NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

File System Monitoring with use of Audit Framework

Ioannis C. Katsanis

Supervisors: **Mema Roussopoulos,** Associate Professor

ATHENS

OCTOBER 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύστημα παρακολούθησης αρχείων με τη χρήση του Audit Framework

Ιωάννης Χ. Κατσάνης

Επιβλέποντες: **Μέμα Ρουσσοπούλου,** Αναπληρώτρια Καθηγήτρια

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2019

Bsc THESIS

File System Monitoring with use of Audit Framework

Ioannis C. Katsanis
A.M.: 1115201500066

Supervisors: **Mema Roussopoulos,** Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύστημα παρακολούθησης αρχείων με τη χρήση του Audit Framework

Ιωάννης Χ. Κατσάνης
Α.Μ.: 1115201500066

Επιβλέποντες: **Μέμα Ρουσσοπούλου,** Αναπληρώτρια Καθηγήτρια

ΠΕΡΙΛΗΨΗ

Την σήμερον ημέρα, υπάρχει η ανάγκη να παρακολουθούμε τις κινήσεις που γίνονται σε ένα σύστημα αρχείων, ώστε να ρυθμίσουμε την συμπεριφορά του και να εκμεταλλευτούμε την γνώση για τον τρόπο που χρησιμοποιείται από χρήστες. Εκτός άλλον, και για αυτό τον σκοπό έχουν αναπτυχθεί αρκετά εργαλεία που προσφέρουν όσο το δυνατό πιο λεπτομερή καταγραφή. Στο λειτουργικό σύστημα Linux υπάρχει το εργαλείο Audit Framework που προσφέρει τέτοιες υπηρεσίες.

Το Audit Framework είναι ένα προσαρμώσιμο εργαλείο που μπορεί με διάφορες λύσεις, να παρέχει υπηρεσίες παρακολούθησης σε επίπεδο πυρήνα συστήματος, χωρίς να υπάρχει μείωση της απόδοσης του. Βέβαια χρειάζεται να αναθεωρήσουμε τις βασικές του ρυθμίσεις, αν θέλουμε καλή απόδοση, αφού το εργαλείο by default έχει αρκετά συντηρητική συμπεριφορά.

Έτσι, προσπαθήσαμε να παράξουμε ένα σύστημα που να αλληλεπιδρά με το Framework και να εξάγει χρήσιμη γνώση για την συμπεριφορά των χρηστών στους υπολογιστές τους. Παρουσιάζεται λοιπόν, το File System Monitoring.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Παρακολούθηση Συστήματος Αρχείων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: παρακολούθηση, σύστημα αρχείων, audit framework, Linux, υπηρεσία λειτουργικού συστήματος

ABSTRACT

Nowadays, there is the need to monitor the operations made in a file system, in order to customize its behavior and take advantage of the information extracted from the way users operate. For this case, there have been developed a lot of tools that offer, as much as possible, detailed monitoring. For the Linux operating system, there is Audit Framework that offers that kind of services.

Audit Framework is a customizable tool, that provides monitoring services, at the operating system's core level, without decreasing the system's performance. There is the need though, to reconsider the default settings, if we need optimal performance, since it has preservative behavior.

Thus, we tried to develop a system that interacts with the Framework and produces useful information about the computer user's behavior. So, we present the File System Monitoring.

SUBJECT AREA: File System Monitoring

KEYWORDS: π.χ. monitoring, file system, audit framework, linux, operating system service

For my family.

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα ήθελα να ευχαριστήσω τους επιβλέποντες, αν. Καθ. Μέμα Ρουσσοπούλου, Νίκο Χονδρό, Μιχάλη Κωνσταντόπουλο, για τη συνεργασία και την πολύτιμη συμβολή του στην ολοκλήρωση της.

CONTENTS

LIST OF PICTURES

LIST OF TABLES

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία διενεργήθηκε κυρίως μέσω emails και τηλεφωνικών κλήσεων, αλλά και μέσω συναντήσεων, όπου έτσι μπορούσα να λάβω πολύτιμη πληροφορία από την κα. Ρουσσοπούλου, τον Νίκο και τον Μιχάλη. Ευχαριστώ αυτούς τους καταπληκτικούς συνεργάτες μου σε αυτή την εργασία για την βοήθεια που μου έδωσαν, αλλά και για τον χρόνο που αφιέρωσαν και προσάρμωσαν στις δικές μου ανάγκες.

PREFACE

This project was done mainly via emails and voice calls, but also with meetings, where i could receive very important information from Mrs. Roussopoulos, Nick and Michalis. I want to thank those amazing partners in this project for their support, but also for their time, that they dedicated to me.

2. Audit Framework

Before we present Audit Framework, we have to answer some questions, like:

Why do we want monitoring?

By monitoring users' behavior with their file system, we can understand where to focus, to provide better services and make easier and faster his interaction with the machine.

or

Why do we use audit framework?

Audit framework offers a lot of mechanisms that provide helpful information about user interaction with the file system. With this tool, we can monitor every move made in the file system and have an accurate picture of the user's behavior.

Description:

The Linux audit framework provides a CAPP-compliant (Controlled Access Protection Profile) auditing system that reliably collects information about any security-relevant (or non-security-relevant) event on a system. It can help you track actions performed on a system.

2.1. What is it made of

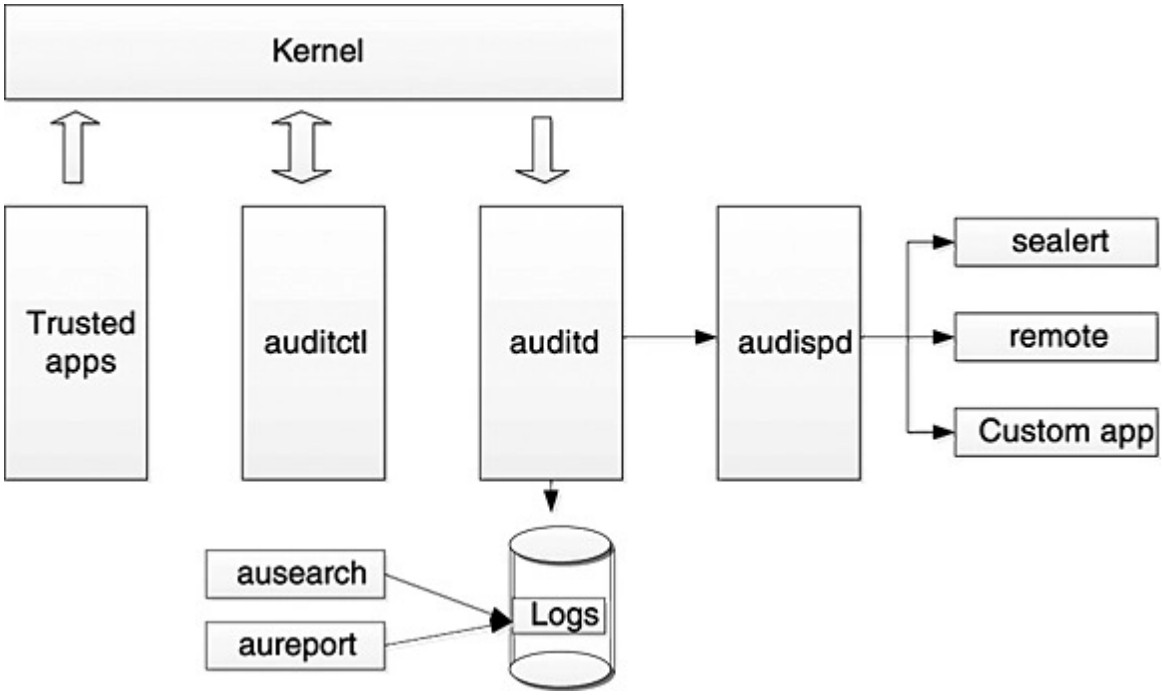
Audit framework is composed of the auditd daemon, responsible for writing the audit messages that were generated through the audit kernel interface and triggered by application and system activity.

This daemon can be controlled by several commands and files:

- *auditctl* : to control the behavior of the daemon on the fly, adding rules etc.
- /etc/audit/audit.rules : contains the rules and various parameters of the auditd daemon
- *aureport* : generate report of the activity on a system
- *ausearch* : search for various events
- *auditspd* : the daemon which can be used to relay event notifications to other applications instead of writing them to disk in the audit log
- *autrace* : this command can be used to trace a process, in a similar way as strace.
- /etc/audit/auditd.conf: configuration file related to the logging.

2.2. How does it work

This is an overview of audit and its environment:



Picture 1 : Audit Framework architecture

In a few words, audit lives inside the kernel of the file system. In that way, it has access to everything that needs to execute kernel code (basically system calls). Mainly, auditd does this job and produces the logs, that are mentioned in the whole document. Thus, as long as audit daemon (auditd) runs, every system call can be monitored and registered.

2.3. Installation

Installing the tool is pretty simple, as long as you have a linux distribution.

Debian/Ubuntu : apt-get install auditd audispd-plugins

Red Hat/CentOS/Fedora : usually already installed (package: audit and audit-libs)

2.4. Configuration

We run the command below every time we boot the system (In the project's directory):

~# python3 main.py

(This script needs root permissions to run (#), as accessing audit log files is prohibited for non-root users.)

There is need for **(2)** confiruration files to change in order to configure the daemon properly.

1) /etc/audit/auditd.conf : This file contains configuration information specific to the audit daemon.

2) /etc/audit/audit.rules : This file contains our rules used to monitor the system calls.

These changes are done **by the script** every time the system boots, so there is no need for manual changes. Also, as most systems do, these configuration choices can be dynamically loaded by the run time, so configuration file may seem unchanged.

How to make program run at boot time (Ubuntu with GUI)

1.Open the Dash and search for *"Startup Applications".*

2.Now click on *Add* and give in the command to run the application.

(sudo python3 /<path_to_folder>/file_system_monitoring_src/main.py)

Configured variables worth mentioning:

backlog buffer: This is the number of maximum audit messages that can be queued before written on disk. Here is set to 30.000 messages. Every message is of size ~9KB, thus the required memory space is 270MB when the queue is at maximum size.

backlog_wait_time: This is the time that kernel waits for the queue above to be drained. In new kernels there is no need to wait, so we set it to 0. This gives best auditing performance as the kernel doesn't lose any records and most importantly user doesn't suffer from "freezed" system.

3. Alternative Tools

Except of Audit Framework, we had some further choices, but all of them were not exactly what we wanted. A brief description of them:

3.1. Inotify

What is it:

The **inotify** API provides a mechanism for monitoring filesystem events. Inotify can be used to monitor individual files, or to monitor directories. When a directory is monitored, inotify will return events for the directory itself, and for files inside the directory.

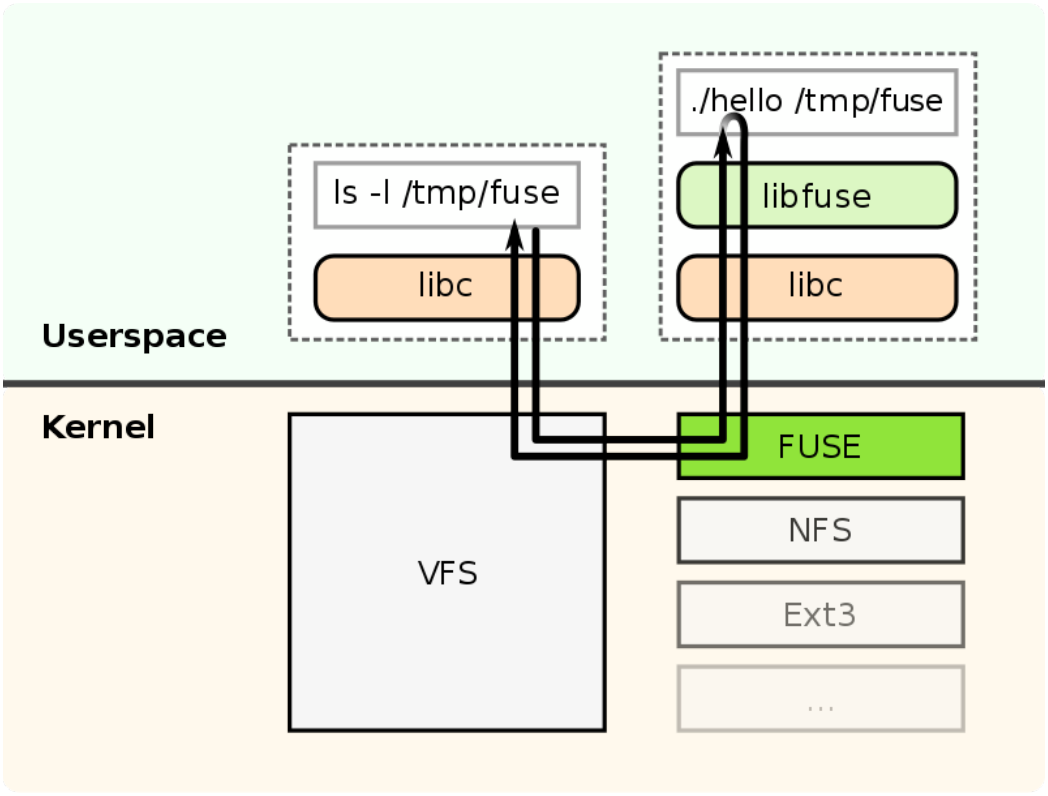
Drawbacks:

This API doesn’t provide enough information for this project to reach its target. There is need for more detailed data associated with every file system’s action (open, read, write etc) such as, in case of a write system call, bytes written, position started writing.

More specifically, Inotify is designed to monitor events that happen in files and directories and not system calls in general. That means that the extracted information is file-oriented and not system call-oriented. So, for example, when it comes to a read system call, we can’t get all the needed information (like the number of bytes that where read), or in a fork system call we cannot register it at all.

3.2. FileSystem in User Space (FUSE)

What is it:



Picture 2 : FUSE command execution flow

Filesystem in Userspace (**FUSE**) is a software interface for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.

Drawbacks:

This API can provide strong monitoring tools, as it can monitor all I/O requests, get detaied data for read/write calls, but is a little difficult/uncomfortable for every user to mount a new file system.So, for user simplicity it is not used.

3.3. Strace

What is it:

strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. The operation of strace is made possible by the kernel feature known as ptrace.
Some Unix-like systems provide other diagnostic tools similar to strace, such as truss.

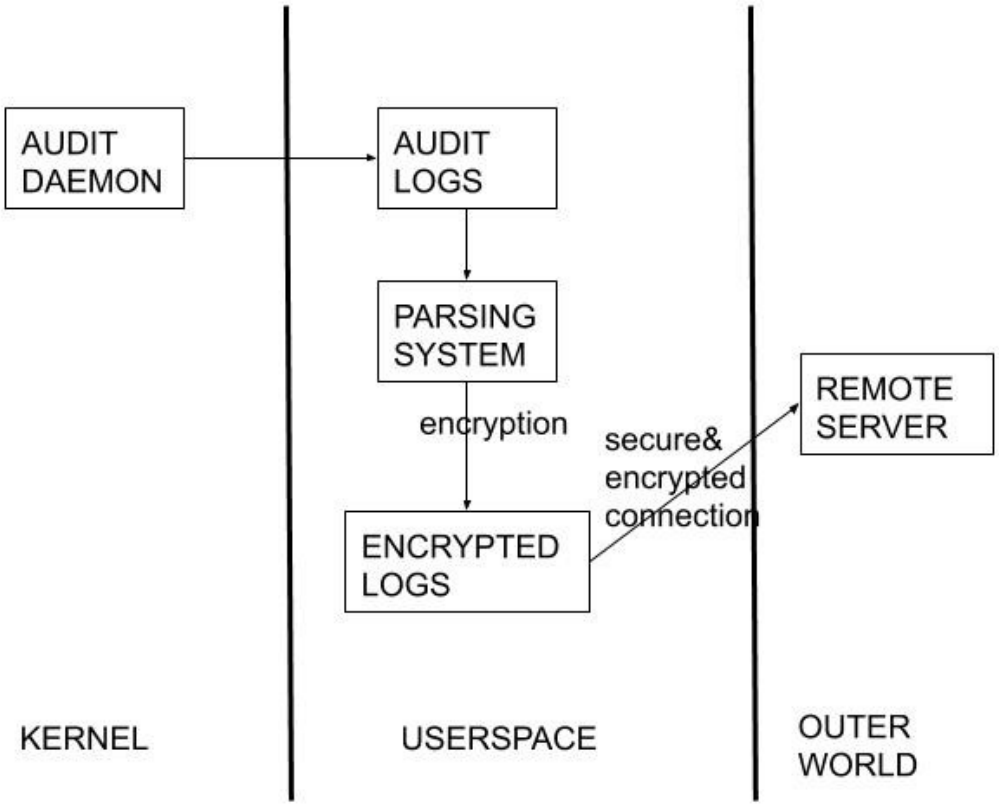
Drawbacks:

The most common use is to start a program using strace, which prints a list of system calls made by the program. This use of the tool is not applicable in our case, because we don't access the monitored data at real time.

The need in this project is to get informative logs for operations made in the file system and analyze them in a second phase. Strace works with real-time system logs, so it is not applicable in our case.

4. How is auditing achieved

The overview of the whole project is the below:



Picture 3 : Auditing project architecture overview

In order to achieve the overview goal, we created the above strategy. In simple words, we use the output of the framework, parse it, encrypt it and store it for statistics.

The details for the implementation of every of the above sub systems are analyzed in the next paragraphs.

4.1. Audit Framework in action

Now, let's describe what was done in order to configure properly audit daemon and get good performance at run time.

Introduction:

Audit Framework accepts rules to format what files, directories and system calls will monitor. With this feature, we can keep only the useful data about the file system and ignore the unnecessary ones. Also, it has a major impact in performance optimizations, since we don't flood the system with information.

The rules that were added at **audit.rules** file, to reach our goal, have to do with:

1) Decreasing the size of every record that the framework produces.

2) Removing the system's folder from the monitored directories.

3) Adding only the system calls in which we are interested to monitor.

4) Filtering data by the current user and ignore other users (e.g. root).

The system calls that are monitored, with a small description of their usage, are the below

- **Open**: It opens a file.
- **Close**: It closes a file.
- **Read**: It reads a number of bytes from a file.
- **Write**: It writes a number of bytes in a file.
- **Dup**: It creates a new file descriptor for a file. Then both old and new file descriptors can be used interchangeably.

- **Dup2**: Same as **Dup** , but using a given new file descriptor.
- **Fork**: It creates a new process by duplicating the calling process. Both Parent and Child process have different address spaces.
- **Vfork**: Same as **Fork** , but the child process and the parent process use the same address space.
- **Clone**: Same as **Fork** , but it allows the child process to share parts of its execution context with the parent process, such as the virtual address space, the table of file descriptors, and the table of signal handlers.
- **Lseek**: It moves the file pointer in a specific, opened file.
- **Mkdir**: It creates a new directory.
- **Rmdir**: It removes a directory.

In general, the overall idea is to monitor only the current user home directory and mainly ignore any unnecessary, non-related-to-user data. Thus, let's suppose that the current user name is *usr*, the directory that will be monitored is the home directory of *usr*.

4.2. Audit Framework Logs Parsing

```
type=SYSCALL msg=audit(1556969892.851:666187): arch=c000003e syscall=1 success=yes exit=1 a0=11 a1=7f385f3d7a97 a2=1 a3=7f385daac960
items=0 ppid=5418 pid=8927 auid=4294967295 uid=1000 gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=(none)
ses=4294967295 comm="Timer" exe="/usr/lib/firefox/firefox" key=(null)
```

Picture 4 : A look of a record in the Audit Framewrok logs

Initially, by using Audit Framework, we record the previously mentioned system calls and directories. Periodically, we collect the records that the framework produced, with the help of a framework's subsystem (ausearch). Then, there is implemented a parsing system, which collects these records and registers any changes in the file system.

More specifically, we keep track of every file in the user's home directory, so for every record we identify the system call that run and any of the parameters that were given. These information are enough to update the parser's structures, which keep the state of the file system in a period.

The structures above are used only because the parsing is happening after the auditing is done. So, we try to simulate the state of the remote file system in deferred time.

4.3. Log Encryption

Note: Any file endings (.jpg, .pdf etc) are kept and not encrypted (for statistical purposes).

In order to make sure that the data (file and directory names) are remaining unknown outside local machine of presence, there must be used an encryption protocol. This means that the data that are produced by the parsing system are encrypted before exiting the user's machine and no one can recognize them.

It is of major importance though, to have a strong encryption function, to make sure that it is impossible to decrypt the data. The data are stored in a remote server and may be exposed to threats, so

The mechanism that is used to encrypt is randomness. There is a random mapping of characters to characters that is randomly generated and stored locally (at the user's computer). So, if we want to take the encrypted character of one, we load the encryption file and we encrypt the character as the mapped character. By this way, decryption is done only by the owner of the file and there are no conflicts of mapped characters.

Thus, in order to hack the encryption system, you have to gain access to the encryption file, which means that you need to hack the whole file system. This is very difficult to happen, but not impossible. An other way to hack it, would be to control the input of the parsing system (audit logs) and maybe make a file with all the valid file characters as its name and take the encrypted file. This method also needs access to the file system to happen.

Nothing is unbreakable, but with this encryption logic there is enough coverage to protect sensitive user data (file and directory names).

4.4. Log Transfer to Server and Statistics

What information is kept for each monitored system call (Numbers).

	FILE NAME	STARTING POSITION	NUMBER OF BYTES	PROCESS ID	FILE HANDLER	OLD FILE HANDLER	NEW FILE HANDLER	PARENT	CHILD
READ	1	2	3	4					
WRITE	1	2	3	4					
OPENDIR	1								
OPEN	1			2	3				
CLOSE	1								
LSEEK	3	4 (position after seek)		1	2				
DUP						1	2		
FORK								1	2
PREAD	1	2	3	4					
PWRITE	1	2	3	4					

The data transfer is secure and every time the system starts monitoring, a password is asked (file2019monitor), in order to send user's encrypted data, that were stored locally, to the server. This is done with rsync .

This password is a given password and NOT any of user's passwords.

After auditing, parsing of the audit logs and encryption of the parsed data is done, we have to send and store these interesting data to a remote server. The log transfer should also be safe and protected, so we use ssh with rsync.

As previously mentioned, all logs are anonymous and there is no knowledge of sensitive data. At this point, we understand that many statistics about users can be concluded.

A simple list of useful statistical observations would be (*with some sample results from my computer, in a day*):

- File system usage of opening a file (read or write):

  What is the cause of opening a file in the file system.

  Number of reads: 91966

  Number of writes: 12759

- Time of day that contributes the most in file system traffic in hour classes (for example, 11.00-13.00):

  The time of day in which the most files/directories are accessed/written/read.

  In time 0-2, 2-4, 4-6, 6-8, 8-10, 10-12, 22-24 there were 0 operations.

  In time 12-14, there were 4289478 operations.

  In time 14-16, there were 1790006 operations.

  In time 16-18, there were 1897903 operations.

  In time 18-20, there were 8524 operations.

  In time 20-22, there were 67941 operations.

- Size (in bytes) of read/write operations:

  The amount of bytes that are read/written in comparison to the size of the file.

Read:		*Write:*	
Size	*Bytes*	*Size*	*Bytes*
4	*202*	*4*	*55*
6	*2119*	*12*	*15*
11	*2264*	*13*	*27*
14	*24079*	*119*	*14*
17	*1*	*3117*	*54*

- Hot (used frequently) vs cold percentage of files:

  The percentage of files that are accessed more times than the declared "limit" (hot), or less times (cold).

  Hot Files: 12078

  Cold Files: 37154

  Hot Percentage: 24.533 %

  Cold Percentage: 75.467 %

- Average depth of file path:

  The average path to a file access.

  Average depth of file paths is 4.983

- Number of file accesses during a day period:

  The number that a file was accessed in a calendar day.

  On 2019-10-12 happened 7237186 file accesses.

  On 2019-10-10 happened 464172 file accesses.

- Percent of read only files. (for example, never edited, only read for a time period):

  The percentage of the files that were opened and only read within time period.

  Read only file percentage: 0.998

- Number of files replicated to cloud:

  The number of files that are opened in a cloud directory (Names are encrypted).

  Number of files replicated to cloud:

  *Cloud Name = *, number of files = 3081*

  *Cloud Name = **, number of files = 0*

  *Cloud Name = ***, number of files = 54*

- Average Number of open operations per directory:

  The number of open operations for all accessed directories.

  Average number of file open operations per directory: 12.67

- Number of directories per Cloud :

  Number of directories that are accessed in a cloud directory(Names are encrypted).

  *Cloud Name = *, number of directories = 1*

  *Cloud Name = **, number of directories = 0*

  *Cloud Name = ***, number of directories = 0*

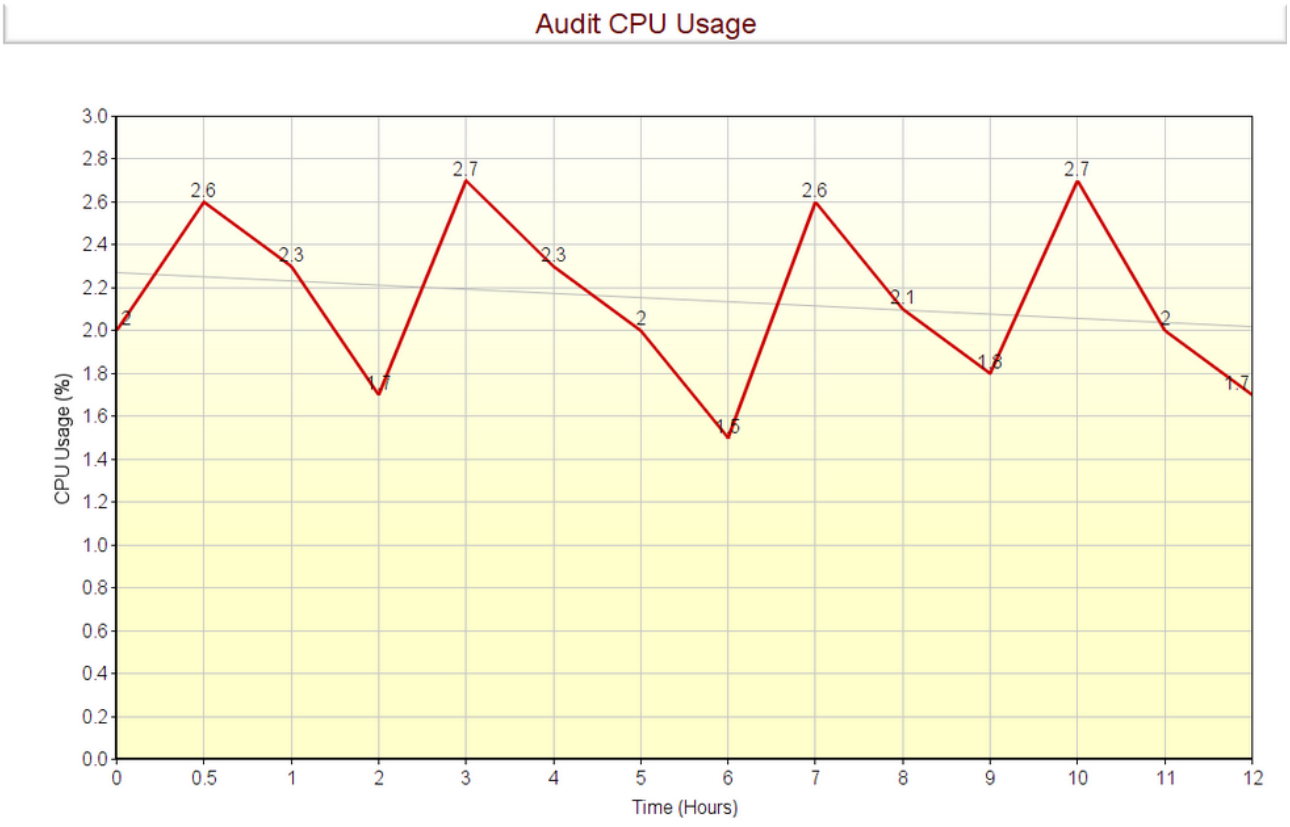
5. Conclusions

Some statistics of audit performance in my machine:

System Specs:

- Memory : 4GB
- CPU : Intel i5 – 7200U @ 2.5GHz
- OS Type : Ubuntu 16.04 LTS – 64 bit

Graphic Representation:



Picture 5: Project's CPU usage over Time

REFERENCES

[1] https://wiki.archlinux.org/index.php/Audit_framework – Audit Framework documentation

[2] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing – Audit Framework red hat portal

[3] http://man7.org/linux/man-pages/man7/inotify.7.html – inotify manual

[4] https://en.wikipedia.org/wiki/Filesystem_in_Userspace – FUSE manual

[5] https://github.com/libfuse/libfuse – FUSE source code

[6] https://en.wikipedia.org/wiki/Strace – strace manual

Πτυχιακή Εργασία

Ιωάννης Κατσάνης

2019

Ράχη τόμου