SIV Report

ET2540 Network Security

Rong Peng(199511207004)
Blekinge Institute of Technology

I. Introduction

SIV (System Integrity Verifier), which can detect file system modifications occurring within a directory tree. The SIV can be run either in Initialization mode or in Verification mode. In Initialization mode, the SIV system will generate initial information of the current monitored directory, which will be saved in the verification and report files. The report file is a text file, and the verification file is a JSON file. In the Verification mode, the SIV program will verify the current monitored directory information with the initial one, when it detects the modification occurring, it will output the statistics and warnings about changes, and the warnings will be also recorded in report file, where can be easily read and check. This project can be possible applied in the version control system, like GitHub, which can detect repositories modification. At the same time, it can also be used in checking some significant folders or files modification, and captures into system log.

II. Design and Implement

A. Program Design

SIV system contains two modes, the initialization mode and the verification mode. In the initialization mode, program will do the required validations then record the initial information of Monitored Directory into report file and verification file. In the verification mode, SIV will do the required validations and detect any modifications based on the verification file.

Coding Language: Python 2.7

Environment: Linux

The following steps are the overview of the program logic:

Step1. Design logic to parse the command line arguments

Step2. Design logic to realize the functions of validations and detection

Step3. Design condition logic to implement two modes

The program is designed in a functional structure, the following figure is the list of the functions:

No	Functions
F1	DirExistsVerify(dirpath)
F2	FilesExistVerify(filepath)
F3	FileOverwriteAsk(filepath)
F4	OutsideMonitorDir(filepath)
F5	HashDigest(hashfunc,vpath)
F6	PathAnalysisWriteFile(monitored_path)
F7	DivergesAnalysisWriteFile()

Fig1. SIV Functions List

[1] Parse the command line arguments

Python *argparse* module helps analyze the command line arguments The pseudo_code interprets the logic to parse input commands:

```
import argparse
parser = argparse.ArgumentParser("<description>")
parser.add_argument("<commands>",<description/params>)
args=parser.parse_args()
```

[2] Functions Structure

F1: Detect the specified monitored directory exists or not

```
If not os.path.isdir(Monitored_Directory):
    #Remind user the path not exists, whether to create?
    Flag= raw_input("description")
    if Flag =='Y' or 'y':
      # create the monitored folder
      os.makedirs(Monitored_Directory)
    elif Flag =='N' or 'n':
      # exit the program
      exit(0)
    else:
      #Invalid input
      F1()
Else:
    Print 'The specified monitored directory exists.'
```

F2: **Verify the specified report & verification files exist**

```
If not os.path.isfile(filepath):
    codecs.open(os.path.abspath(filepath),'w','utf-8')
```

SIV system must have report and verification two files to record information, so if the specified file not exist, system will help create automatically.

F3: Ask the user whether overwrite the files

```
While True:
    prompt = raw_input("Notion")
    if prompt == 'Y' or prompt == 'y':
        codecs.open(os.path.abspath(filepath), 'w', 'utf-8')
        print "Done the Overwrite!"
```

```
            break
        elif prompt == 'N' or 'n':
            print "No Overwrite"
            break
        else:
            print "Invalid  input"
            continue
    return prompt
```

F4: Verify the specified file outside the specified Monitored directory

If os.path.commonprefix([os.path.abspath(MonitoredDir), os.path.abspath(filepath)]) != os.path.abspath(MonitoredDir):

```
    Mark="True"
    return Mark
```

os.path.commonprefix module will return the longest common path of the comparative paths.

F5: Generate the Hash message digest

```
If  hashfunc == 'sha1'or 'SHA1':
    sha1 = hashlib.sha1()
    sha1.update(vpath)
    digest = sha1.hexdigest()
Elif hashfunc =='md5' or 'MD5':
    md5 = hashlib.md5()
    md5.update(vpath)
    digest = md5.hexdigest()
Else:
    digest='  '
return digest
```

If the specified Hash function is not support by SIV, it will leave blank for the hash message digest in the verification file.

F6: Analyze the initial information of the Monitored Directory and write them to the corresponding files

For loop walk through the monitored directory:

```
    dir_size = os.path.getsize(dir_path)
    dir_date = datetime.date.fromtimestamp(
    os.path.getmtime(dir_path)).isoformat()
    dir_user = os.stat(dir_path).st_uid
    dir_grp = os.stat(dir_path).st_gid
    dir_right = oct(os.stat(dir_path).st_mode & 0o777)
```

```
with open(VerificationFilePath, 'w') as ver_file:
    json.dump(……)
with open(ReportFilePath, 'w') as rep_file:
    rep_file.write(……)
```

SIV get the most of files/folders properties from the python os.stat module, and record them in JSON file, some of the information writes in the report file, in the text format.

F7. Detect the 6 types change

For loop walk through the monitored directory, iteractively record the current monitored directory information, then read the verification file to compare if contain diverges:

-New or removed files or directories

Find whether the files/directories paths is in the current system, then can know whether it have been removed or created a new one. *temporary_json_record*

```
for record in verification_json_file:
    if not os.path.exists(record):
        warning issues count++
        x= Warning the file/folder has been removed
        f= open(report_file_path,'a')
        print >>f,x
        f.close()
        print x
for path in currentMonD_temp_dictionary:
    if path not in the verification file:
        warning issues count++
        x= Warning the file/folder has been added
        f= open(report_file_path,'a')
        print >>f,x
        f.close()
        print x
```

-File with a different size than recorded

Size can get from os.path.getsize(d_path) function.

```
If current_dir_size != initial_dir_size:
    warning issues count++
    x= Warning message
    f= open(report_file_path,'a')
    print >>f,x
    f.close()
    print x
```

-Files with a different message digest than computed before

If *current_file_hash* ***!=*** *initial_file_hasht:*

warning issues count++

x= Warning message and the hash_digest

f= open(report_file_path,'a')

print >>f,x

f.close()

print x

-Files/directories with a different user/group

User/group id can get from *os.stat(d_path).st_uid/st_gid* can functions, they are in similar logic:

If *current_dir_st_uid* ***!=*** *initial_dir_st_uid* ***or***

If *current_dir_st_gid* ***!=*** *initial_dir_st_gid:*

warning issues count++

x= Warning message

f= open(report_file_path,'a')

print >>f,x

f.close()

print x

-Files/directories with modified access right

Access right can get from *oct(os.stat(d_path).st_mode & 0o777)* function.

If *current_dir_right* ***!=*** *initial_dir_right:*

warning issues count++

x= Warning message

f= open(report_file_path,'a')

print >>f,x

f.close()

print x

-Files/directories with a different modification date

Date can get from *datetime.date.fromtimestamp (os.path.getmtime(d_path)).isoformat()* function.

If *current_dir_modified_time* ***!=*** *initial_dir_righ_modified_timet:*

warning issues count++

x= Warning message

f= open(report_file_path,'a')

print >>f,x

f.close()

print x

[3] Conditional Logic for two modes

If mode = **initialization mode**:

1. Begin to calculate the start time
2. Call F1 to ensure Monitored directory exists or not
3. Call F2 to ensure report and verification files exists
4. Logic to verify the specified Hash function supported by the SIV or not
5. Call F4 to verify both files are outside the specified Monitored Directory
6. Call F3 to ask user whether overwrite the report and verification files respectively,
7. If overwrite, the specified file will be an empty file; no overwrite, it will call F5 to get the information then write to the file. At the same time, calculate the end time of initialization mode.

If mode = **verification mode**:

1. Begin to calculate the start time
2. Verify three of specified files path should be existing
3. Call F4 to ensure report and verification files are outside the specified Monitored Directory
4. Verify the specified verification file and report file cannot be an empty(0 size) file.
5. If through the above validation, verify the specified verification file path is same with the path recorded in verification JSON file. Uses *linecache.getline()* to extract the "full pathname to verification file" in report file to compare.
6. If equal to that pathname, call F3 to ask user whether overwrite the specified report file or not.
7. If overwrite, the report file will be an empty file; no overwrite, it will call F6 to detect the diverges then generate warnings and write summary to the report file. During this mode, verification file is not allowed to modify. At the same time, calculate the end time of verification mode.

B. Circumstances that will terminate the program

The following circumstances happen will terminate SIV:

1. Both modes, report file or verification files are inside the specified monitored directory;
2. In the verification mode, any of specified path(report file/verification file/monitored directory) not exists/has been deleted;
3. In the verification mode, if the verification file or report file is an empty/have been overwritten;
4. In the verification mode, when the specified verification file path is different from the path which recorded in initialization mode. To prevent moving/distorting the verification file, SIV should do the validation for the sake of security;

5. In the initialization mode, when the monitored directory not exists and user reject to create a new one;
6. User has given answer when ask whether overwrite or not, then program done the flow.

C. Verification file format

Verification file records information in JSON Key-Value format. The following is the verification.json file format:

```
{
File _path :{
"file_right": 4 digits permission mask,
"file_hash":hash code,
"file_user":4 digits ID,
"file_group":4 digits ID,
"file_size": size_num,
"file_path": url,
"file_Last_Modified_Date":datetime
},
Directory_path:{
"dir_Last_Modified_Date": datetime,
"directory_path": url,
"directory_size": size_num,
"directory_group":4 digits ID,
"directory_user": 4 digits ID,
"directory_right": 4 digits permission mask
}
}
```

D. Report file format

```
----------Initialization _mode----------
Full pathname to monitored directory: xxxxxxxxxx
Full pathname to verification file:xxxxxxxxxx
Number of directories parsed: xx
Number of files parsed: xx
Time to complete the initialization mode: xxxxx
------------------------------------------------------------
Warning: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

----------Verification _mode----------
Full pathname to monitored directory: xxxxxxxxxx
Full pathname to verification file:xxxxxxxxxx
Full pathname to report file:xxxxxxxxxx
Number of directories parsed: xx
Number of files parsed: xx
Number of warning issued:xx
Time to complete the verification mode: xxxxx
------------------------------------------------------------
```

III. USAGE

Premise Command explanation

python: the system coded in python language;
-i : initialization mode
-v: verification mode;
-D: monitored directory;
-V: verification file;
-R: report file;
-H: hash function;
sha1/SHA1: the default hash function, you can also specify "md5/MD5".

The both modes can specify the full path or relative path, that's up to user. But which should be reminded that all the paths cannot be **Non-existent path,** SIV system will **not** help create folder except for the specified monitored folder. SIV will **not** help generate files except for the report file and verification file. Therefore, if the specified path contains non-existent folder path, system will fail. Please follow the below command format to run SIV friendly.

A. Initialization Mode Command

python *siv.py* **–i -D** <*Monitored_Dir_Path*> **-V** <*Verification_Path*> **–R** <*Path_Report**.txt***> **-H [sha1|md5]**

Use Examples:
python /home/rong/SIV/siv.py **–i –D** /home/rong/SIV/monitored_dir **–V** /home/rong/SIV/verification.json **–R** /home/rong/SIV/report.txt **–H SHA1**

B. Verification Mode Command

python *siv.py* **–v -D** <*Monitored_Dir_Path*> **-V** <*Verification_Path*> **-R** <*Path_Report**.txt***> **-H [sha1|md5]**

Use Examples:
python /home/rong/SIV/siv.py **–i –D** /home/rong/SIV/monitored_dir **–V** /home/rong/SIV/verification.json **–R** /home/rong/SIV/report.txt**–H MD5**

C. Regard new adding files/folders in verification mode

The diverges detection always based on the initial status of the monitored directory. Therefore, any changes made in the new adding files/folders in verification mode, the warning just only prompts the file/folder has been added, unless user rerun the initialization mode and then modify the file/folder.

D. Ownership &Permission of siv.py

After downloading the SIV code, please check the ownership and permission. Using `sudo chown –R <your_role>` and `sudo chmod` to change the ownership and access permission to make the SIV run in your server.

IV. LIMITATION

The clumsy system is not so intelligent that it contains several limitations as below.

1. The naming of the files/folders under the monitored directory cannot be the special letters/Chinese words/Japanese words/Swedish words. The verification mode will fail to compare such files/folders. Recommends to name them only in English.
2. Cannot put the pictures, PDF, videos into the monitored directory, it cannot record.
3. If the user manually opens and modifies the report or verification file generated in the initialization mode before running verification mode, the SIV system will recognize it as a circumstance that the user does the initialization mode manually. Although it is not good behavior, the system is still allowing to do that.