Post Module Assignment Submission Form

MODULE TITLE: Cyber Security for Virtualisation Systems

MODULE CODE: WM00I-10

STUDENT ID NUMBER (2096386)

Contents

CSVS PMA Question

The simplest approach is that with no original submission, actually making a submission is the implied most important shortcoming.

The existing re-sub-PMA is (for them) effectively, what are the five most significant shortcomings as a result of making no submission at all.

This effectively reduces to what are the five most important features of the original submission that would get it up to the pass standard and thus demonstrate the achievement of the learning outcomes of the module.

That pretty much will require them to do the original assignment anyway, then prioritise the five most significant things they omitted.

Executive Summary

An organisation has contacted us as Docker specialists to assist in hardening their existing web application and make them runnable in a Docker container. This document will outline the various processes preformed to harden and secure the Docker containers, making them less vulnerable to attacks.

The document will be structured as follows. Section 1 outlines the functional outputs of the web application and the necessary interactions of both containers. Section 2 covers the necessary hardening measures taken to mitigate as many security concerns as possible. Section 3 outlines the created policies and runtime commands the web application will launch with. Lastly, Section 4 will list the various omissions made during the implementation.

Section 1: Preliminary

After completing the installation of the web application with the given files, the web application was launched and tested to understand it's functionality and

the web application was tested for the following inputs, with the

After completing the prior installation of the web application, the web application was tested to understand how it's meant to be interacted with. The following actions are outlined in the Table 1 below, the visual representation shown in Figure 1.

Interaction Type	**Specific Inputs**	**Output**
Action 1 – Normal Input	**Username:** Dave **Suggestion:** Testing!	Username and Suggestion is outputted as normal
Action 2 – No Input	**Username:** **Suggestion:**	Blank line is indented
Action 3 – Only Username	**Username:** Andrew **Suggestion:**	User is outputted without any suggestions
Action 4 – Only Suggestion	**Username:** **Suggestion:** New Test!	Suggestion is supplied without a user
Action 5 – Single Quote	**Username: "** **Suggestion:** "	" is outputted in both User and Suggestion

Table 1: User Inputs into Web Application

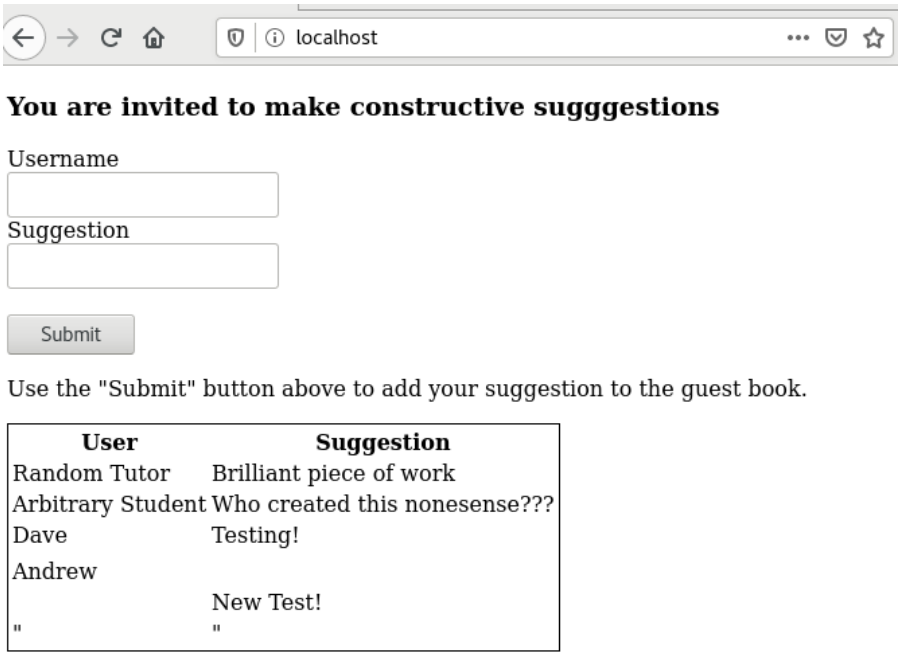

Figure 1: Web Application

The next step was to understand the role of both the web application container and the database container. The following Table 2 describes this and both container's respective interactions.

Container Interaction	Web Container	Database Container
1	Creates website interface.	Contains initial default username and suggestion values
2	Uses POST requests to submit user inputted data.	Stores inputted username and suggestions

Table 2: Container Functionality

Web Container:

- Uses POST requests to submit user inputted data provided in the input fields of "Username" and "Suggestion"
- Creates a simple interface in which a user can interact with the web application

Database Container:

- Stores initial username and suggestion values
- Newly added data is removed when the container is removed and added again

Section 2: Image Stripping

Having further understanding of role of each container and the extent of the web applications functionality, this task will revolve the hardening of the created image. This being done by utilising image stripping.

This image stripping will be done by using the strip-cmd file with it's provided parameters to execute the strip-image file. The strip-image command script requires the correct parameters to strip the image properly. Initial attempts of the image stripping succeeded in drastically lowering the file size, reducing the possibilities of the container be attacked. However, it did not output all the files needed to create a working container.

```
1 #!/bin/bash
2
3 ./strip-image  \
4   -i u2096386/csvs2021-web_i \
5   -t test/stripped-webserver_i:0.5 \
6   -d Dockerfile \
7   -p ngix \
8   -p php \
9   -p php-mysqlnd \
10  -p php-fpm \
11  -d Dockerfile \
12
```

Figure 2: Initial Strip-cmd

From reading the Dockerfile supplied with the web server, the following parameters were supplied to the strip-cmd. The following additions in Figure 3 ensure that the file directories are properly preserved during the image stripping.

```
1 #!/bin/bash
2
3 ./strip-image  \
4   -i u2096386/csvs2021-web_i \
5   -t test/stripped-webserver_i:0.5 \
6   -d Dockerfile \
7   -p ngix \
8   -p php \
9   -p php-mysqlnd \
10  -p php-fpm \
11  -d Dockerfile \
12  -f /etc/nginx/nginx.conf \
13  -f /etc/php.ini \
14  -f /etc/php-fpm.d/www.conf \
15  -f /etc/nginx/conf.d/php-fpm.conf \
16  -f docker-entrypoint.sh
17
```

Figure 3: Correct Strip-cmd

Figure 4 shows the drastic reduction in file size occurring from Strip-cmd.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test/stripped-webserver_i	0.5	f0d21c782399	48 seconds ago	28.9MB
u2096386/csvs2021-web_i	latest	2fbae6a7354c	8 days ago	525MB

Figure 5: Stripped Image

From this newly created image, a container was created and its contents exported to a tar folder. The commands to do so are the following:

docker create --name webstripped_c test/stripped-webserver_i:0.5

docker export -o /tmp/web.tar test/stripped-webserver_i:0.5

A container was created from the image and exported to a .tar file, granting us access to the file directory as seen in Figure 6.

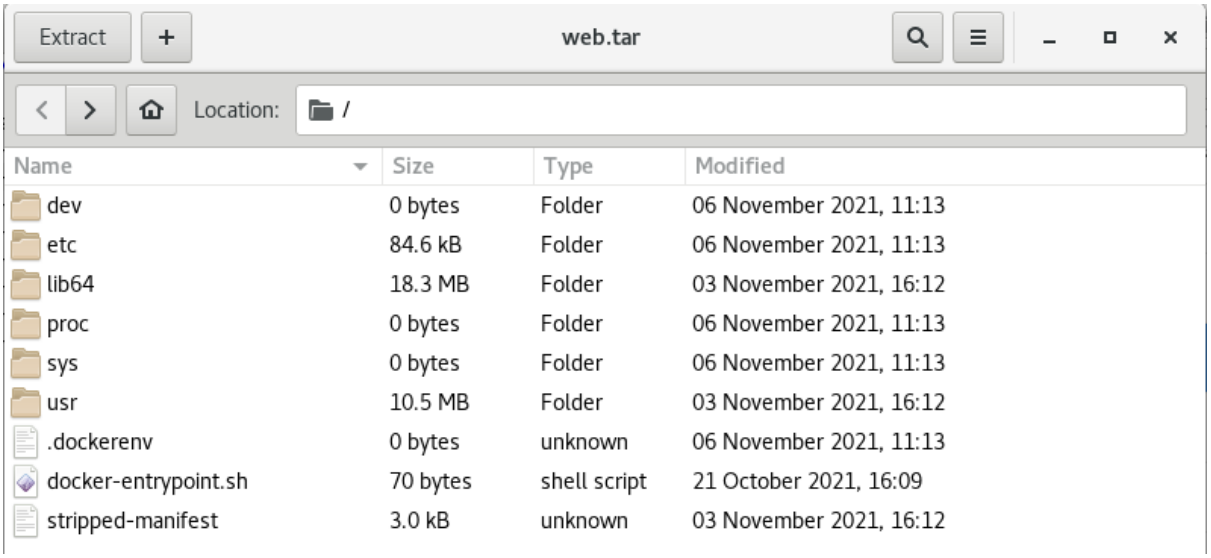

Figure 6: Stripped Image File Directory

The prior section was tested against what was initially presented in the Preliminary section and works against it.

Section 3: Runtime Hardening

SELinux

After running the initial one-off commands which installed and configured SELinux, a policy file of docker_web.pp was created. Next, the container web_c was created from the image of u2096386/csvs2021-web_i. The web server container was run using the following command:

docker run --name web_c -p 80:80 --security-opt label:type:docker_web_t u2096386/csvs2021-web_i

Running the commands of “sudo cat /var/log/audit/audit.log” and “sudo ausearch -m avc --start recent | audit2allow -r” the following audit2allow additions were shown. Table X shows all the initial rules created during this process.

Rule No.	Name of rule	Keep
1	type node_t;	✓
2	type http_port_t;	✓
3	type docker_web_t;	✓
4	class capability { chown net_bind_service setgid setuid };	✓
5	class tcp_socket { bind create listen name_bind node_bind setopt };	✓
6	class udp_socket create;	✓
7	allow docker_web_t http_port_t:tcp_socket name_bind;	✓
8	allow docker_web_t node_t:tcp_socket node_bind;	✓
9	allow docker_web_t self:capability { chown net_bind_service setgid setuid };	✓
10	allow docker_web_t self:tcp_socket { bind create listen setopt };	✓
11	allow docker_web_t self:udp_socket create;	✓

Table 3: Audit2Allow Rules

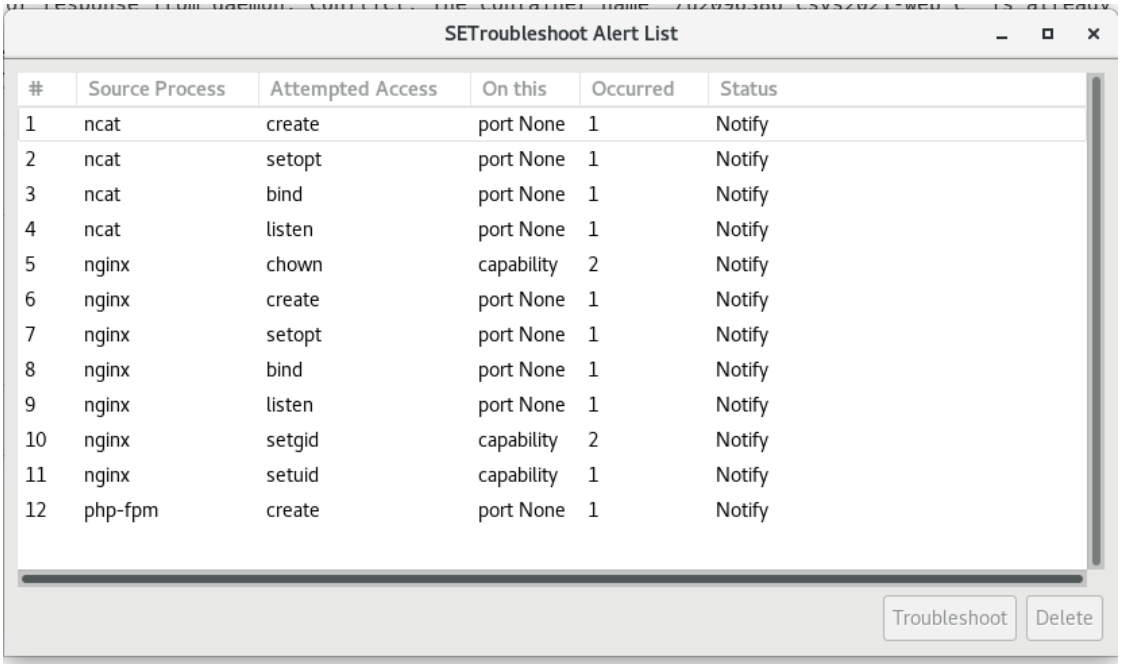

Figure 7: SELinux Troubleshooting Notification

The above Figure 7 are the SELinux troubleshooting alerts raised from the prior process.

When comparing the rules shown in Table 3 to the error message shown in Figure 7, the decision was made that none of the rules needed to be removed, as both the classes and functions were used and removing one would cause SELinux errors later on during this process.

By adding the above rules to the docker_web.te file and repeating the process of recompiling it and amending it to fix whichever errors occurred, we reach a point where there are no more AVC denial and SELinux Trouble shooter reports, the following Table 4 shows the completed docker_web.te file.

Rule No.	Name of rule
1	type node_t;
2	type http_port_t;
3	type docker_web_t;
4	class capability { chown net_bind_service setgid setuid };
5	class tcp_socket { bind accept getattr create listen name_bind node_bind setopt };
6	class udp_socket { create connect getattr setopt };
7	allow docker_web_t http_port_t:tcp_socket name_bind;
8	allow docker_web_t node_t:tcp_socket node_bind;
9	allow docker_web_t self:capability { chown net_bind_service setgid setuid };
10	allow docker_web_t self:tcp_socket { bind accept getattr create listen setopt };
11	allow docker_web_t self:udp_socket { create connect getattr setopt };

Table 4: Amended File

The prior section was tested against what was initially presented in the Preliminary section and works against it.

Seccomp

There was great trouble in creating a Seccomp policy. This is primarily due to the implementation of stracing not working properly. Meaning we were unable to gain the necessary security calls needed to further amend the policy. Since the stracing wasn't successful, the default security calls given to containers was used for the implementation of the web server. Additional information regarding this can be found in the next Important Omissions section, and the runnable commands can be found commented in the webserver Dockerfile.

Important Omissions

Due to the time constraints place on this assignment, the vast number of which in which both containers could be secured, and numerous technical difficulties there were significant omissions made during this entire process. This final section will go into further detail about each of the omissions.

Omission 1: Stracing

As mentioned in the aforementioned Section 3, the implementation of Stracing was not successful. Resulting in us using the default system call, and not those and the container specific calls. This implementation involved creating a new web image and container, instead installing it with the strace package and having a created a readable and writable directory in which our straced results would be outputted to. Had Stracing been successfully implemented, further insight could have been gained into how each container worked. With all the relevant calls being exported to a file sorted, these could have easily been combined with the default system calls to make a strong security policy.

Omission 2: Database Persistence

When creating the database container, the issue of data persistence became an issue. If someone were to relaunch the container and image, the user inputted information to the database would become lost. If there was more time to complete this, a possible solution would be to mount the database on the host machine, with the database container referencing that. This would fix the issue of data persistence and would overall be a much safer and more secure way to go about it.

Omission 3: Security Policy

Despite the default security policy being put in place, it is possible for a user to run the container without having it run against said policy. This could lead to tasks such as using chmod to change the readability and execution of files. A potential fix could be to make only the necessary file executable and make the remaining read-only.