# AUDIT KEY POINTS,
# CONFIGURATION AND PERFORMANCE NOTES

## How to run it (Quick tips)

1. Go to project folder.
2. Open a terminal in this folder.
3. Run: `sudo python3 main.py`
4. 
   1. If it is NOT the first time you use the script, you will be asked for the password to upload local logs. (See further below what the password is and information about this procedure)
   2. Else no need to do anything else.

Audit Framework is a monitoring framework that records actions made by user or kernel. It can filter records by system call, user or file access.

For our needs, we use the framework to monitor system calls.

### 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.

### 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.

### Other alternatives

Except of audit, we had some further choices, but all of them were not exactly what we wanted. Some of them:

strace: Provides fully informative system call monitoring, but it is impractical in our case, cause we don't access the data right after a system call is executed. So, we won't always be able to track all file system moves.

Inotify: Almost what we wanted, but it is not informative enough. Can't track system calls' arguments

## Short summary of concept

Client Side:

When user runs the script, the audit daemon starts monitoring. Periodically, we use a script to parse the records produced by the daemon. Then, we produce a list of actions of our interest (e.g. file opened, bytes read, bytes written). This list gets encrypted by a random and consistent hash function and gets sent to the server via ssh. In the mean time, the audit daemon doesn't stop running.

The data transfer is secure and every time the system starts monitoring, a password is asked, 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.

Server Side:

The data gets collected and we can output some statistics.

## 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)

## Audit configuration

Software Requirements:

- Installed python 3

System Requirements:

- ~270MB of RAM space at worst case (this can be reduced if needed)

Installation:

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

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

Configuration:

We run the command below every time we boot the system:

```
~# 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.

Configured variables worth mentioning:

backlog: 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.

Some statistics of audit performance in my machine:

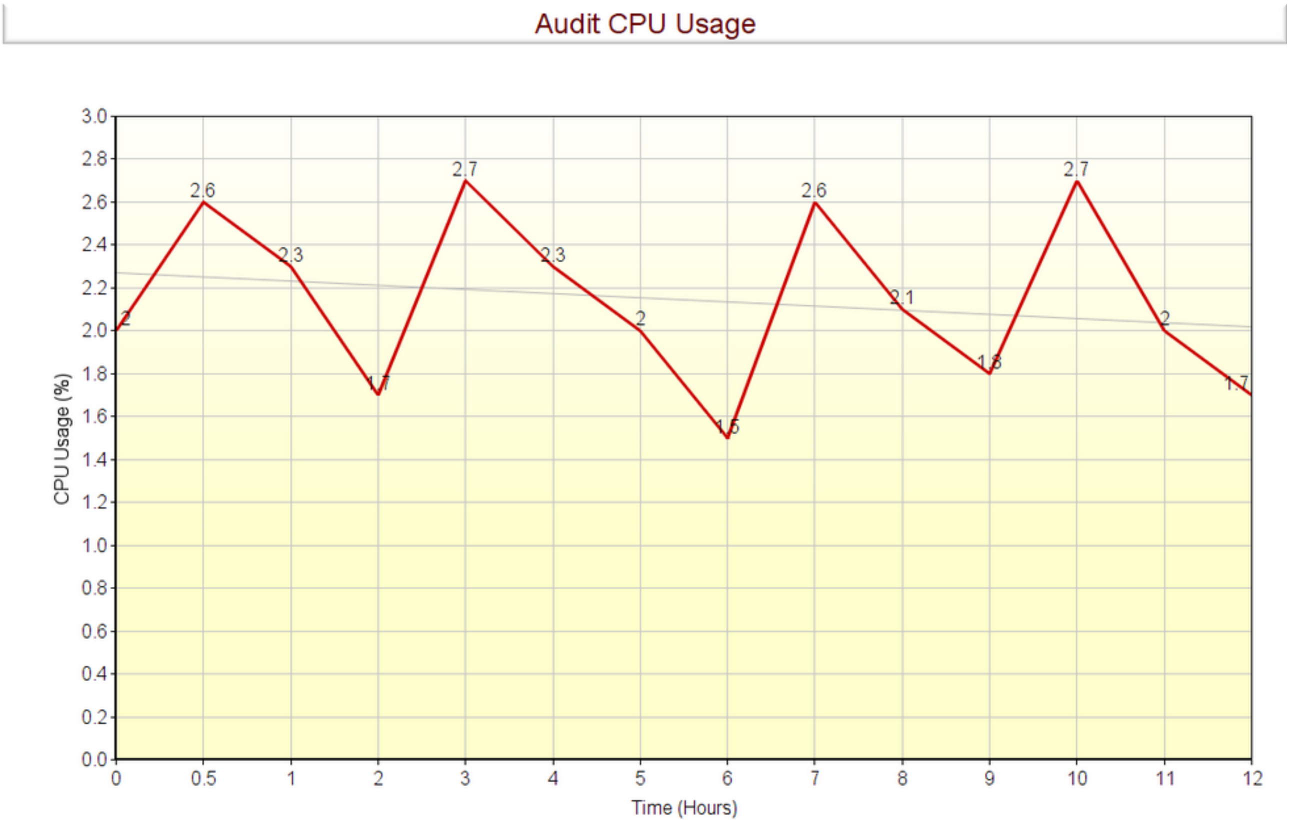
System Specs:

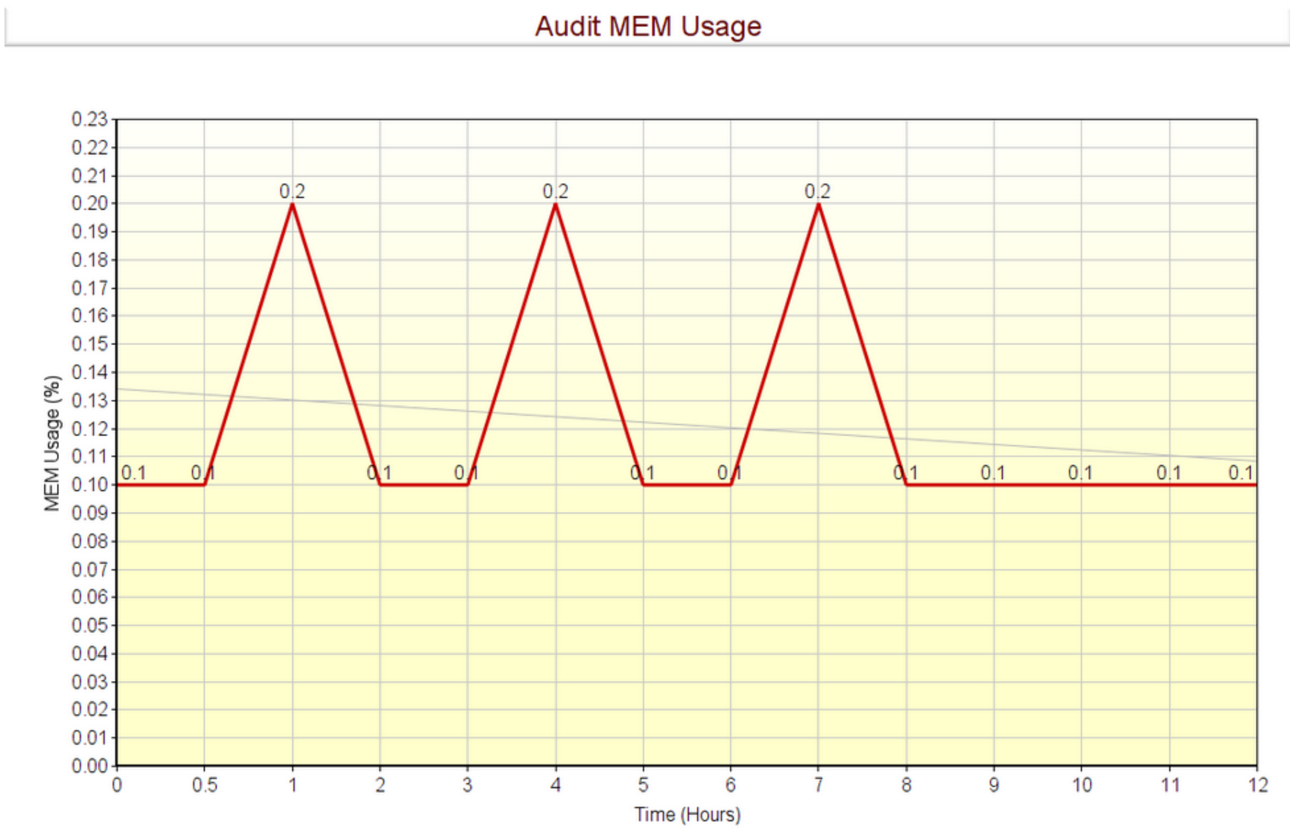
- Memory : 4GB
- CPU : Intel i5 – 7200U @ 2.5GHz
- OS Type : Ubuntu 16.04 LTS – 64 bit

| **TIME (hours)** | **Memory Usage (%)** | **CPU Usage (%)** |
|---|---|---|
| 0 | 2 | 0.1 |
| 0.5 | 2.6 | 0.1 |
| 1 | 2.3 | 0.2 |
| 2 | 1.7 | 0.1 |
| 3 | 2.7 | 0.1 |
| 4 | 2.3 | 0.2 |
| 5 | 2 | 0.1 |
| 6 | 1.5 | 0.1 |
| 7 | 2.6 | 0.2 |
| 8 | 2.1 | 0.1 |
| 9 | 1.8 | 0.1 |
| 10 | 2.7 | 0.1 |
| 11 | 2 | 0.1 |

| | | |
|---|---|---|
| 12 | 1.7 | 0.1 |

## Graphic Representation:





What columns mean for each monitored system call.

| | 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 | | | | | |
| OPEN DIR | 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 | | | | | |