

AppArmor - Technical report

Luka Minjoraia

1 Introduction

The aim is to study is to deliver fully documented technical report about AppArmor, its features and all the other technology that was used to fulfill this project.

2 Definition and Methods

2.1 What is AppArmor?

Apparmor is a Mandatory Access Control system which uses LSM kernel enhancements to restrict programs to certain resources. AppArmor does this with profiles loaded into the kernel when the system starts. Apparmor has two types of profile modes, enforcement and complain. Profiles in enforcement mode enforce that profile's rules and report violation attempts in syslog or auditd. Profiles in complain mode don't enforce any profile rules, their purpose is to monitor and log violation attempts.

2.2 Methods

The report in this paper, will go through the work that has been done, which is divided by two parts - 1)development, application and testing of custom AppArmor profiles. 2) Development of python-based application for reducing the routine operation inflexibility and attempting to develop a monitoring system, Both types of the works will be briefly discussed in the paper.

3 Custom profiling for Securing Services and Applications using AppArmor

AppArmor is a powerful MAC system and it has some built-in profiles for commonly used application, so I decided to make a custom profile for something that I'd perhaps use in future and it is not already in the list of built-in profiles, which is Nginx.

3.1 Custom Profile for Nginx

The profile primarily will restrict access to certain directories and allow certain ones, to demonstrate that I will create two directories

```
luka@luka-VirtualBox:~$ sudo mkdir -p /www/allowed  
luka@luka-VirtualBox:~$ sudo mkdir -p /www/notallowed
```

Then index.html files inside these directories :

```
luka@luka-VirtualBox:~$ sudo nano /www/allowed/index.html
...
<html>
    <h1 Safe space </h1>
</html>
luka@luka-VirtualBox:~$ sudo nano /www/notallowed/index.html
...
<html>
    <h1 Unsafe space </h1>
</html>
```

Next, nginx configuration file is modified and nginx service restarted.

```
luka@luka-VirtualBox:~$ sudo nano /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
    # multi_accept on;
}
http {

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
```

```

        server {
            listen      8080;
            location    /{
                root    /www;
            }
        }
    #}
}

```

```
luka@luka-VirtualBox:~$ sudo nginx -s reload
```

Now both directories are available, next step is to create a profile.

```
luka@luka-VirtualBox:/etc/apparmor.d$ sudo aa-autodep nginx
Writing updated profile for /usr/sbin/nginx
```

That will generate a AppArmor profile, now policies need to be entered to limit the access to the directory.

```

sudo gedit /etc/apparmor.d/usr.sbin.nginx
...
# Last Modified: Sun Jun 28 19:05:18 2020
#include <tunables/global>
/usr/sbin/nginx {
    #include <abstractions/apache2-common>
    #include <abstractions/base>
    #include <abstractions/nis>
    capability dac_override,
    capability dac_read_search,
    capability net_bind_service,
    capability setgid,
    capability setuid,
    /www/allowed/* r,
    deny /data/www/notallowed/* r,
    /etc/group r,
    /etc/nginx/conf.d/ r,
    /etc/nginx/mime.types r,
    /etc/nginx/nginx.conf r,
    /etc/nsswitch.conf r,

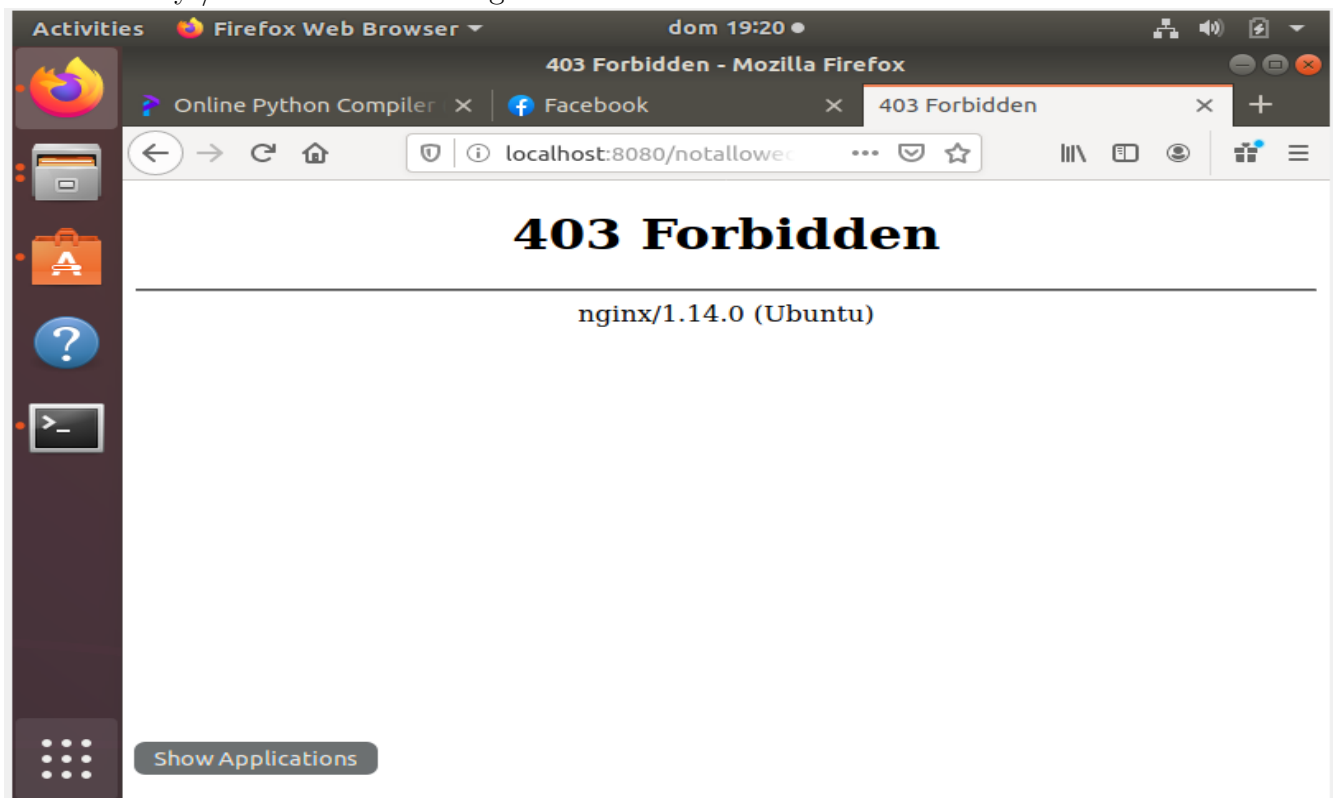
```

```
/etc/passwd r,  
/etc/ssl/openssl.cnf r,  
/run/nginx.pid rw,  
/usr/sbin/nginx mr,  
/var/log/nginx/access.log w,  
/var/log/nginx/error.log w,  
}
```

”deny /www/notallowed/* r,”is the policy that completely disallows the directory, let’s see by enforcing the profile and testing it.

```
sudo aa-enforce nginx  
Setting /usr/sbin/nginx to enforce mode.  
sudo /etc/init.d/apparmor reload  
[ ok ] Reloading apparmor configuration (via systemctl):  
sudo service nginx restart
```

The directory /notallowed is no longer accessible.



4 Command-line based python application for AppArmor re-interpretation

The python application is realized through embedding it into command-line based application, main objective is to ease the inflexibility of AppArmor, provide better filtered results and implement a monitoring system.

4.1 Introduction

The app is made through making python files executable - each command represents each executable that references main module, path is globalized in environment file.

4.2 Source code and demonstration

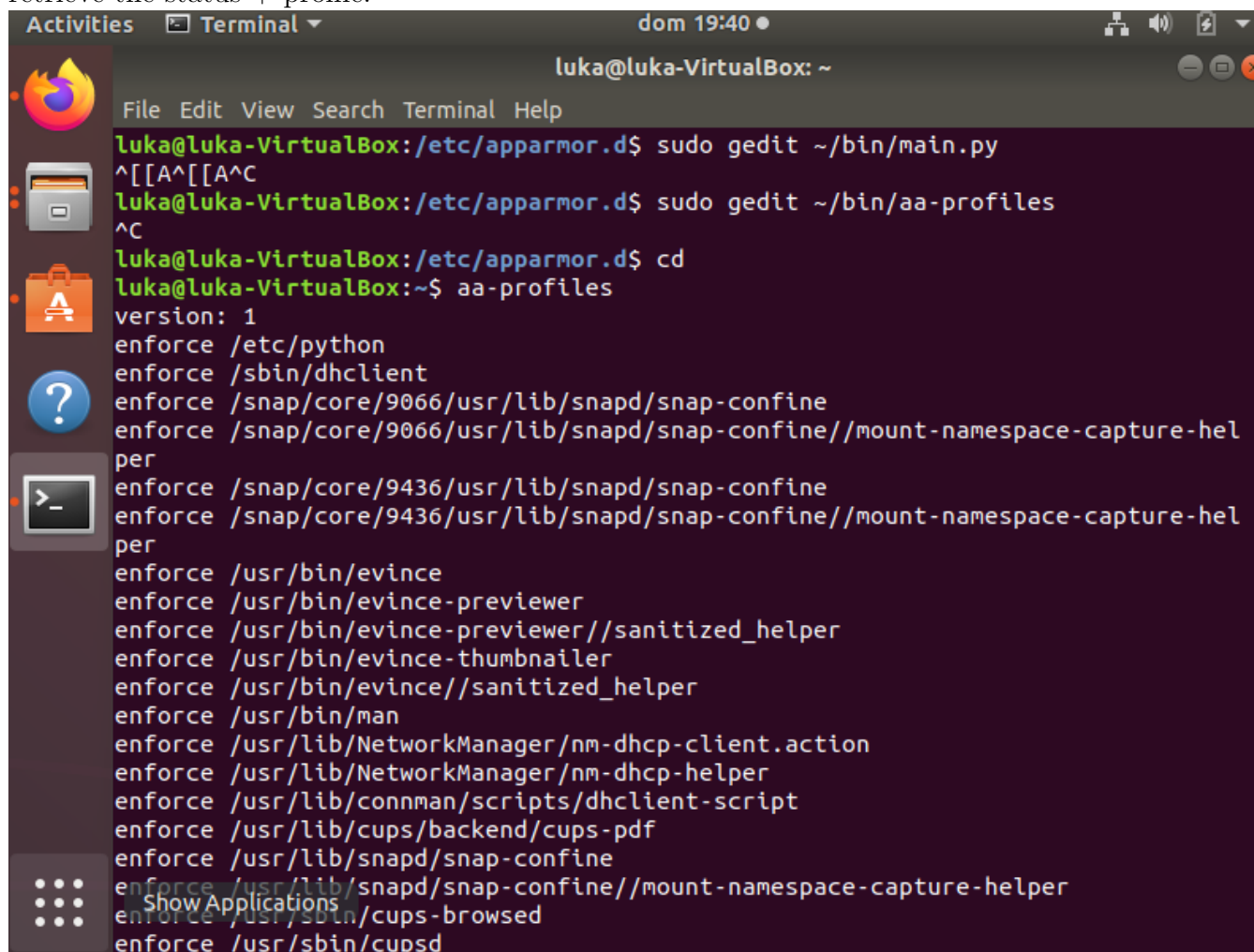
Main code is centralized in main.py which looks like this :

```
#!/usr/bin/python3.8
import subprocess
import json
import os
import argparse
class AppArmorService:
    def get_profiles(self):
        result = subprocess.run(['sudo', 'apparmor-status', '--json'], stdout=subprocess.PIPE)
        data = result.stdout.decode('utf-8')
        p_status = json.loads(data)
        return p_status
    def get_unconfined(self):
        result = subprocess.run(['sudo', 'aa-unconfined', '--paranoid'], stdout=subprocess.PIPE)
        data = result.stdout.decode('utf-8')
        lines = data.split('\n')
        apps = []
        for line in lines:
            app = line.split(' ')
            if len(app) > 1:
                apps.append((app[0], app[1]))
        return apps
    def disable_profile(self, profile):
        os.system('sudo ln -s ' + profile + ' /etc/apparmor.d/disable/')
```

This module contains a class, containing the functionality of each command, which are executed from the separate module - let's take "aa-profiles"

```
#!/usr/bin/env python3.8
import main
service = main.AppArmorService()
profile_iterator = service.get_profiles()
print('version:_ ' + profile_iterator['version'])
for profile in profile_iterator['profiles']:
    status = profile_iterator["profiles"][profile]
    print(status + "_ " + profile)
```

Service is retrieved, formatted and outputted - let's test it out, all I need to run is : "aa-profiles" to retrieve the status + profile.



The screenshot shows a terminal window titled "dom 19:40" with the user "luka@luka-VirtualBox: ~". The terminal displays the following commands and output:

```
luka@luka-VirtualBox:/etc/apparmor.d$ sudo gedit ~/bin/main.py
^[[A^[[A^C
luka@luka-VirtualBox:/etc/apparmor.d$ sudo gedit ~/bin/aa-profiles
^C
luka@luka-VirtualBox:/etc/apparmor.d$ cd
luka@luka-VirtualBox:~$ aa-profiles
version: 1
enforce /etc/python
enforce /sbin/dhclient
enforce /snap/core/9066/usr/lib/snapd/snap-confine
enforce /snap/core/9066/usr/lib/snapd/snap-confine//mount-namespace-capture-helper
enforce /snap/core/9436/usr/lib/snapd/snap-confine
enforce /snap/core/9436/usr/lib/snapd/snap-confine//mount-namespace-capture-helper
enforce /usr/bin/evince
enforce /usr/bin/evince-previewer
enforce /usr/bin/evince-previewer//sanitized_helper
enforce /usr/bin/evince-thumbnailer
enforce /usr/bin/evince//sanitized_helper
enforce /usr/bin/man
enforce /usr/lib/NetworkManager/nm-dhcp-client.action
enforce /usr/lib/NetworkManager/nm-dhcp-helper
enforce /usr/lib/connman/scripts/dhclient-script
enforce /usr/lib/cups/backend/cups-pdf
enforce /usr/lib/snapd/snap-confine
enforce /usr/lib/snapd/snap-confine//mount-namespace-capture-helper
enforce /usr/sbin/cups-browsed
enforce /usr/sbin/cupsd
```

5 Conclusion

The assignment has been incredibly productive on many levels, starting from AppArmor and Linux-python interactions, ending with understanding how internal Linux systems work during the profiling. All of the code that has been used here is uploaded on github : <https://github.com/Lukaa0/Python-AppArmor> and on Virtual.IPB.