Software Security

Course Project 51

Project Part #2 – Log Auditing

Team Members :

Vinay Kumar Gorle	1224671734	vgorle@asu.edu
Jyoti	1224121327	jnolas15@asu.edu
Janardhan Manthena	1224389556	mjanardh@asu.edu

Problem Statement:

The sequential records of data that are important and/or necessary to upholding the system's security are contained in an audit log. One can find precise information on the actions or changes that have affected a certain system operation, event, or process in these records. Typically, audit logs include specific information on the source address, destination, user login information, and a time stamp. They also maintain track of which sources were viewed.

The challenge handled in this project is to evaluate logs which are generated by logging tools (sysdig) and visually plot a graph which is easy to understand by an end user

This project is divided into 3 parts.

1. **Generating tuples :** Every log generated by sysdig is passed into a tuple which consists of subject, action and object. Here the subject is the process, object is the resource and the action performed by process on the resource.
2. **Generating graph :** The tuples were plotted in the form of a graph which can be easily visualized and understood. The direction of the edges depends upon the action performed by the process, if the action is anything related to write (send message, write) the edge

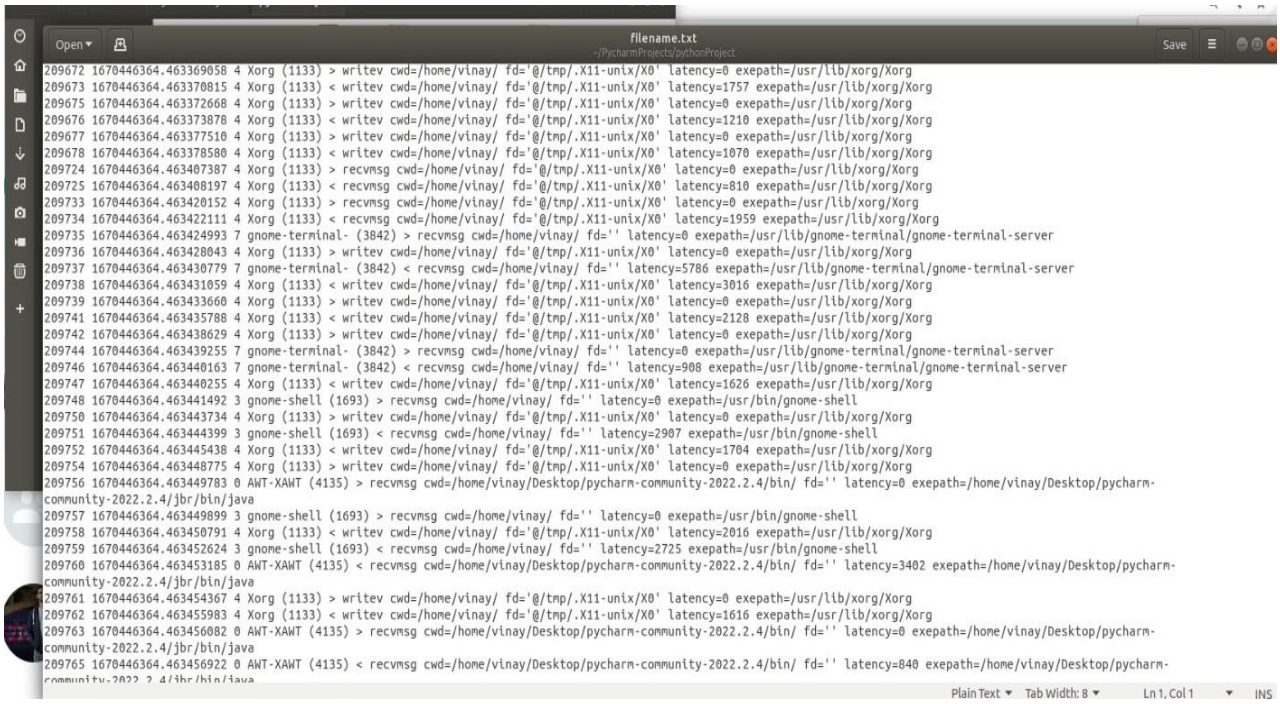
points the resource else the edge will point the process.

3. **Backtracking :** the above generated graph represents a means to visualize the events logged but the end graph is very big that it cant be easily understood. To tackle this issue we provided a means which can perform backtracking on a specific node (POI) and retain only those nodes and edges which are responsible for the point of interest (POI) node to occur.

Project implementation: As part of this project, the first task is to generate the event logs. To do that the following command is used.

sudo sysdig -p"%evt.num %evt.rawtime.s.%evt.rawtime.ns %evt.cpu %proc.name (%proc.pid) %evt.dir %evt.type cwd=%proc.cwd fd='%fd.name' latency=%evt.latency exepath=%proc.exepath " "proc.name!=tmux and (evt.type=read or evt.type=readv or evt.type=write or evt.type=writev or evt.type=fcntl or evt.type=accept or evt.type=execve or evt.type=clone or evt.type=pipe or evt.type=rename or evt.type=sendmsg or evt.type=recvmsg)" and proc.name!=sysdig > filename.txt

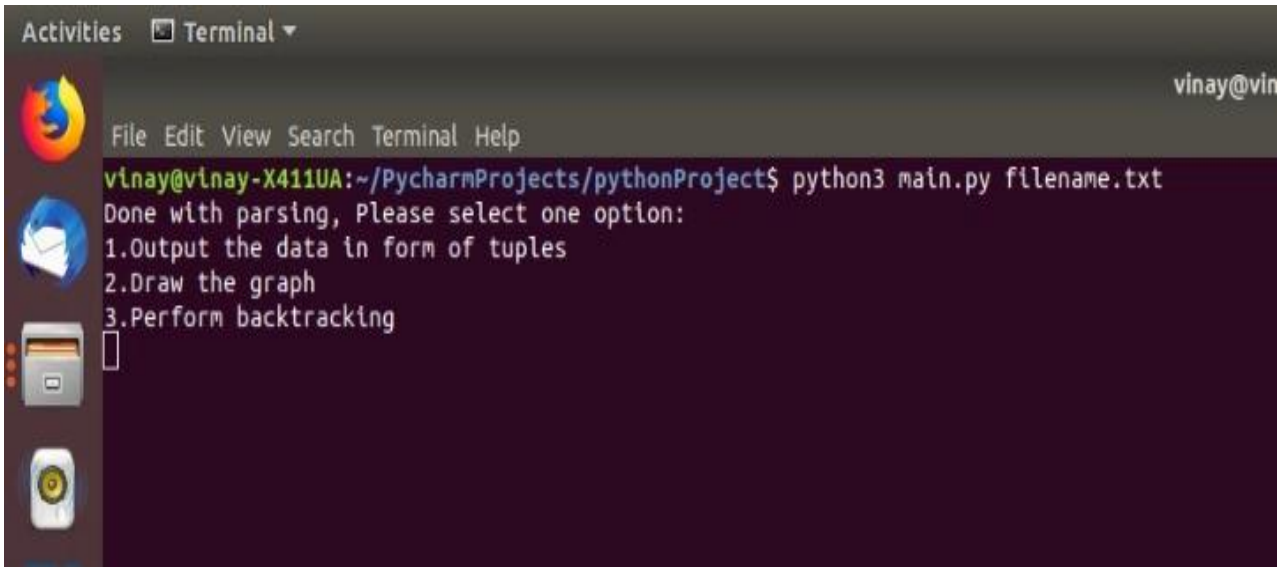
The logs obtained by running the above command is shown below.



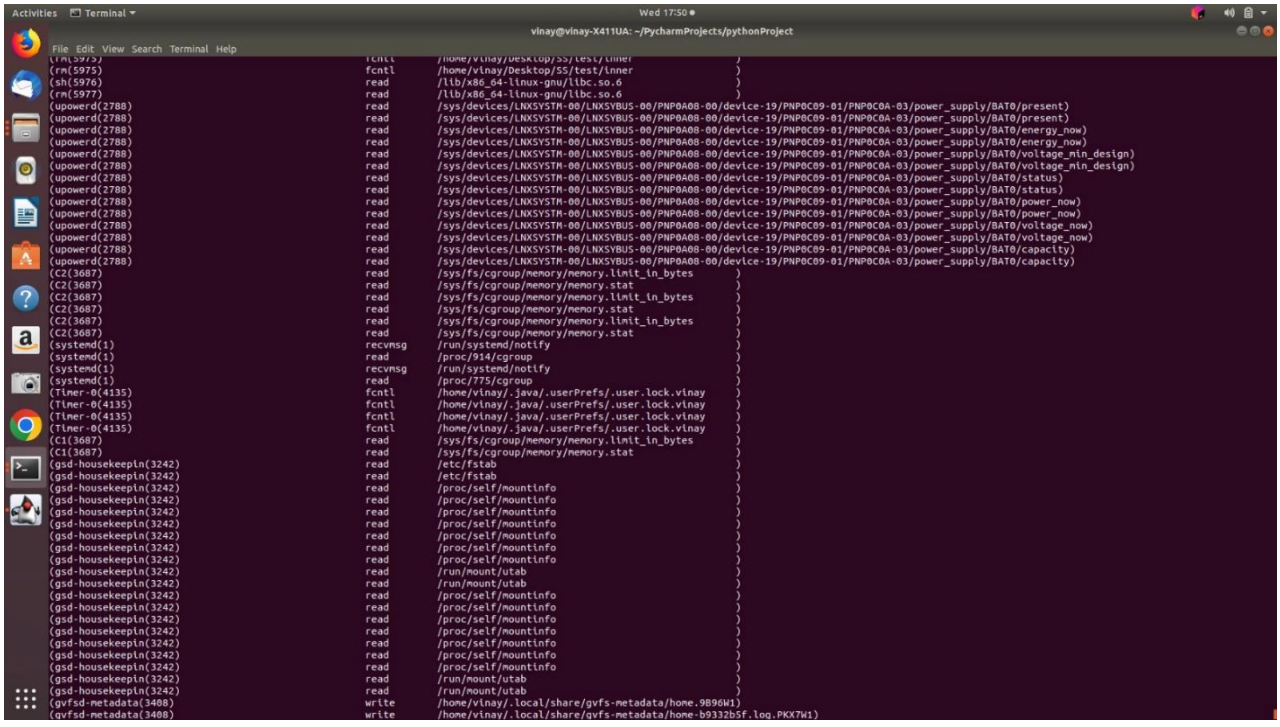
These logs are saved to a file named filename.txt. This file is passed onto the main.py as

an argument.

Output of our code:



This is the home page on running the project file main.py, main.py takes the log file as an argument, parses it and provides above mentioned options to the user. On selecting the first option the below mentioned output is printed on the console. This output consists of tuples holding subject, object and action of every event.



On selecting option 2, a graph final.gv.pdf is generated. This graph consists of all the events logged by the logging tool.

Note: Since the generated graph is very big, it can't be attached to this document instead it is submitted along with this document in the name of final.gv.pdf.

On selecting option 3, the program asks for a node to be backtracked. Once the user gives an appropriate input the program runs the backtracking algorithm to generate finalbfs.gv.pdf file.

As an example we have written a python code which will download a file from googleone drive, unzips it and runs the program present in the zip file. The program writes an output to a file named finaloutput.txt.

The example program:

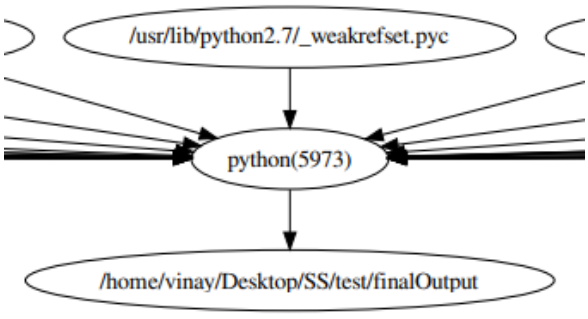
Open ▾ layer4.py ~/Desktop/SS/test

filename.txt ×

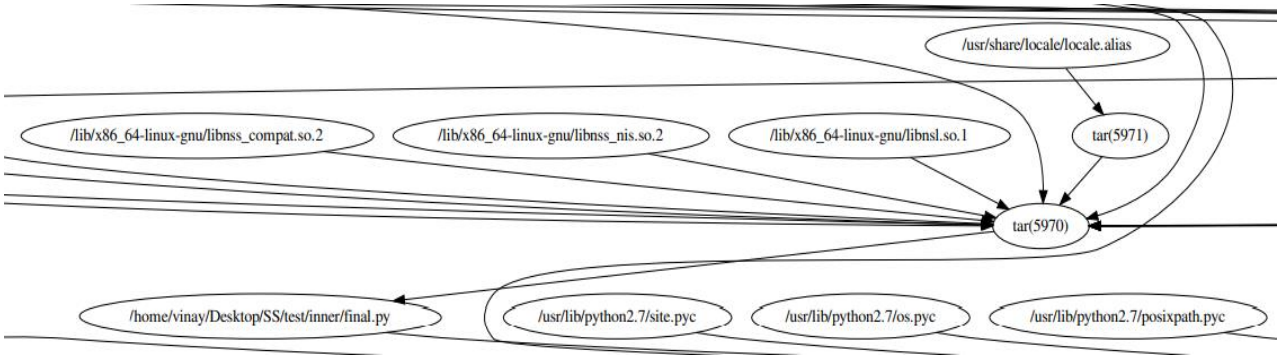
```
mport os

s.system('curl -L "https://docs.google.com/uc?export=download&id=1OT31XEI9UmANZl5A504EN03YUYBkVa_-" > inner.tar.xz')
rint("uncompressing received")
s.system("tar -xf inner.tar.xz")
rint("forking")
s.system("python ./inner/final.py")
s.system("rm -r ./inner")
s.system("rm inner.tar.xz")
```

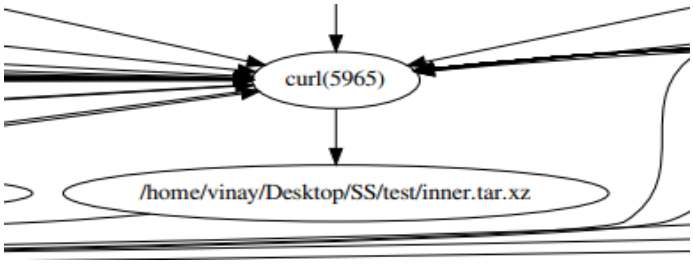
Now we are trying to backtrack finaloutput.txt and therefore provided it as an input for the program. The resultant graph consists of below mentioned parts.



As you can see the file finalOutput is generated by process python (5973) this python process takes several library files which are not of our interest but the input for this file is final.py whose generation is shown below.



As you can see the file final.py is generated by the process tar(5970). Tar is an unzipping process which takes a zip file as an input and extracts the contents of the zip file. The tar has several other dependencies which we are not considering. The input for the tar is a zip file whose generation is shown below.



As you can see the file is generated by process curl (5965) which downloads the file from googleone.drive.

As the backtracking graph is also very big we have mentioned only the files which are responsible for the generation of the finalOutput file. The entire backtracking graph can be viewed in finalbfs.gv.pdf.