eCTF Development Help

Ben Janis

January 27, 2021

MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

Sections

- Walkthrough of repository and relevant code
- Walkthrough of Docker
- Walkthrough of creating deployment

- NOTE: This document is designed as a reference to help you understand the implementation of system; you don't need to read the sections in order

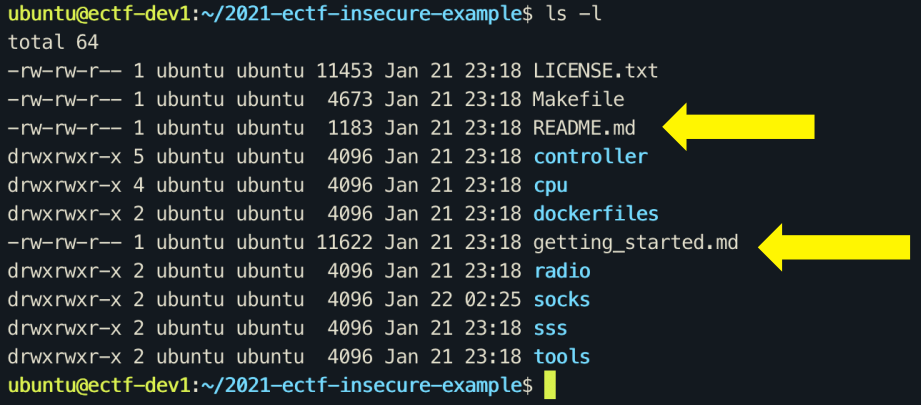
Sections

- **Walkthrough of repository**
- Walkthrough of Docker
- Walkthrough of creating deployment

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Here we are in the root of the repository

Let’s walk through the top-level structure



The first thing you should do once getting the document is to read through the README and getting_started.md

This provides great information about the repository and working with the deployments

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

The top level Makefile contains all of the rules to build and deploy a system

You may not change this file

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

controller/ contains all source code necessary to build the SCEWL Bus Controller

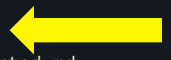
You will be spending much of your development time in this directory

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

cpu/ contains all source code necessary to build the CPUs of various SEDs

You don’t need to change anything in here, though if you want to create an SED to test your design during development, it will go in here

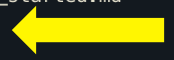
```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```



dockerfiles/ contains the Dockerfiles used to build a deployment

If you need to customize your build process or add packages to the Docker containers, you will need to modify some of these files

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```



radio/ contains the Radio Waves Emulator

You may not change this and can ignore it unless you are curious about how the network emulation backend works

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

socks/ is a directory to hold sockets for the network backend

You can usually ignore this, however if you start getting errors about “socket not found,” check in this directory. If any of the files in here are directories, delete them and try running it again. If you continue to get errors and find directories in here, something during the deployment build process is crashing (likely the SCEWL Bus Controller) and failing to create a socket

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

sss/ contains all files needed for the SSS

All your changes to the SSS must go in this directory

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

tools/ contains a variety of utility tools

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Let’s now look at controller/

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

The controller README provides information about the controller directory in the reference implementation

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

CMSIS/ contains a generic library to interact with ARM CMSIS

You probably can ignore this if you use our provided Makefile as a template

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This Makefile is used to build the SCEWL Bus Controller

It is called during the build process by 2c_build_controller.Dockerfile

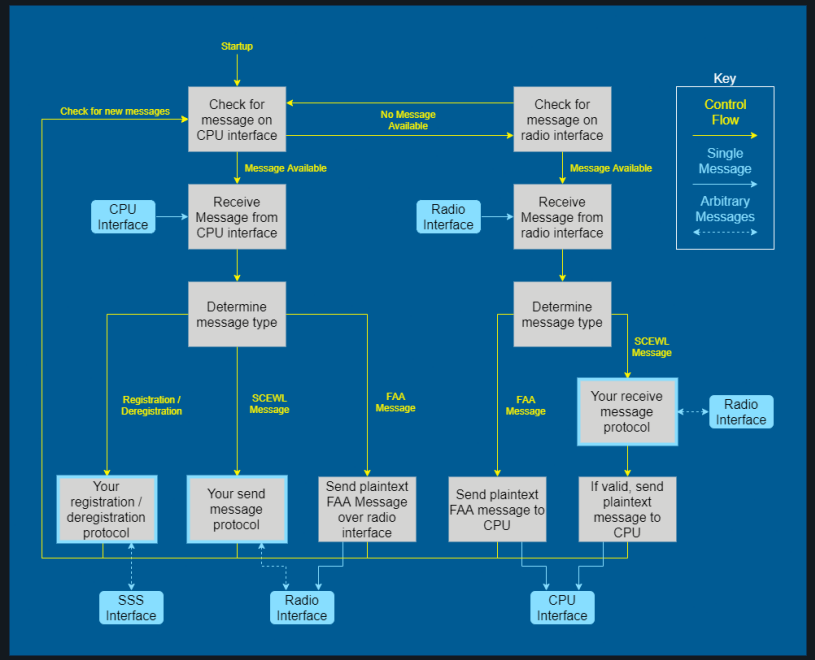
You may modify it to add libraries or files and change the format as you wish

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

controller.c and controller.h implement the main functionality of the SCEWL Bus Controller

Most of your development on the SCEWL Bus Controller will be in this file

It implements the main loop of the controller shown in Figure 5 of the rules document:



```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

interface.c and interface.h implement the raw byte drivers of the three interfaces of the SCEWL Bus Controller: the CPU interface, the SSS interface, and the radio interface

You likely don’t need to change these

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

lm3s/ contains chip-specific libraries and headers for the Stellaris microcontroller being emulated

You likely won’t need to change these

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l controller/
total 40
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:18 CMSIS
-rw-rw-r-- 1 ubuntu ubuntu 2991 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 3335 Jan 21 23:18 README.md
-rw-rw-r-- 1 ubuntu ubuntu 7435 Jan 21 23:18 controller.c
-rw-rw-r-- 1 ubuntu ubuntu 3668 Jan 21 23:18 controller.h
-rw-rw-r-- 1 ubuntu ubuntu 1979 Jan 21 23:18 interface.c
-rw-rw-r-- 1 ubuntu ubuntu 2438 Jan 21 23:18 interface.h
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 lm3s
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 21 23:19 tiny-AES-c
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

tiny-AES-c/ is a git submodule with a simple crypto library

See controller/Makefile for instructions on how to build it into the reference example

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 9 ubuntu ubuntu 4096 Jan 21 23:18 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Now let's look at cpu/

NOTE: You don't need to modify anything in this directory, and any modifications you do make will be ignored

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 9 ubuntu ubuntu 4096 Jan 21 23:18 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This README gives information about the repository

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 9 ubuntu ubuntu 4096 Jan 21 23:18 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

scewl_bus_driver/ contains the implementation of the SCEWL Bus Driver, the library that provides an interface for the user code of the CPU to the SCEWL Bus

You MAY NOT modify any files in here

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 9 ubuntu ubuntu 4096 Jan 21 23:18 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

seds/ contains the source code for the user code of the CPU

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 5 ubuntu ubuntu 4096 Jan 22 03:25 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/seds/
total 16
-rw-rw-r-- 1 ubuntu ubuntu  963 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 common
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 echo_client
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 echo_server
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Looking inside seds/ we can see another README and a few other directories

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 5 ubuntu ubuntu 4096 Jan 22 03:25 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/seds/
total 16
-rw-rw-r-- 1 ubuntu ubuntu  963 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 common
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 echo_client
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 echo_server
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

common/ contains utilities that will be used by attack-phase SEDs and can be ignored during the design phase

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  632 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 scewl_bus_driver
drwxrwxr-x 5 ubuntu ubuntu 4096 Jan 22 03:25 seds
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l cpu/seds/
total 16
-rw-rw-r-- 1 ubuntu ubuntu  963 Jan 21 23:18 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 common
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 echo_client
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 21 23:18 echo_server
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Each type of SED has its own directory in cpu/seds

echo_client and echo_server are the two SEDs that are given to you during the development phase

As their names imply, they implement a server that echoes back any message it receives and a client that sends a message to the echo server and listens for the response

You may modify these for testing your design or use them as a template for writing your own SED

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l dockerfiles/
total 28
-rw-rw-r-- 1 ubuntu ubuntu  285 Jan 21 23:18 0_create_radio.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  784 Jan 21 23:18 1a_create_sss.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  822 Jan 21 23:18 1b_create_controller_base.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  587 Jan 21 23:18 2a_build_cpu.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  568 Jan 21 23:18 2b_create_sed_secrets.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu 1801 Jan 21 23:18 2c_build_controller.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  340 Jan 21 23:18 3_remove_sed.Dockerfile
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Now let's take a quick look at dockerfiles/

The entire build process is done inside Docker containers

Each step of the process is controlled by a Dockerfile

We will go into more detail about these Dockerfiles in the Docker section (starting on slide 37)

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l sss/
total 8
-rwxrwxr-x 1 ubuntu ubuntu 4163 Jan 21 23:18 sss.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Next, let's check out the SSS

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l sss/
total 8
-rwxrwxr-x 1 ubuntu ubuntu 4163 Jan 21 23:18 sss.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```


In the reference implementation, the SSS is implemented as one Python file

Your team is welcome to use another language or add more files, however they all must be in the sss/ directory

Finally, let's check out the tools/ directory

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l tools/
total 20
-rwxrwxr-x 1 ubuntu ubuntu 1217 Jan 21 23:18 deploy_echo.sh
-rwxrwxr-x 1 ubuntu ubuntu 1118 Jan 21 23:18 deploy_re.sh
-rw-rw-r-- 1 ubuntu ubuntu 3561 Jan 21 23:18 faa.py
-rwxrwxr-x 1 ubuntu ubuntu 2142 Jan 21 23:18 launch_sed.sh
-rw-rw-r-- 1 ubuntu ubuntu 2034 Jan 21 23:18 mitm.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l tools/
total 20
-rwxrwxr-x 1 ubuntu ubuntu 1217 Jan 21 23:18 deploy_echo.sh
-rwxrwxr-x 1 ubuntu ubuntu 1118 Jan 21 23:18 deploy_re.sh
-rw-rw-r-- 1 ubuntu ubuntu 3561 Jan 21 23:18 faa.py
-rwxrwxr-x 1 ubuntu ubuntu 2142 Jan 21 23:18 launch_sed.sh
-rw-rw-r-- 1 ubuntu ubuntu 2034 Jan 21 23:18 mitm.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

First are two deployment scripts

These automatically run the make commands to build and launch a deployment

You can modify these however you like or even make your own for easing the testing process, although we will ignore any changes during testing

NOTE: the deploy_re.sh script does not actually build the devices, but rather uses pre-built Docker images that we have pushed to Docker Hub

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l tools/
total 20
-rwxrwxr-x 1 ubuntu ubuntu 1217 Jan 21 23:18 deploy_echo.sh
-rwxrwxr-x 1 ubuntu ubuntu 1118 Jan 21 23:18 deploy_re.sh
-rw-rw-r-- 1 ubuntu ubuntu 3561 Jan 21 23:18 faa.py
-rwxrwxr-x 1 ubuntu ubuntu 2142 Jan 21 23:18 launch_sed.sh
-rw-rw-r-- 1 ubuntu ubuntu 2034 Jan 21 23:18 mitm.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

faa.py is the FAA transceiver that allows for communication with SEDs over the FAA channel

You should not modify this, and if you do, we will ignore any changes during testing

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l tools/
total 20
-rwxrwxr-x 1 ubuntu ubuntu 1217 Jan 21 23:18 deploy_echo.sh
-rwxrwxr-x 1 ubuntu ubuntu 1118 Jan 21 23:18 deploy_re.sh
-rw-rw-r-- 1 ubuntu ubuntu 3561 Jan 21 23:18 faa.py
-rwxrwxr-x 1 ubuntu ubuntu 2142 Jan 21 23:18 launch_sed.sh
-rw-rw-r-- 1 ubuntu ubuntu 2034 Jan 21 23:18 mitm.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

launch_sed.sh automates spinning up a single SED

You may not change this file

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l tools/
total 20
-rwxrwxr-x 1 ubuntu ubuntu 1217 Jan 21 23:18 deploy_echo.sh
-rwxrwxr-x 1 ubuntu ubuntu 1118 Jan 21 23:18 deploy_re.sh
-rw-rw-r-- 1 ubuntu ubuntu 3561 Jan 21 23:18 faa.py
-rwxrwxr-x 1 ubuntu ubuntu 2142 Jan 21 23:18 launch_sed.sh
-rw-rw-r-- 1 ubuntu ubuntu 2034 Jan 21 23:18 mitm.py
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

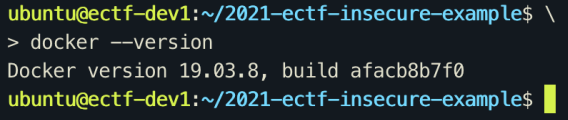
mitm.py is a basic implementation of a man-in-the-middle device for this competition

You may use it during the design phase to view network traffic for debugging and during the attack phase for conducting network attacks

Our reference implementation only intercepts, prints, and forwards all messages, but you may modify the script to do whatever you would like with network traffic during the attack phase

Sections

- Walkthrough of repository and relevant code
- **Walkthrough of Docker**
- Walkthrough of creating deployment



Docker is a system that enables running mini, isolated machines with entire operating systems and file systems on your host machine

Through Docker, you can build environments and run programs in isolation from your host operating system and file system

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ \
> docker --version
Docker version 19.03.8, build afacb8b7f0
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

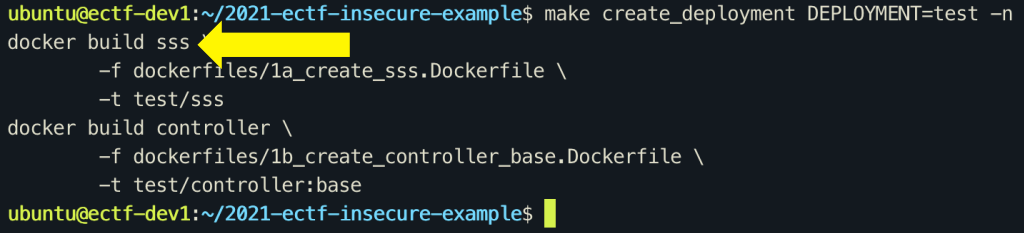
The general workflow with Docker involves two steps:

First, we use “docker build” and Dockerfiles to incrementally build our container into a desired environment

Second, once the environment is set up, we use “docker run” to launch programs inside the container

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test -n
docker build sss \
       -f dockerfiles/1a_create_sss.Dockerfile \
       -t test/sss
docker build controller \
       -f dockerfiles/1b_create_controller_base.Dockerfile \
       -t test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

The general syntax for a call to “docker build” is straightforward



First, a directory is provided for the build environment as context

The container will have access to any files in this directory, so we try to minimize its scope to only what is necessary

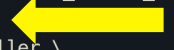
In this case, we are building the SSS, so we only map in the sss directory

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test -n
docker build sss \
        -f dockerfiles/1a_create_sss.Dockerfile \
        -t test/sss
docker build controller \
        -f dockerfiles/1b_create_controller_base.Dockerfile \
        -t test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Next, we provide a Dockerfile

Dockerfiles describe the steps to take to build the container (more on that later)

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test -n
docker build sss \
       -f dockerfiles/1a_create_sss.Dockerfile \
       -t test/sss
docker build controller \
       -f dockerfiles/1b_create_controller_base.Dockerfile \
       -t test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```



Finally, we specify a name or “tag” for the resulting container

For this competition, the tag follows one of the following two formats:

For general containers in a deployment:

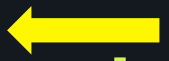
<deployment name>/<component>:latest

For SED-specific containers in a deployment:

<deployment name>/<component>:<SED name>_<SCEWL ID>

See Section 4.1 of the rules for details

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make remove_sed DEPLOYMENT=test SCEWL_ID=10 NAME=test_sed -n
docker rmi -f test/cpu:test_sed_10
docker rmi -f test/controller:test_sed_10
docker build sss \
        -f dockerfiles/3_remove_sed.Dockerfile \
        -t test/sss \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=10
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```



We also have the option of passing arguments to the Dockerfile

Here, we define the argument DEPLOYMENT to be “test” and SCEWL_ID to be “10”

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l dockerfiles/
total 28
-rw-rw-r-- 1 ubuntu ubuntu  285 Jan 21 23:18 0_create_radio.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  784 Jan 21 23:18 1a_create_sss.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  822 Jan 21 23:18 1b_create_controller_base.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  587 Jan 21 23:18 2a_build_cpu.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  568 Jan 21 23:18 2b_create_sed_secrets.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu 1801 Jan 21 23:18 2c_build_controller.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  340 Jan 21 23:18 3_remove_sed.Dockerfile
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Let’s look at the structure of a Dockerfile, starting with 1a_create_sss.Dockerfile

```
# 2021 Collegiate eCTF
# SSS Creation Dockerfile
# Ben Janis
#
# (c) 2021 The MITRE Corporation

FROM ubuntu:focal

# Add environment customizations here
# NOTE: do this first so Docker can used cached containers to skip reinstalling everything
RUN apt-get update && apt-get upgrade -y && \
    apt-get install -y python3

# add any deployment-wide secrets here
RUN mkdir /secrets

# map in SSS
# NOTE: only sss/ and its subdirectories in the repo are accessible to this Dockerfile as .
# NOTE: you can do whatever you need here to create the sss program, but it must end up at /sss
# NOTE: to maximize the useage of container cache, map in only the files/directories you need
#       (e.g. only mapping in the files you need for the SSS rather than the entire repo)
ADD sss.py /sss

```

Here we have the Dockerfile for building the initial SSS

```
# 2021 Collegiate eCTF
# SSS Creation Dockerfile
# Ben Janis
#
# (c) 2021 The MITRE Corporation

FROM ubuntu:focal

# Add environment customizations here
# NOTE: do this first so Docker can used cached containers to skip reinstalling everything
RUN apt-get update && apt-get upgrade -y && \
    apt-get install -y python3

# add any deployment-wide secrets here
RUN mkdir /secrets

# map in SSS
# NOTE: only sss/ and its subdirectories in the repo are accessible to this Dockerfile as .
# NOTE: you can do whatever you need here to create the sss program, but it must end up at /sss
# NOTE: to maximize the useage of container cache, map in only the files/directories you need
#       (e.g. only mapping in the files you need for the SSS rather than the entire repo)
ADD sss.py /sss

```

Dockerfiles all must start with a “FROM” statement

This indicates which container tag to start with as a basis

Here, we start with a container that will come with an Ubuntu Focal OS

In later Dockerfiles, we can use tags that we have created to incrementally modify our containers

```
1  # 2021 Collegiate eCTF
2  # SSS Creation Dockerfile
3  # Ben Janis
4  #
5  # (c) 2021 The MITRE Corporation
6
7  FROM ubuntu:focal
8
9  # Add environment customizations here
10 # NOTE: do this first so Docker can used cached containers to skip reinstalling everything
11 RUN apt-get update && apt-get upgrade -y && \
12     apt-get install -y python3
13
14 # add any deployment-wide secrets here
15 RUN mkdir /secrets
16
17 # map in SSS
18 # NOTE: only sss/ and its subdirectories in the repo are accessible to this Dockerfile as .
19 # NOTE: you can do whatever you need here to create the sss program, but it must end up at /sss
20 # NOTE: to maximize the useage of container cache, map in only the files/directories you need
21 #       (e.g. only mapping in the files you need for the SSS rather than the entire repo)
22 ADD sss.py /sss
23
```

Next, we can use “RUN” statements

These each run a command inside the container and their effects will change the environment of the container

Here, we first install all packages we need for the SSS and then create a secrets directory

NOTE: It is good practice to push steps that may have varying effects across runs as far down the file as possible to utilize Docker’s caching mechanism as much as possible

```
1  # 2021 Collegiate eCTF
2  # SSS Creation Dockerfile
3  # Ben Janis
4  #
5  # (c) 2021 The MITRE Corporation
6
7  FROM ubuntu:focal
8
9  # Add environment customizations here
10 # NOTE: do this first so Docker can used cached containers to skip reinstalling everything
11 RUN apt-get update && apt-get upgrade -y && \
12     apt-get install -y python3
13
14 # add any deployment-wide secrets here
15 RUN mkdir /secrets
16
17 # map in SSS
18 # NOTE: only sss/ and its subdirectories in the repo are accessible to this Dockerfile as .
19 # NOTE: you can do whatever you need here to create the sss program, but it must end up at /sss
20 # NOTE: to maximize the useage of container cache, map in only the files/directories you need
21 #       (e.g. only mapping in the files you need for the SSS rather than the entire repo)
22 ADD sss.py /sss
23
```

Finally, there are “ADD” statements

These add a file from the directory passed as an argument to the build command into the filesystem of the container

Here, we add ssh.py from sss/ in the repository, which will be placed in the root of the container’s file system and named “sss”

NOTE: Every time a file is changed in the context directory, every statement that proceeds after the “ADD” won’t be able to be cached, so place these as far down the file as possible

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 03:25 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l dockerfiles/
total 28
-rw-rw-r-- 1 ubuntu ubuntu  285 Jan 21 23:18 0_create_radio.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  784 Jan 21 23:18 1a_create_sss.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  822 Jan 21 23:18 1b_create_controller_base.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  587 Jan 21 23:18 2a_build_cpu.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  568 Jan 21 23:18 2b_create_sed_secrets.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu 1801 Jan 21 23:18 2c_build_controller.Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu  340 Jan 21 23:18 3_remove_sed.Dockerfile
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

To view a few more advanced statements, let's look at 2c_build_controller.Dockerfile

```
#
# (c) 2021 The MITRE Corporation

ARG DEPLOYMENT

#################################################################
# if you want to copy files from the sss container,             #
# first create an intermediate stage:                           #
#                                                               #
# FROM ${DEPLOYMENT}:sss as sss                                 #
#                                                               #
# Then see box below                                            #
#################################################################

# load the base controller image
FROM ${DEPLOYMENT}/controller:base

# map in controller to /sed
# NOTE: only cpu/ and its subdirectories in the repo are accessible to this Dockerfile as .
ADD . /sed

#################################################################
# Copy files from the SSS container                             #
#                                                               #
# COPY --from=sss /secrets/${SCEWL_ID}.secret /sed/sed.secret   #
#                                                               #
#################################################################
# IT IS NOT RECOMMENDED TO KEEP DEPLOYMENT-WIDE SECRETS IN THE  #
# SED FILE STRUCTURE PAST BUILDING, SO CLEAN UP HERE AS NECESSARY #
#################################################################

# generate any other secrets and build controller
WORKDIR /sed
ARG SCEWL_ID
RUN make SCEWL_ID=${SCEWL_ID}
RUN mv /sed/gcc/controller.bin /controller

# NOTE: If you want to use the debugger with the scripts we provide,
#       the ELF file must be at /controller.elf
RUN mv /sed/gcc/controller.axf /controller.elf
```

First, we can capture arguments passed in by “docker build” with the “ARG” statement

Here, we first capture “DEPLOYMENT” and later we capture “SCEWL_ID”

They can be referenced later with ${arg}

NOTE: Like before, any time this argument changes, Docker will be unable to use its cache for future steps, so ARGs should be pushed down as far as possible in the file

```
#
# (c) 2021 The MITRE Corporation

ARG DEPLOYMENT

#################################################################
# if you want to copy files from the sss container,             #
# first create an intermediate stage:                           #
#                                                               #
# FROM ${DEPLOYMENT}:sss as sss                                 #
#                                                               #
# Then see box below                                            #
#################################################################

# load the base controller image
FROM ${DEPLOYMENT}/controller:base

# map in controller to /sed
# NOTE: only cpu/ and its subdirectories in the repo are accessible to this Dockerfile as .
ADD . /sed

#################################################################
# Copy files from the SSS container                             #
#                                                               #
# COPY --from=sss /secrets/${SCEWL_ID}.secret /sed/sed.secret   #
#                                                               #
#################################################################
# IT IS NOT RECOMMENDED TO KEEP DEPLOYMENT-WIDE SECRETS IN THE  #
# SED FILE STRUCTURE PAST BUILDING, SO CLEAN UP HERE AS NECESSARY #
#################################################################

# generate any other secrets and build controller
WORKDIR /sed
ARG SCEWL_ID
RUN make SCEWL_ID=${SCEWL_ID}
RUN mv /sed/gcc/controller.bin /controller

# NOTE: If you want to use the debugger with the scripts we provide,
#       the ELF file must be at /controller.elf
RUN mv /sed/gcc/controller.axf /controller.elf
```

Next, the “WORKDIR” statement simply changes the working directory of the container for all future statements

This includes future calls to “docker run” from the command line

Here, we move into the sed/ directory before we build the SED

```
 4 #
 5 # (c) 2021 The MITRE Corporation
 6
 7 ARG DEPLOYMENT
 8
 9 ##################################################################
10 # if you want to copy files from the sss container,              #
11 # first create an intermediate stage:                            #
12 #                                                                #
13 # FROM ${DEPLOYMENT}:sss as sss                                  #
14 #                                                                #
15 # Then see box below                                             #
16 ##################################################################
17
18 # load the base controller image
19 FROM ${DEPLOYMENT}/controller:base
20
21 # map in controller to /sed
22 # NOTE: only cpu/ and its subdirectories in the repo are accessible to this Dockerfile as .
23 ADD . /sed
24
25 ##################################################################
26 # Copy files from the SSS container                              #
27 #                                                                #
28 # COPY --from=sss /secrets/${SCEWL_ID}.secret /sed/sed.secret    #
29 #                                                                #
30 ##################################################################
31 # IT IS NOT RECOMMENDED TO KEEP DEPLOYMENT-WIDE SECRETS IN THE    #
32 # SED FILE STRUCTURE PAST BUILDING, SO CLEAN UP HERE AS NECESSARY #
33 ##################################################################
34
35 # generate any other secrets and build controller
36 WORKDIR /sed
37 ARG SCEWL_ID
38 RUN make SCEWL_ID=${SCEWL_ID}
39 RUN mv /sed/gcc/controller.bin /controller
40
41 # NOTE: If you want to use the debugger with the scripts we provide,
42 #       the ELF file must be at /controller.elf
43 RUN mv /sed/gcc/controller.axf /controller.elf
```

Finally, you can copy files from another container using the “FROM … as” and “COPY --from” statements

First, you must create a stage of the build by importing a container with a name using “FROM … as”

Next, we can copy a file into our container with “COPY --from”

Here (commented out) we import the SSS container with the name “sss,” and then copy over a file containing secrets generated during 2b_create_sed_secrets.Dockerfile

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make launch_sss DEPLOYMENT=test SOCK_ROOT=$PWD/socks -n
docker run  -v /home/ubuntu/2021-ectf-insecure-example/socks:/socks \
       test/sss \
       /sss /socks/sss.sock
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

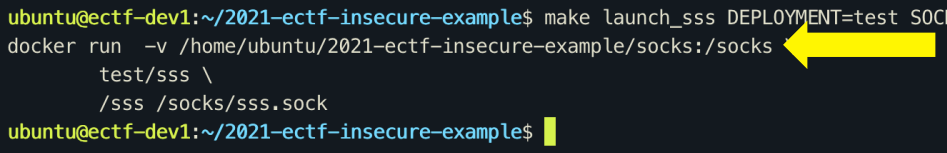
Once we have built our container with “docker build” and Dockerfiles we can now run programs inside the container

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make launch_sss DEPLOYMENT=test SOCK_ROOT=$PWD/socks -n
docker run  -v /home/ubuntu/2021-ectf-insecure-example/socks:/socks \
       test/sss \
       /sss /socks/sss.sock
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```


Again, the syntax is straightforward

First, we provide the tag of the container to run in

Here we use the test/sss container we previously built



Next, we may provide “-v” arguments

Each “-v” argument maps a directory from our host filesystem into the filesystem of Docker

In this case, we are mapping the socks/ directory in the repo into /socks/ in the container

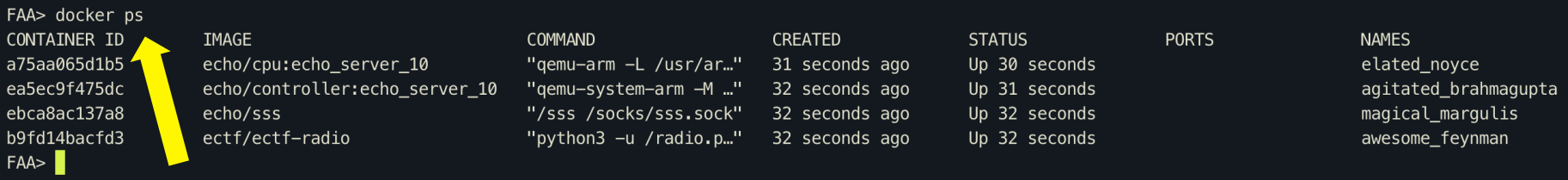
```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make launch_sss DEPLOYMENT=test SOCK_ROOT=$PWD/socks -n
docker run  -v /home/ubuntu/2021-ectf-insecure-example/socks:/socks \
       test/sss \
       /sss /socks/sss.sock
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Finally, we provide the command that we want to run inside that container

Here, we launch the SSS program, passing it the path to the socket /socks/sss.sock inside the container's filesystem (after we mapped it in with "-v") as an argument

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make launch_sss DEPLOYMENT=test SOCK_ROOT=$PWD/socks -n
docker run  -v /home/ubuntu/2021-ectf-insecure-example/socks:/socks \
       test/sss \
       /sss /socks/sss.sock
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Additionally, while not seen here, if we pass "docker run" the "-d" flag, it will run detached from the terminal, although its stdout and stderr may still print to the terminal



After we launch a deployment, the FAA transceiver can interact with Docker directly

With “docker ps” we can see all running docker containers including their container IDs, tags, and the commands that were used to launch them

Here, we can see the deployment running with one SED: the echo server

```
FAA> docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS               NAMES
a75aa065d1b5        echo/cpu:echo_server_10        "qemu-arm -L /usr/ar…"   31 seconds ago      Up 30 seconds                           elated_noyce
ea5ec9f475dc        echo/controller:echo_server_10 "qemu-system-arm -M …"   32 seconds ago      Up 31 seconds                           agitated_brahmagupta
ebca8ac137a8        echo/sss                       "/sss /socks/sss.sock"   32 seconds ago      Up 32 seconds                           magical_margulis
b9fd14bacfd3        ectf/ectf-radio                "python3 -u /radio.p…"   32 seconds ago      Up 32 seconds                           awesome_feynman
FAA> docker kill a75aa
a75aa
FAA> ea5ec9f475dc1603c8d4b00bce66944e46b94c40bb0b4bb953af9fc8a9ca7777

FAA> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ebca8ac137a8        echo/sss            "/sss /socks/sss.sock"   5 minutes ago       Up 5 minutes                            magical_margulis
b9fd14bacfd3        ectf/ectf-radio     "python3 -u /radio.p…"   5 minutes ago       Up 5 minutes                            awesome_feynman
FAA>
```

We can manually kill running containers with “docker kill”

The container to kill can specified by the first few characters of its container ID

```
FAA> docker ps
CONTAINER ID        IMAGE                           COMMAND                  CREATED             STATUS              PORTS               NAMES
a75aa065d1b5        echo/cpu:echo_server_10         "qemu-arm -L /usr/ar…"   31 seconds ago      Up 30 seconds                           elated_noyce
ea5ec9f475dc        echo/controller:echo_server_10  "qemu-system-arm -M …"   32 seconds ago      Up 31 seconds                           agitated_brahmagupta
ebca8ac137a8        echo/sss                        "/sss /socks/sss.sock"   32 seconds ago      Up 32 seconds                           magical_margulis
b9fd14bacfd3        ectf/ectf-radio                 "python3 -u /radio.p…"   32 seconds ago      Up 32 seconds                           awesome_feynman
FAA> docker kill a75aa
a75aa
FAA> ea5ec9f475dc1603c8d4b00bce66944e46b94c40bb0b4bb953af9fc8a9ca7777

FAA> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ebca8ac137a8        echo/sss            "/sss /socks/sss.sock"   5 minutes ago       Up 5 minutes                            magical_margulis
b9fd14bacfd3        ectf/ectf-radio     "python3 -u /radio.p…"   5 minutes ago       Up 5 minutes                            awesome_feynman
FAA>
```

We can manually kill running containers with “docker kill”

The container to kill can specified by the first few characters of its container ID

NOTE: When you kill the CPU or SCEWL Bus Controller of an SED, the other automatically dies, which is why we see the long hash indicating that a detached container has died

```
FAA> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ebca8ac137a8        echo/sss            "/sss /socks/sss.sock"   6 minutes ago       Up 6 minutes                            magical_margulis
b9fd14bacfd3        ectf/ectf-radio     "python3 -u /radio.p…"   6 minutes ago       Up 6 minutes                            awesome_feynman
FAA> docker kill magical_margulis
magical_margulis
FAA> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
b9fd14bacfd3        ectf/ectf-radio     "python3 -u /radio.p…"   6 minutes ago       Up 6 minutes                            awesome_feynman
FAA>
```

We can also kill containers by their shorthand name

```
FAA> docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS              PORTS               NAMES
e68fb9a8349a        echo/cpu:echo_server_10            "qemu-arm -L /usr/ar…"   12 seconds ago      Up 11 seconds                           great_bell
5c130ab80239        echo/controller:echo_server_10     "qemu-system-arm -M …"   14 seconds ago      Up 13 seconds                           nervous_williamson
874b4dfa2f4f        echo/sss                           "/sss /socks/sss.sock"   14 seconds ago      Up 13 seconds                           quirky_poincare
404ad7246a95        ectf/ectf-radio                    "python3 -u /radio.p…"   14 seconds ago      Up 13 seconds                           silly_lewin
FAA> docker kill $(docker ps -q)
e68fb9a8349a
5c130ab80239
874b4dfa2f4f
404ad7246a95
```


We can also kill all running containers at once

You can automate this by adding “alias dockerka=‘docker kill $(docker ps –q)’” to your .bashrc

WARNING: This kills all containers on the system, including those being run by your teammates

Sections

- Walkthrough of repository and relevant code
- Walkthrough of Docker
- **Walkthrough of creating deployment**

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Now that we have all the pieces, let's put them together to build and launch a deployment

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test
docker build sss \
        -f dockerfiles/1a_create_sss.Dockerfile \
        -t test/sss
Sending build context to Docker daemon  7.215kB
Step 1/4 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/4 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y python3
 ---> Using cache
 ---> 893a368cc91f
Step 3/4 : RUN mkdir /secrets
 ---> Using cache
 ---> b2eb5daeb0c1
Step 4/4 : ADD sss.py /sss
 ---> Using cache
 ---> 5077e25fb462
Successfully built 5077e25fb462
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/1b_create_controller_base.Dockerfile \
        -t test/controller:base
Sending build context to Docker daemon  6.869MB
Step 1/2 : FROM ectf/ectf-qemu:latest
 ---> 193bc374c17e
Step 2/2 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y binutils-arm-none-eabi gcc-arm-none-eabi make
 ---> Using cache
 ---> 8c6fed831eed
Successfully built 8c6fed831eed
Successfully tagged test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

First, we create a deployment

The DEPLOYMENT argument defines the name of the deployment and will be kept the same for all steps

To avoid conflicts, different teammates on the same machine should use different values for DEPLOYMENT

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test
docker build sss \
        -f dockerfiles/1a_create_sss.Dockerfile \
        -t test/sss
Sending build context to Docker daemon  7.215kB
Step 1/4 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/4 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y python3
 ---> Using cache
 ---> 893a368cc91f
Step 3/4 : RUN mkdir /secrets
 ---> Using cache
 ---> b2eb5daeb0c1
Step 4/4 : ADD sss.py /sss
 ---> Using cache
 ---> 5077e25fb462
Successfully built 5077e25fb462
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/1b_create_controller_base.Dockerfile \
        -t test/controller:base
Sending build context to Docker daemon  6.869MB
Step 1/2 : FROM ectf/ectf-qemu:latest
 ---> 193bc374c17e
Step 2/2 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y binutils-arm-none-eabi gcc-arm-none-eabi make
 ---> Using cache
 ---> 8c6fed831eed
Successfully built 8c6fed831eed
Successfully tagged test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This first builds an SSS

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test
docker build sss \
        -f dockerfiles/1a_create_sss.Dockerfile \
        -t test/sss
Sending build context to Docker daemon  7.215kB
Step 1/4 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/4 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y python3
 ---> Using cache
 ---> 893a368cc91f
Step 3/4 : RUN mkdir /secrets
 ---> Using cache
 ---> b2eb5daeb0c1
Step 4/4 : ADD sss.py /sss
 ---> Using cache
 ---> 5077e25fb462
Successfully built 5077e25fb462
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/1b_create_controller_base.Dockerfile \
        -t test/controller:base
Sending build context to Docker daemon  6.869MB
Step 1/2 : FROM ectf/ectf-qemu:latest
 ---> 193bc374c17e
Step 2/2 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y binutils-arm-none-eabi gcc-arm-none-eabi make
 ---> Using cache
 ---> 8c6fed831eed
Successfully built 8c6fed831eed
Successfully tagged test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This first builds an SSS

You can see it run each step of the Dockerfile we went through

If it fails, you can look through this output to see exactly where the failing step was, with output printed to stderr shown in red

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test
docker build sss \
        -f dockerfiles/1a_create_sss.Dockerfile \
        -t test/sss
Sending build context to Docker daemon  7.215kB
Step 1/4 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/4 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y python3
 ---> Using cache
 ---> 893a368cc91f
Step 3/4 : RUN mkdir /secrets
 ---> Using cache
 ---> b2eb5daeb0c1
Step 4/4 : ADD sss.py /sss
 ---> Using cache
 ---> 5077e25fb462
Successfully built 5077e25fb462
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/1b_create_controller_base.Dockerfile \
        -t test/controller:base
Sending build context to Docker daemon  6.869MB
Step 1/2 : FROM ectf/ectf-qemu:latest
 ---> 193bc374c17e
Step 2/2 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y binutils-arm-none-eabi gcc-arm-none-eabi make
 ---> Using cache
 ---> 8c6fed831eed
Successfully built 8c6fed831eed
Successfully tagged test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This first builds an SSS

You can see it run each step of the Dockerfile we went through

If it fails, you can look through this output to see exactly where the failing step was, with output printed to stderr shown in red

And finally, the resulting container is tagged

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make create_deployment DEPLOYMENT=test
docker build sss \
        -f dockerfiles/1a_create_sss.Dockerfile \
        -t test/sss
Sending build context to Docker daemon  7.215kB
Step 1/4 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/4 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y python3
 ---> Using cache
 ---> 893a368cc91f
Step 3/4 : RUN mkdir /secrets
 ---> Using cache
 ---> b2eb5daeb0c1
Step 4/4 : ADD sss.py /sss
 ---> Using cache
 ---> 5077e25fb462
Successfully built 5077e25fb462
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/1b_create_controller_base.Dockerfile \
        -t test/controller:base
Sending build context to Docker daemon  6.869MB
Step 1/2 : FROM ectf/ectf-qemu:latest
 ---> 193bc374c17e
Step 2/2 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y binutils-arm-none-eabi gcc-arm-none-eabi make
 ---> Using cache
 ---> 8c6fed831eed
Successfully built 8c6fed831eed
Successfully tagged test/controller:base
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Next, a base SCEWL Bus Controller container is built

You should install all packages needed to build the SCEWL Bus Container here and pre-build any files that don't change across different SEDs

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make add_sed DEPLOYMENT=test SED=echo_server SCEWL_ID=10 NAME=echo_server
docker build cpu \
        -f dockerfiles/2a_build_cpu.Dockerfile \
        -t test/cpu:echo_server_10 \
        --build-arg SED=echo_server \
        --build-arg SCEWL_ID=10 \
        --build-arg CUSTOM=
Sending build context to Docker daemon  35.38kB
Step 1/13 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/13 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y make binutils-arm-linux-gnueabi gcc-arm-linux-gnueabi qemu-user
 ---> Using cache
 ---> 3ea32dde7541
Step 3/13 : ARG SED
 ---> Using cache
 ---> 43ee5f681b5a
Step 4/13 : ARG SCEWL_ID
 ---> Using cache
 ---> 8f4213b260df
Step 5/13 : ARG CUSTOM
 ---> Using cache
 ---> 250eb0367238
Step 6/13 : ADD scewl_bus_driver /scewl_bus_driver
 ---> Using cache
 ---> f3554ade84d5
Step 7/13 : WORKDIR /scewl_bus_driver
 ---> Using cache
 ---> 07a1aa49eed7
Step 8/13 : RUN make SCEWL_ID=${SCEWL_ID}
 ---> Using cache
 ---> d0c2d9c4048b
Step 9/13 : ADD seds/common /common
 ---> Using cache
 ---> 130e527eb0d6
Step 10/13 : ADD seds/${SED} /sed
 ---> Using cache
 ---> a559b3ec25cf
Step 11/13 : WORKDIR /sed
 ---> Using cache
 ---> 9afff67713c9
Step 12/13 : RUN make SED=${SED} SCEWL_ID=${SCEWL_ID} ${CUSTOM}
 ---> Using cache
 ---> d5bf93c37e4b
Step 13/13 : RUN mv /sed/main /cpu
 ---> Using cache
 ---> 151971e8415a
Successfully built 151971e8415a
Successfully tagged test/cpu:echo_server_10
docker build sss \
        -f dockerfiles/2b_create_sed_secrets.Dockerfile \
```

Next, we need to add the SEDs

The SED argument defines the directory in cpu/seds/ to use as a source

SCEWL_ID defines the ID of the SED and must be unique for each new SED

NAME is used in the tag of the container and simply helps keep track of tags

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make add_sed DEPLOYMENT=test SED=echo_server SCEWL_ID=10 NAME=echo_server
docker build cpu \
        -f dockerfiles/2a_build_cpu.Dockerfile \
        -t test/cpu:echo_server_10 \
        --build-arg SED=echo_server \
        --build-arg SCEWL_ID=10 \
        --build-arg CUSTOM=
Sending build context to Docker daemon  35.38kB
Step 1/13 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/13 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y make binutils-arm-linux-gnueabi gcc-arm-linux-gnueabi qemu-user
 ---> Using cache
 ---> 3ea32dde7541
Step 3/13 : ARG SED
 ---> Using cache
 ---> 43ee5f681b5a
Step 4/13 : ARG SCEWL_ID
 ---> Using cache
 ---> 8f4213b260df
Step 5/13 : ARG CUSTOM
 ---> Using cache
 ---> 250eb0367238
Step 6/13 : ADD scewl_bus_driver /scewl_bus_driver
 ---> Using cache
 ---> f3554ade84d5
Step 7/13 : WORKDIR /scewl_bus_driver
 ---> Using cache
 ---> 07a1aa49eed7
Step 8/13 : RUN make SCEWL_ID=${SCEWL_ID}
 ---> Using cache
 ---> d0c2d9c4048b
Step 9/13 : ADD seds/common /common
 ---> Using cache
 ---> 130e527eb0d6
Step 10/13 : ADD seds/${SED} /sed
 ---> Using cache
 ---> a559b3ec25cf
Step 11/13 : WORKDIR /sed
 ---> Using cache
 ---> 9afff67713c9
Step 12/13 : RUN make SED=${SED} SCEWL_ID=${SCEWL_ID} ${CUSTOM}
 ---> Using cache
 ---> d5bf93c37e4b
Step 13/13 : RUN mv /sed/main /cpu
 ---> Using cache
 ---> 151971e8415a
Successfully built 151971e8415a
Successfully tagged test/cpu:echo_server_10
docker build sss \
        -f dockerfiles/2b_create_sed_secrets.Dockerfile \
```

First, the CPU is built and tagged

```
 ---> d5bf95c37e4b
Step 13/13 : RUN mv /sed/main /cpu
 ---> Using cache
 ---> 151971e8415a
Successfully built 151971e8415a
Successfully tagged test/cpu:echo_server_10
docker build sss \
        -f dockerfiles/2b_create_sed_secrets.Dockerfile \
        -t test/sss \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=10
Sending build context to Docker daemon  7.215kB
Step 1/3 : ARG DEPLOYMENT
Step 2/3 : FROM ${DEPLOYMENT}/sss
 ---> 5077e25fb462
Step 3/3 : ARG SCEWL_ID
 ---> Using cache
 ---> 56f25c02608a
Successfully built 56f25c02608a
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/2c_build_controller.Dockerfile \
        -t test/controller:echo_server_10 \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=10
Sending build context to Docker daemon   6.87MB
Step 1/8 : ARG DEPLOYMENT
Step 2/8 : FROM ${DEPLOYMENT}/controller:base
 ---> 8c6fed831eed
Step 3/8 : ADD . /sed
 ---> Using cache
 ---> fe168bb6e645
Step 4/8 : WORKDIR /sed
 ---> Using cache
 ---> ac3ef92cf94c
Step 5/8 : ARG SCEWL_ID
 ---> Using cache
 ---> b94d264b34a4
Step 6/8 : RUN make SCEWL_ID=${SCEWL_ID}
 ---> Using cache
 ---> 2f015e156363
Step 7/8 : RUN mv /sed/gcc/controller.bin /controller
 ---> Using cache
 ---> 8abdffc892b6
Step 8/8 : RUN mv /sed/gcc/controller.axf /controller.elf
 ---> Using cache
 ---> fef2e5aeb183
Successfully built fef2e5aeb183
Successfully tagged test/controller:echo_server_10
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Next, secrets for the SED may be generated in the SSS container

This step imports from the current SSS container, modifies it by generating any necessary secrets for the new SED, and re-tags it, allowing SED secrets to persist with the SSS

```
 ---> d5bf95c37e4b
Step 13/13 : RUN mv /sed/main /cpu
 ---> Using cache
 ---> 151971e8415a
Successfully built 151971e8415a
Successfully tagged test/cpu:echo_server_10
docker build sss \
        -f dockerfiles/2b_create_sed_secrets.Dockerfile \
        -t test/sss \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=10
Sending build context to Docker daemon  7.215kB
Step 1/3 : ARG DEPLOYMENT
Step 2/3 : FROM ${DEPLOYMENT}/sss
 ---> 5077e25fb462
Step 3/3 : ARG SCEWL_ID
 ---> Using cache
 ---> 56f25c02608a
Successfully built 56f25c02608a
Successfully tagged test/sss:latest
docker build controller \
        -f dockerfiles/2c_build_controller.Dockerfile \
        -t test/controller:echo_server_10 \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=10
Sending build context to Docker daemon   6.87MB
Step 1/8 : ARG DEPLOYMENT
Step 2/8 : FROM ${DEPLOYMENT}/controller:base
 ---> 8c6fed831eed
Step 3/8 : ADD . /sed
 ---> Using cache
 ---> fe168bb6e645
Step 4/8 : WORKDIR /sed
 ---> Using cache
 ---> ac3ef92cf94c
Step 5/8 : ARG SCEWL_ID
 ---> Using cache
 ---> b94d264b34a4
Step 6/8 : RUN make SCEWL_ID=${SCEWL_ID}
 ---> Using cache
 ---> 2f015e156363
Step 7/8 : RUN mv /sed/gcc/controller.bin /controller
 ---> Using cache
 ---> 8abdffc892b6
Step 8/8 : RUN mv /sed/gcc/controller.axf /controller.elf
 ---> Using cache
 ---> fef2e5aeb183
Successfully built fef2e5aeb183
Successfully tagged test/controller:echo_server_10
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Finally, the SED itself is built

As we saw earlier, it may copy any secret files it needs from the SSS container using “FROM … as”

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make add_sed DEPLOYMENT=test SED=echo_client SCEWL_ID=11 NAME=echo_client CUSTOM='TGT_ID=10'
docker build cpu \
        -f dockerfiles/2a_build_cpu.Dockerfile \
        -t test/cpu:echo_client_11 \
        --build-arg SED=echo_client \
        --build-arg SCEWL_ID=11 \
        --build-arg CUSTOM=TGT_ID=10
Sending build context to Docker daemon  35.38kB
Step 1/13 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/13 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y make binutils-arm-linux-gnueabi gcc-arm-linux-gnueabi qemu-user
 ---> Using cache
 ---> 3ea32dde7541
Step 3/13 : ARG SED
 ---> Using cache
 ---> 43ee5f681b5a
Step 4/13 : ARG SCEWL_ID
 ---> Using cache
 ---> 8f4213b260df
Step 5/13 : ARG CUSTOM
 ---> Using cache
 ---> 250eb0367238
Step 6/13 : ADD scewl_bus_driver /scewl_bus_driver
 ---> Using cache
 ---> f3554ade84d5
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make add_sed DEPLOYMENT=test SED=echo_client SCEWL_ID=12 NAME=echo_client CUSTOM='TGT_ID=10'
docker build cpu \
        -f dockerfiles/2a_build_cpu.Dockerfile \
        -t test/cpu:echo_client_12 \
        --build-arg SED=echo_client \
        --build-arg SCEWL_ID=12 \
        --build-arg CUSTOM=TGT_ID=10
Sending build context to Docker daemon  35.38kB
Step 1/13 : FROM ubuntu:focal
 ---> 306e6575a9e4
Step 2/13 : RUN apt-get update && apt-get upgrade -y &&     apt-get install -y make binutils-arm-linux-gnueabi gcc-arm-linux-gnueabi qemu-user
 ---> Using cache
 ---> 3ea32dde7541
Step 3/13 : ARG SED
 ---> Using cache
 ---> 43ee5f681b5a
Step 4/13 : ARG SCEWL_ID
 ---> Using cache
 ---> 8f4213b260df
Step 5/13 : ARG CUSTOM
 ---> Using cache
 ---> 250eb0367238
Step 6/13 : ADD scewl_bus_driver /scewl_bus_driver
 ---> Using cache
 ---> f3554ade84d5
```

For the echo deployment, this step needs to be repeated at least one more time to add clients to the deployment

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make remove_sed DEPLOYMENT=test SCEWL_ID=12 NAME=echo_client
docker rmi -f test/cpu:echo_client_12
Untagged: test/cpu:echo_client_12
Deleted: sha256:704966ca0bd2acd09978c31d72404c93d3e1dde0e432222c8d612627dc85817f
Deleted: sha256:4a69eb64c70ea7b5f9d40b1ecbd457b1b19e2a4ae6cadbaca765f1c74174559b
Deleted: sha256:314a832a56633b0358e1feecafa09f1817ee9feb5af268ba63cfc5f3670de62c
Deleted: sha256:0f1f0fbcef4ae9abf7244f45010d8aa08c7b320e26cabb3dd920785ae8ac0604
Deleted: sha256:42336c348064049b727a298ca2223e68bd975a2332ec4f95649ab967bacb2f6e
Deleted: sha256:c49576606fa0b286635f3b9e289244d3a4ee78a5633880b9e4eea8947f4ad7d3
Deleted: sha256:482d0c56beb83ecd441e31489989d58487253e0eadc7ea2d20eac76d62efcc73
Deleted: sha256:4a145f59fe97e79520b0c9a605a9bc85ef54a6519cbd21f95806c52808f473ba
Deleted: sha256:32a4b7f7cf4544bdc65d9cbe4433e364b207d348df1b22dd2df3ab0cb96acbc8
Deleted: sha256:5f9ebaca1692f5ed65b87d9e00f19b342f04e2c841f20a34f5d782c4b6e0ea64
Deleted: sha256:4e9eb083b06b8fd81029bd418c5ddaab28e55a3a3081c6f427bdcfda25b3f7ae
docker rmi -f test/controller:echo_client_12
Untagged: test/controller:echo_client_12
Deleted: sha256:c067ad63977f0c5460c439884a4d6ff836b231ef1e50636f5ebf300faffd8728
Deleted: sha256:27b675b27176d2eb6d75945608359195912cc6cee0b19648cef3948674aa17cb
Deleted: sha256:ecf702a1194862005f6f8b4a777f30c879aa16573e4d7543c9083b0f19c760ed
Deleted: sha256:eac293eed51d937341247fae3b6a3484de052ab4a8027ba75520fac9c770f747
Deleted: sha256:faa066bc450a60bd62b78326555192cae3eabc2fb678ee8cd4740b621a86e319
Deleted: sha256:3582176fd22c3d410202edcde2460ef59ec33856cfcd101fc2c31cb5be0bb349
docker build sss \
        -f dockerfiles/3_remove_sed.Dockerfile \
        -t test/sss \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=12
Sending build context to Docker daemon  6.996kB
Step 1/3 : ARG DEPLOYMENT
Step 2/3 : FROM ${DEPLOYMENT}/sss
 ---> 4eeb1dada333
Step 3/3 : ARG SCEWL_ID
 ---> Running in 6891341259b5
Removing intermediate container 6891341259b5
 ---> 65f16aea3b5b
Successfully built 65f16aea3b5b
Successfully tagged test/sss:latest
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Finally, we must be able to remove an SED from the deployment

This involves first deleting both SED containers

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make remove_sed DEPLOYMENT=test SCEWL_ID=12 NAME=echo_client
docker rmi -f test/cpu:echo_client_12
Untagged: test/cpu:echo_client_12
Deleted: sha256:704966ca0bd2acd09978c31d72404c93d3e1dde0e432222c8d612627dc85817f
Deleted: sha256:4a69eb64c70ea7b5f9d40b1ecbd457b1b19e2a4ae6cadbaca765f1c74174559b
Deleted: sha256:314a832a56633b0358e1feecafa09f1817ee9feb5af268ba63cfc5f3670de62c
Deleted: sha256:0f1f0fbcef4ae9abf7244f45010d8aa08c7b320e26cabb3dd920785ae8ac0604
Deleted: sha256:42336c348064049b727a298ca2223e68bd975a2332ec4f95649ab967bacb2f6e
Deleted: sha256:c49576606fa0b286635f3b9e289244d3a4ee78a5633880b9e4eea8947f4ad7d3
Deleted: sha256:482d0c56beb83ecd441e31489989d58487253e0eadc7ea2d20eac76d62efcc73
Deleted: sha256:4a145f59fe97e79520b0c9a605a9bc85ef54a6519cbd21f95806c52808f473ba
Deleted: sha256:32a4b7f7cf4544bdc65d9cbe4433e364b207d348df1b22dd2df3ab0cb96acbc8
Deleted: sha256:5f9ebaca1692f5ed65b87d9e00f19b342f04e2c841f20a34f5d782c4b6e0ea64
Deleted: sha256:4e9eb083b06b8fd81029bd418c5ddaab28e55a3a3081c6f427bdcfda25b3f7ae
docker rmi -f test/controller:echo_client_12
Untagged: test/controller:echo_client_12
Deleted: sha256:c067ad63977f0c5460c439884a4d6ff836b231ef1e50636f5ebf300faffd8728
Deleted: sha256:27b675b27176d2eb6d75945608359195912cc6cee0b19648cef3948674aa17cb
Deleted: sha256:ecf702a1194862005f6f8b4a777f30c879aa16573e4d7543c9083b0f19c760ed
Deleted: sha256:eac293eed51d937341247fae3b6a3484de052ab4a8027ba75520fac9c770f747
Deleted: sha256:faa066bc450a60bd62b78326555192cae3eabc2fb678ee8cd4740b621a86e319
Deleted: sha256:3582176fd22c3d410202edcde2460ef59ec33856cfcd101fc2c31cb5be0bb349
docker build sss \
        -f dockerfiles/3_remove_sed.Dockerfile \
        -t test/sss \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=12
Sending build context to Docker daemon  6.996kB
Step 1/3 : ARG DEPLOYMENT
Step 2/3 : FROM ${DEPLOYMENT}/sss
 ---> 4eeb1dada333
Step 3/3 : ARG SCEWL_ID
 ---> Running in 6891341259b5
Removing intermediate container 6891341259b5
 ---> 65f16aea3b5b
Successfully built 65f16aea3b5b
Successfully tagged test/sss:latest
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Finally, we must be able to remove an SED from the deployment

This involves first deleting both SED containers

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make remove_sed DEPLOYMENT=test SCEWL_ID=12 NAME=echo_client
docker rmi -f test/cpu:echo_client_12
Untagged: test/cpu:echo_client_12
Deleted: sha256:704966ca0bd2acd09978c31d72404c93d3e1dde0e432222c8d612627dc85817f
Deleted: sha256:4a69eb64c70ea7b5f9d40b1ecbd457b1b19e2a4ae6cadbaca765f1c74174559b
Deleted: sha256:314a832a56633b0358e1feecafa09f1817ee9feb5af268ba63cfc5f3670de62c
Deleted: sha256:0f1f0fbcef4ae9abf7244f45010d8aa08c7b320e26cabb3dd920785ae8ac0604
Deleted: sha256:42336c348064049b727a298ca2223e68bd975a2332ec4f95649ab967bacb2f6e
Deleted: sha256:c49576606fa0b286635f3b9e289244d3a4ee78a5633880b9e4eea8947f4ad7d3
Deleted: sha256:482d0c56beb83ecd441e31489989d58487253e0eadc7ea2d20eac76d62efcc73
Deleted: sha256:4a145f59fe97e79520b0c9a605a9bc85ef54a6519cbd21f95806c52808f473ba
Deleted: sha256:32a4b7f7cf4544bdc65d9cbe4433e364b207d348df1b22dd2df3ab0cb96acbc8
Deleted: sha256:5f9ebaca1692f5ed65b87d9e00f19b342f04e2c841f20a34f5d782c4b6e0ea64
Deleted: sha256:4e9eb083b06b8fd81029bd418c5ddaab28e55a3a3081c6f427bdcfda25b3f7ae
docker rmi -f test/controller:echo_client_12
Untagged: test/controller:echo_client_12
Deleted: sha256:c067ad63977f0c5460c439884a4d6ff836b231ef1e50636f5ebf300faffd8728
Deleted: sha256:27b675b27176d2eb6d75945608359195912cc6cee0b19648cef3948674aa17cb
Deleted: sha256:ecf702a1194862005f6f8b4a777f30c879aa16573e4d7543c9083b0f19c760ed
Deleted: sha256:eac293eed51d937341247fae3b6a3484de052ab4a8027ba75520fac9c770f747
Deleted: sha256:faa066bc450a60bd62b78326555192cae3eabc2fb678ee8cd4740b621a86e319
Deleted: sha256:3582176fd22c3d410202edcde2460ef59ec33856cfcd101fc2c31cb5be0bb349
docker build sss \
        -f dockerfiles/3_remove_sed.Dockerfile \
        -t test/sss \
        --build-arg DEPLOYMENT=test \
        --build-arg SCEWL_ID=12
Sending build context to Docker daemon  6.996kB
Step 1/3 : ARG DEPLOYMENT
Step 2/3 : FROM ${DEPLOYMENT}/sss
 ---> 4eeb1dada333
Step 3/3 : ARG SCEWL_ID
 ---> Running in 6891341259b5
Removing intermediate container 6891341259b5
 ---> 65f16aea3b5b
Successfully built 65f16aea3b5b
Successfully tagged test/sss:latest
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

Finally, we must be able to remove an SED from the deployment

This involves first deleting both SED containers

And then the SSS is imported, modified to remove the SED from the SSS, and retagged

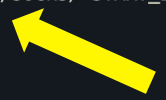
```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ ls -l
total 64
-rw-rw-r-- 1 ubuntu ubuntu 11453 Jan 21 23:18 LICENSE.txt
-rw-rw-r-- 1 ubuntu ubuntu  4673 Jan 21 23:18 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  1183 Jan 21 23:18 README.md
drwxrwxr-x 5 ubuntu ubuntu  4096 Jan 21 23:18 controller
drwxrwxr-x 4 ubuntu ubuntu  4096 Jan 21 23:18 cpu
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 dockerfiles
-rw-rw-r-- 1 ubuntu ubuntu 11622 Jan 21 23:18 getting_started.md
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 radio
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 22 02:25 socks
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 sss
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan 21 23:18 tools
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

With the deployment built, it can now be deployed

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make deploy DEPLOYMENT=test FAA_SOCK=faa.sock MITM_SOCK=mitm.sock SOCK_ROOT=$PWD/socks/ START_ID=10 END_ID=12
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks ectf/ectf-radio \
       python3 -u /radio.py 10 12 faa.sock mitm.sock
24e1b23c402555ceba78c4519a401683fae4e2c589251ab4a87877a1bab42802
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks \
       test/sss \
       /sss /socks/sss.sock
94d8695a04741133092ed08311a6248454df645e6a01f93f28c4a4d51f22d7ea
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

First, we must start the deployment with “deploy”

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make deploy DEPLOYMENT=test FAA_SOCK=faa.sock MITM_SOCK=mitm.sock SOCK_ROOT=$PWD/socks/ START_ID=10 END_ID=12
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks ectf/ectf-radio \
       python3 -u /radio.py 10 12 faa.sock mitm.sock
24e1b23c402555ceba78c4519a401683fae4e2c589251ab4a87877a1bab42802
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks \
       test/sss \
       /sss /socks/sss.sock
94d8695a04741133092ed08311a6248454df645e6a01f93f28c4a4d51f22d7ea
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```



First, we must start the deployment with “deploy”

To avoid conflicts with teammates working on the same machine, avoid using the same SOCK_ROOT

NOTE: SOCK_ROOT must always be an absolute path

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make deploy DEPLOYMENT=test FAA_SOCK=faa.sock MITM_SOCK=mitm.sock SOCK_ROOT=$PWD/socks/ START_ID=10 END_ID=12
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks ectf/ectf-radio \
       python3 -u /radio.py 10 12 faa.sock mitm.sock
24e1b23c402555ceba78c4519a401683fae4e2c589251ab4a87877a1bab42802
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks \
       test/sss \
       /sss /socks/sss.sock
94d8695a04741133092ed08311a6248454df645e6a01f93f28c4a4d51f22d7ea
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This first launches the Radio Wave Emulator detached from the terminal, which ties together the backend network

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make deploy DEPLOYMENT=test FAA_SOCK=faa.sock MITM_SOCK=mitm.sock SOCK_ROOT=$PWD/socks/ START_ID=10 END_ID=12
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks ectf/ectf-radio \
       python3 -u /radio.py 10 12 faa.sock mitm.sock
24e1b23c402555ceba78c4519a401683fae4e2c589251ab4a87877a1bab42802
docker run -d -v /home/ubuntu/2021-ectf-insecure-example/socks/:/socks \
       test/sss \
       /sss /socks/sss.sock
94d8695a04741133092ed08311a6248454df645e6a01f93f28c4a4d51f22d7ea
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$
```

This first launches the Radio Wave Emulator detached from the terminal, which ties together the backend network

Then, the SSS is launched, detached from the terminal

These two should continue running for the lifetime of the deployment

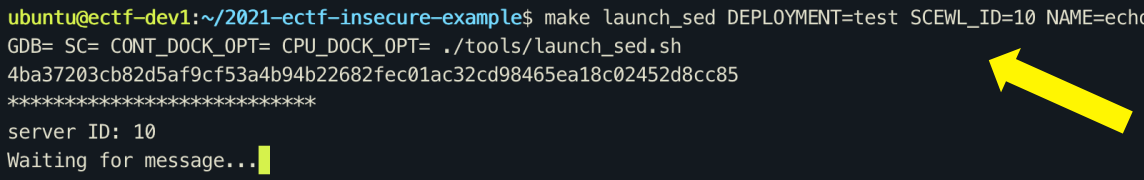
```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make launch_faa FAA_SOCK=socks/faa.sock
python3 tools/faa.py socks/faa.sock
Welcome to the FAA transceiver.
Press enter to check for new messages.
Type help or ? to list commands.

FAA>
```

Before we launch any SEDs, let's launch the FAA transceiver

This opens a command prompt where you can send and receive messages from SEDs

```
ubuntu@ectf-dev1:~/2021-ectf-insecure-example$ make launch_sed DEPLOYMENT=test SCEWL_ID=10 NAME=echo_server SOCK_ROOT=$PWD/socks
GDB= SC= CONT_DOCK_OPT= CPU_DOCK_OPT= ./tools/launch_sed.sh
4ba37203cb82d5af9cf53a4b94b22682fec01ac32cd98465ea18c02452d8cc85
**************************
server ID: 10
Waiting for message...
```

In another terminal, we can now launch our SED

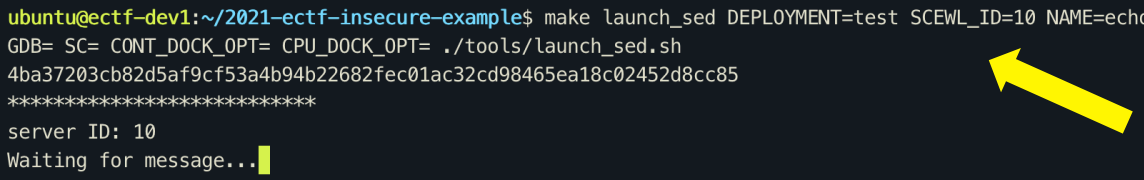


In another terminal, we can now launch our SED

To launch it in the background, use “make launch_sed_d” instead

To launch it with stdin attached, use “make launch_sed_i”

To launch it and hook GDB to the SCEWL Bus Controller, use “make launch_sed_gdb”



The last step to get it running is to launch the echo client and view the flag in the FAA receiver.

I'll leave that to you ☺