:

- Create Paths object(Paths.get(File file)) >find Size of the file> scan veriFile set of content (veriSet) searching size. If(!veriSet.contains(size))> Print name and size in report(**different size than recorded**)**,**numberOfWarnings++

-GetHash(file) .if (! VeriSet.contains(HashValue)) >print name and hashValue in report**(different message digest than recorded)** **,**numberOfWarnings++

-GetUser(file). If (!veriSet.contains(user))> print name and username in report**(different user than recorded before)** **,**numberOfWarnings++

-GetGroupOwner(file). If (!veriSet.contains(GroupOwner))> print name and groupOwner(**different group than recorded before**) **,**numberOfWarnings++

-GetAccessRight(file). If (!veriSet.contains(accessRight))> print name and Access Right**(different Access right than recorded before),**numberOfWarnings++

GetLastModificationTime(file). If (!veriSet.contains(LastModificationTime))> print name and LastModificationTime **(different modification than recorded),**numberOfWarnings++

3.**Usage**

The system is designed to run through command line Interface. Development and testing was done in ubuntu Linux. So, it platform-dependent. Command options and attributes values are passed following description below. Failure to do so will lead to exceptions which may result into system crashing.

Initialization mode:

Initialization mode is when the system collects file attributes for all files/sub-directories under a given directory(monitored directory). Collected attributes values are printed in a file(verification file). A very short summary of initialization process is also printed in another file(report file). Message digest of the content of every file is computed by hash function (SHA1 or MD5) passed in command line. All the command line arguments names are passed by the user and it is advised to provide absolute path names of files unless they are stored in the current directory. It is not necessary to create a file before passing it in command line, the important thing is to give the name you want to give the file and it will be created before being written. Verification file is set to readOnly mode while being created.

For convenience reasons while reading verification file written during initialization mode, it is highly recommended to do not include space in names of files which are in monitored directory; example: " document 1" instead use"document1"(no space between "document" and "1").

Pass command line arguments as:

Java>Siv>-i>-D>monitored_Dir.>-V>verification_file>-R>report_file>-H hashfunction

>: denotes space

Verification mode:

In this mode, the system checks if any change occurred in monitored directory after initialization mode. That is why we need to feed the system with the file containing information recorded during initialization mode, then the system will compute new file attributes and compare them with the records in verification file. Also in this mode the user provides report file name, then the actual file object will be created at runtime.

Pass command line arguments as:

Java>Siv>-v>-D>monitored_dir.>-V>verification_file>-R>report_file

>: denotes space

Help:

Help mode prints out a brief information about ways of running the system.

4.Limitations

It is not possible to identify files names containing spaces while reading verification file content.

All possible exceptions are not declared which may cause unexpected termination of some processes.