MCS: Secure Execution Environments 2023-24

Practical Exercises:
Program confinement with `AppArmor`

Due date: no date

Changelog

- v1.0 - Initial version.

1 Introduction

The goal of this laboratory guide is to show how to make use of AppArmor to create extra access control limitations to programs running in Linux systems.

These exercises must be executed in Linux systems, with a kernel 2.6 or above, which can run on virtual machines.

2 Installation

The `AppArmor` user-land components (the ones that do not belong to the kernel) should already be part of your Linux distribution. Check that with the following command:

```
dpkg -l apparmor
```

This command should present a version number equal or above 3.0.

Furthermore, the `AppArmor` system should be up and running, you can check that with the following command:

```
aa-enabled
```

Also, the `AppArmor` "service" should be already running, you can check that with the following command:

```
service apparmor status
```

or

```
/etc/init.d/apparmor status
```

Some other `AppArmor` tools are available in other packages, namely `apparmor-utils`, which can be installed with the following command:

```
sudo apt install apparmor-utils
```

Also, extra profiles can be installed with the `apparmor-profiles` package:

```
sudo apt install apparmor-profiles
```

3 Installed profiles

`AppArmor` uses a set of profiles installed in the kernel. Profiles can be added in many ways and circumstances, and remain installed until being removed.

Profiles are installed, updated and removed with the `apparmor_parser` command. This command validates the structure of the profiles and, if correct, an suitable for the current kernel, installs them. A new profile is added with the option `-a`, replaced with the option `-r` and deleted with the option `-d`.

The set of installed profiles can be observed, in first hand, with the contents of the pseudo-file `/sys/kernel/security/apparmor/profiles`:

```
sudo cat /sys/kernel/security/apparmor/profiles
```

From the listing of this file one can see the name of all the installed profiles and their enforcement mode (usually enforce and complain).

To get a list of the profiles that are being actually applied to running processes, we can check the contents of their file `/proc/PID/attr/apparmor/current`, where PID stands for their PID. To get this information about a shell console, run the following command:

```
cat /proc/$$/attr/apparmor/current
```

where $$ is an automatic variable that gives the PID of the shell.

All this information is given by the command `aa-status`:

```
sudo aa-status
```

4 Profile editing

For experiencing the creation of profiles, we will first experiment with a C program, `faccess.c`, that tests several ways for opening a file.

```c
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>

void
test_open( char * filename, int mode, char * mode_str )
{
    int fd = open( filename, mode );

    if (fd != -1) {
        printf( "Open %s %s\n", filename, mode_str );
        close( fd );
    }
    else {
        fprintf( stderr, "File %s cannot be open %s (errno = %d, %s)\n",
                filename, mode_str, errno, strerror( errno) );
    }
}

void
test_link( char * filename )
{
    char * new_name = malloc( strlen( filename ) + 5 );
    sprintf( new_name, "%s_new", filename );

    if (link( filename, new_name ) != -1) {
        printf( "Link %s to %s\n", filename, new_name );
    }
    else {
        fprintf( stderr, "Cannot link %s to %s (errno = %d, %s)\n",
                filename, new_name, errno, strerror( errno) );
    }

    unlink( new_name );
}


int
main( int argc, char * argv[] )
{
    umask( 0 );

    if (argc >= 2) {
        while (--argc) {
            test_open( argv[argc], O_WRONLY | O_CREAT | S_IRWXU | S_IRWXG, "WC" );
            test_open( argv[argc], O_RDONLY, "RO" );
            test_open( argv[argc], O_WRONLY, "WO" );
            test_open( argv[argc], O_WRONLY | O_APPEND, "WA" );
            test_open( argv[argc], O_RDWR, "RW" );
            test_link( argv[argc] );
            unlink( argv[argc] );
        }

    }

    return 0;
}
```

For convenience, we will locate this file in the /tmp directory and we will work on it. To compile it, use the command

```
gcc -o faccess faccess.c
```

or simply use the make command for it, since it knows how to make applications from a single C file with the same base name:

```
make faccess
```

4.1 Initial profile

Edit the file faccess-profile and include the following content:

```
abi <abi/3.0>,

include <tunables/global>

@{BASE_DIR}=/tmp

profile file_access @{BASE_DIR}/faccess {
}
```

This is a basically empty profile. Because it denies everything unless otherwise stated, it will severely affect your application.

Install the profile:

```
sudo apparmor_parser -a faccess-profile
```

and check that it belong to the list of the loaded enforcing policies:

```
sudo apparmor_status | less
```

Now, run the faccess command normaly:

```
./faccess x
```

and see that you can successfully perform all operations with the new file x (that is removed at the end).

Now, run the command with the AppArmor confinement:

```
aa-exec -p file_access ./faccess x
```

and verify that you get an error. What is happening is that the application cannot even get access to the shared libraries that it uses, that you can see with this command:

```
ldd faccess
```

Edit the file faccess-profile and include a file from AppArmor abstractions (pre-written permissions) that already solves this issue:

```
abi <abi/3.0>,

include <tunables/global>

@{BASE_DIR}=/tmp

profile file_access @{BASE_DIR}/faccess {
  include <abstractions/base>
}
```

Reinstall the profile:

```
sudo apparmor_parser -r faccess-profile
```

and run again the command with the `AppArmor` confinement:

```
aa-exec -p file_access ./faccess x
```

You can see now that you get errors on all your attempts to access file `x` and create `x_new`.

Edit the file `faccess-profile` and include a line to allow the application to write in files on the `/tmp` directory:

```
abi <abi/3.0>,

include <tunables/global>

@{BASE_DIR}=/tmp

profile file_access @{BASE_DIR}/faccess {
  include <abstractions/base>

  /tmp/** w,
}
```

Note the comma at the end of the new rule.

Reinstall the profile and run again the command with the `AppArmor` confinement. See that now the application is able to perform all open operations for writing (except when opening RW).

Edit again the profile and add to the rule line the write to read files located in `/tmp` and to create new file names (links) in that directory.

```
abi <abi/3.0>,

include <tunables/global>

@{BASE_DIR}=/tmp

profile file_access @{BASE_DIR}/faccess {
  include <abstractions/base>

  /tmp/** wrl,
}
```

Reinstall the profile and run again the command with the `AppArmor` confinement. See that now the application is able to perform all operations.