This one’s a bit of a clusterfuck my bad

----------GETTING THE WORDPRESS SERVER UP----------  
  
-Update the docker-compose.yaml if you haven’t already.

-Start up the containers and try accessing the web interface with “http://[IP of VM]:[port you opened]/wp-admin”

-ex: http://192.168.0.138:8000/wp-admin

-You will need to go through some basic installation/account setup but it should be fast. We will come back later to obtain the reverse shell into the server.

***-EV charger: user:ucVrMS3nhGUA0h@up0*** ***-Monorail: user:!Z4rtgiC1erXoHzk3o***

***-Power plant: user:6X\*2d1!a285UG1tFnV***

**Now we need to set custom iptables rules for the specific containers to create the firewalls.**-First add/update some PLC container, I did a cooling pump for the power plant:

Cooling-pump\_Power-plant:

image: rastasheep/ubuntu-sshd

container\_name: Cooling-pump\_Power-plant

command: bash -c "apt update && apt install -y net-tools && apt install -y iputils-ping && /usr/sbin/sshd -D"

cap\_add:

- NET\_ADMIN

security\_opt:

- apparmor:unconfined

networks:

network\_Power-plant:

ipv4\_address: 192.168.102.10

-”cap\_add” and “security\_opt” parameters will let us manipulate the iptables.

-Now we actually define firewall rules in the iptables. Docker exec into the respective containers as listed below and write the appropriate commands as follows (you need “iptables”, so do an “apt update” and “apt install iptables” if needed):

----------PLC CONTAINERS----------

-**iptables -P INPUT DROP**

# sets default behavior to drop all incoming (Input) packets (-P for setting default Policy)

-**iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT**

# '-A INPUT': Append ('-A') a rule to the Input (incoming traffic) ruleset. Note that it specifically appends, meaning adds the rule after the one above. The processing of rules goes sequentially, so first the INPUT DROP rule would get processed and then this one.

# '-m conntrack --ctstate ESTABLISHED,RELATED': loads the state tracking Module ('-m'). It allows the firewall to know what state a packet is in, whether that be a new connection, an existing one, related, or invalid (Stateful Inspection). Essentially a "smart" firewall that remembers previously accepted connections and whether one is related (thereby accepting it), new, or invalid based on the context of the flow of traffic (more efficient). The ESTABLISHED and RELATED flags just tell the firewall to allow packets from an already-established connection or those from related connections.

# '-j ACCEPT': Jumps ('-j') to the ACCEPT action, aka let the packet through.

-**iptables -A INPUT -i lo -j ACCEPT**

# the '-i lo' effectively allows all local traffic to be processed (accepts traffic if the Incoming [-i] interface is the Loopback [lo] interface, aka the interface a system uses to communicate with itself).

-**iptables -A INPUT -p tcp --dport 502 -s 192.168.102.0/24 -j ACCEPT**

# the '-p tcp' specifies that the rule only matches TCP packets, the '--dport 502' specifies that the Destination Port (--dport) is 502 (ModBus default port), and the '-s [IP]' specifies the subnet to apply this rule to (makes it so you can only access the other containers if already on the subnet, aka makes it so the user has to hack the web server and pivot into the network).

* \****NOTE***: MAKE SURE TO CHANGE THE SUBNET TO MATCH THE CORRESPONDING SYSTEM. For example, .102 is for the power plant but you need to change it to .100 for the EV charger and .101 for the monorail. This should be the only thing you have to change about these iptables commands.

----------WEB SERVER CONTAINER----------

-**iptables -P INPUT DROP**

-**iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT**

-**iptables -A INPUT -i lo -j ACCEPT**

-**iptables -A INPUT -p tcp --dport 80 -j ACCEPT**

# specify destination port as 80 (http) instead of 502, and don’t specify a network restriction since it’s a web server that should be able to be accessed from any network.

-----CHECKING TO SEE IF THE RULES APPLIED-----

-**iptables -L -v -n**

# should see the list of rules you made if done correctly (the first rule is represented by the “Chain INPUT (policy DROP …)” line, it’s not listed with the other 3).

-Checking if *only* port 80 (HTTP) is open on the web server:

-Try pinging it. It shouldn’t work because ping doesn’t use port 80.

-Run **nmap -p 1-1000 [IP of web server]** to check all open ports 1-1000. Only 80 should show up (takes a few seconds).

-Checking if PLC containers are inaccessible from outside the subnet:

-Run **nmap -p 502 [IP of PLC container]** to check port 502’s status. It should say closed since we are not on the same subnet(?? \*\***Did this again recently when adding the firewall rules to the EV charger and monorail systems, but now it’s saying the host is down when I try to nmap a PLC container. But this might be the intended behavior since you shouldn’t be able to communicate with one of these unless you’re on the same subnet, which you aren’t when doing it from the local host? Ask Zane about this!)**

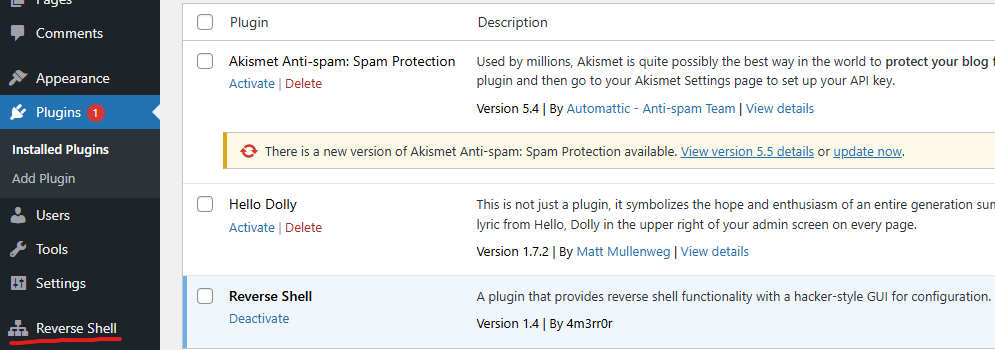
-Run it again after docker exec-ing into the web server. Now you are on the same subnet, so you should see a different result. The port will still say closed since no services are actively listening in on the port yet (no OpenPLC communication yet), but you should see its MAC address indicating that you’re on the same subnet.

----------OBTAINING THE REVERSE SHELL----------

-Download reverse-shell.php from the following link: <https://github.com/4m3rr0r/Reverse-Shell-WordPress-Plugin/tree/main>

-Change the IP near the bottom to that of the host machine (machine/VM running the docker containers. For me I changed it from 127.0.0.1 to 192.168.0.138.

-Zip this file and add it as a plugin. Go Plugins on the left side, Add Plugin, then select a file and choose your zipped reverse-shell.php. Then activate it and you should see a new menu option on the left for Reverse Shell:



-Click on it and you should see a black and green interface, with the info filled in as specified in the php file.

-Run **nc -nvlp 4444** on the local machine (VM) to make it listen on that port. Then click connect and it should work as long as the IP is correct. If you get a “connection refused” it is almost certainly because the VM is not listening on 4444 anymore (it seems to time out after about a minute).

-If it worked, the web server page should start showing the loading icon and you should see “Connection received on [IP of web server]...”. Now you can make sure it works properly by running a “whoami”, which should return “www-data”, or a “id”, which should return the uid, gid, and groups variables. As mentioned before it seems to time out after about a minute, and on the web server interface it says there was a critical error. Not sure if that’s BS and it just simply timed out or if it's an actual error that needs troubleshooting. But it says that it connected literally right above it and the previous commands returned the expected results so I think that error is just BS.

***Useful commands:***

* + **sudo nano docker-compose.yaml**
    - Edit the docker compose file, or create it if it doesn’t exist
  + **sudo docker compose up -d**
    - Start all of the containers defined in the yaml file
  + **sudo docker compose stop**
    - Stops all running containers defined in the yaml file
  + **sudo docker compose exec --user root [name of container] bash**
    - Provides a shell to whichever container you specify, as root. Useful for debugging.
    - **sudo docker exec -it [name of container] bash**
      * For non-docker compose containers (standalone docker containers created manually, without a yaml file).
  + ***sudo docker compose logs [name of container]***
    - Displays logs of specified container. Used on the “guacamole” ones (ones that you connect to in order to get the guacamole web interface), useful for debugging.
  + **sudo docker ps -a**
    - Check all containers and various information about each one (debugging)
    - **sudo docker rm [container name/id]**
      * Remove an old, outdated container
  + **sudo docker image ls**
    - View all currently saved Docker images. Sometimes when trying to rebuild an image, it tries to write to some of the same locations that the old one uses, causing an error. Use this to see and delete (below) conflicting images.
    - **sudo docker image rm [image name/id]**
      * Remove an image
  + **sudo docker container inspect [name of container] --format "{{json .NetworkSettings.Ports}}"**
    - Check specified container’s open ports

----------THINGS TO ASK ZANE ABOUT----------

-General testing of firewall rules to make sure they work as intended

-Specifically the bolded thing above about the nmapping to a PLC container from outside the subnet. Why did it change? Is this the intended behavior and I inadvertently fixed it?

-Rationale for why I can’t ping web servers correct (because ping doesn’t use port 80 and the web servers only listen on port 80)?

-The “Chain FOWARD/OUTPUT” rules in all of the containers when I run “iptables -L -v -n”. What do they mean? They weren’t there before.

-”privileged:tre” flags on wordpress containers. Wasn’t able to manipulate iptables without adding this. Will having this flag trivialize the hacking of the web server?