Unix Security Project Report

TREVOR DALLAS-MAC LELLAN
2283304

For this project, I really wanted to use it as an opportunity to research more into cybersecurity, and to do that I wanted to demonstrate some ways that our machines are vulnerable to bad actors and show off step-by-step some of the ways we can protect ourselves and our machines online.

Firstly, I wanted to demonstrate the risks associated with hosting online services and having misconfigured open ports. For the purposes of keeping this report short, I'm only going to cover SSH and HTTP, since that's what we've covered throughout the scope of this class, also because covering a lot of other webservices and ports would quickly become very repetitive.

General:

There are a lot of general security practices that are important to implement both as a general user and as any kind of administrator overseer many machines. A lot of these are very simple fixes that anyone can add with ease.

- Attackers have many resources available to exploit known vulnerabilities in outdated software and packages, so update them regularly.
    - `sudo apt update && sudo apt upgrade -y`
    - You could also use a tool like unattended-upgrades to upgrade automatically

- Implement strong password rules, or use tools like pam_pwquality.

- Use firewalls and configure rules

    - For easier use, install a firewall like ufw (uncomplicated firewall) for ease of filtering network traffic.

- When having multiple users on a single machine, it's good practice to manage their privileges closely to reduce the risk of privilege escalation and so that regular users can't access files or run commands that they aren't supposed to be able to.

- Even after configuring firewall rules, machine settings, and permissions, the weakest link in any secure system is going to be the user, so it's best to be educated on security issues and safe behaviours.

  - Be aware of phishing attacks, look for misspelled domain names and spelling. Don't click on any links and don't download any files
  - Verify website authenticity before entering any sensitive information
  - Only download files from known and trusted sources
  - Use tools like 2 factor authentication whenever possible

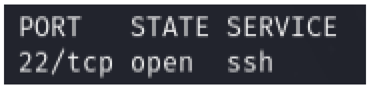
SSH:

To start off, I created two new Debian VMs using VirtualBox as well as a Kali VM to simulate an attacker.

- I started off with installing SSH on both Debian machines
  - `sudo apt-get install openssh-server`

- Then I started the SSH service
  - `sudo systemctl start ssh`
  - `sudo systemctl enable ssh`

- Then, from an attackers point of view, if they used a tool like nmap they would see an open ssh port on the connected machine



```
PORT   STATE SERVICE
22/tcp open  ssh
```

- From this, an attacker can use a tool like Hydra and one of the many publicly available password tables to brute force the password and login. From there, they can use many different methods depending on what's available to them to escalate their privileges in your machine, run scripts, install backdoors, or do anything else they could want to do.

- To prevent this kind of attack is very simple and there are many methods a user could implement.

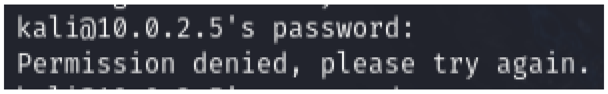
  - Firstly, it’s important to follow strong password rules, that includes:
    - Long passwords, at least 12 characters
    - A mix of numbers, symbols, upper, and lowercase letters
    - Don’t use real words or names
    - Don’t reuse passwords for multiple accounts

  - While strong passwords are good in general, for SSH we can do away with them entirely and rely solely on SSH keys
    - Open SSH configuration with `sudo nano /etc/ssh/sshd_config`
    - Generate SSH keys with `ssh-keygen`
    - SSH into the other machine and copy the public key with `ssh-copy-id ‘username’@’ip’`
    - Set `PasswordAuthentication no` on the machine you’ve copied the key to
    - Restart SSH with `systemctl restart ssh` on that machine
    - Now you can connect to that machine with only keys, deterring brute force attacks

  - You can install rate-limiting with fail2ban, which monitors and punishes repeated failed authentication attempts
    - Install it with `sudo apt-get install fail2ban`
    - Then, configure it with `sudo nano /etc/fail2ban/jail.local`
    - Add the script:

      ```
      [ssh]
      enabled = true
      maxretry = 2
      port = 22
      bantime = 6000
      findtime = 6000
      ```
    - This checks for a connection on our ssh port and IP bans anyone who failed the password check a total of 3 times for 1 hour

- Additionally, you could configure your firewall to only accept SSH connections from just 1 specific IP address
  - By default, Linux uses iptables for it's firewall, so just use the command
    ```
    sudo iptables -A INPUT -p tcp -s 'ip address' –dport 22 -j ACCEPT
    sudo iptables-save | sudo tee /etc/iptables/rules.v4
    ```
  - Now when connecting from a machine that isn't the one we just added, it will display "Permission denied" when trying to connect



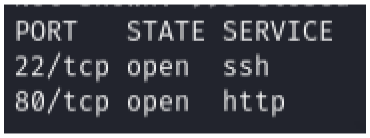
```
kali@10.0.2.5's password:
Permission denied, please try again.
```

HTTP:

Using the same setup as before with 2 different virtual machines, a Debian machine and a Kali machine, I set up an Apache webserver to demonstrate some of it's vulnerabilities and what you can do to fix them.

- Firstly, install and run Apache on the Debian machine
  - `sudo apt-get install apache2`
  - `sudo systemctl start apache2`

- Now, if an attacker were to run an nmap on your ip address, they would see 2 open ports



```
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

- Now with this default configuration, if an attacker ran a command like 'curl' on your webserver, they would get a lot of important information, like software versions and directory listings.

- This is important information to attackers, because like it was mentioned previously, knowing which version software you're running lets attackers exploit known vulnerabilities on that software version and with open directory listings they can browse server files.

- To fix this, we can turn off directory listings by running `sudo nano /etc/apache2/apache2.conf` and removing the "Indexes" option from the Directory block

- We can also remove version information with this command

  - `sudo nano /etc/apache2/conf-available/security.conf`
  - Change `ServerTokens` to `Prod`
  - Change `ServerSignature` to `Off`

- After making these changes, just restart the webserver
  - `sudo systemctl restart apache2`

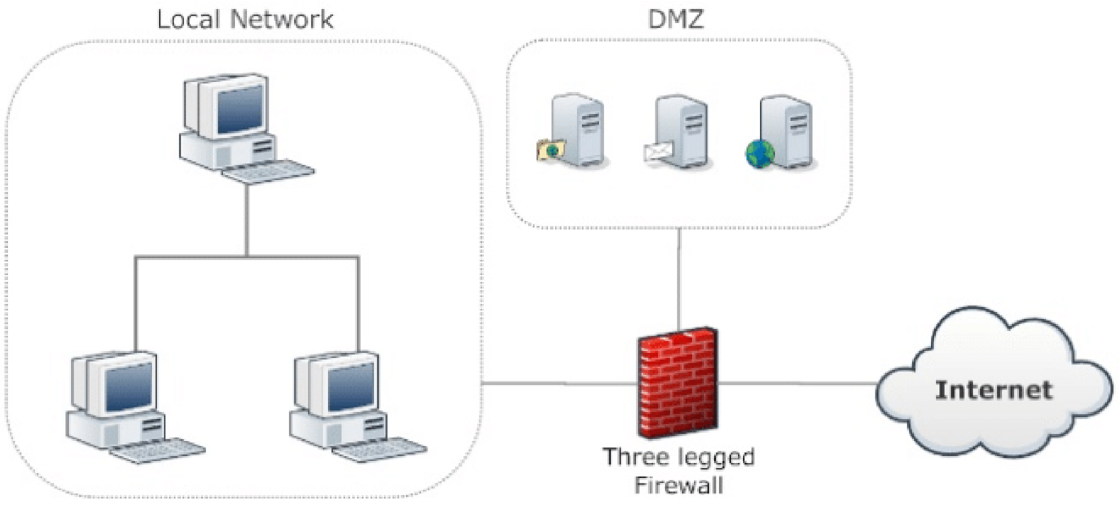
- You can use a package like certbot to get an SSL certificate
  - `sudo apt install certbot python3-certbot-apache`
    `sudo certbot –apache`

- You should also use fail2ban to monitor and ban failed login attempts using the same config we used with SSH and modifying it slightly to fit the webserver

```
[apache-auth]
enabled = true
port = http,https
filter = apache-auth
maxretry = 2
bantime = 6000
findtime = 6000
```

- You can also use a tool like a Web Application Firewall, or use UFW to manage firewall rules to filter traffic based on your specific needs.

Finally, the last thing I wanted to cover in this project is a network topology known as a DMZ or Demilitarized Zone.



The DMZ network adds another degree of separation from your machines to the internet. Instead of running your services like SSH and Apache webservers from your own machine and having the vulnerabilities be directly into your own machine where you keep sensitive information, you instead run them on one of the DMZ machines that acts sort of as the middle-man for internet traffic. For example, lets say you're running a webserver on your own machine, all the database files for the site are stored locally, and if someone could brute force admin credentials on your webserver they'd have access to critical and sensitive files and information on your local machine, however if you instead ran the webserver in a DMZ, an attacker would instead break into a machine with nothing important on it and they would have a much harder time trying to get access to your own machine in a properly configured DMZ.

Configuring a DMZ is a relatively straightforward process, but again like with demonstrating HTTP servers, there are just so many different uses and situations that could be covered that it goes beyond the scope of this report, but the process for creating a DMZ is generally the same.

You want to configure the firewall to block all incoming and outgoing traffic by default, with iptables it should look something like this

```
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD
sudo iptables -P OUTPUT ACCEPT
```

After this, you want to configure it to allow only incoming HTTP traffic and to drop all other traffic

```
sudo iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 80 -j ACCEPT

sudo iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 443 -j ACCEPT

sudo iptables -A FORWARD -i eth0 -o eth1 -j DROP
```

For this example, let’s assume that your webserver needs a database, which you keep on your local machine, so for your webserver to work correctly it needs to be able to receive SQL data which by default, MySql uses port 3306

```
sudo iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 3306 -j ACCEPT

sudo iptables -A FORWARD -i eth1 -o eth2 -j DROP
```

Then, to allow administrators to access and modify the webserver if needed, you can allow for an SSH connection as well

```
sudo iptables -A FORWARD -i eth2 -o eth1 -p tcp --dport 22 -j ACCEPT

sudo iptables -A FORWARD -i eth2 -o eth1 -j DROP
```

Then, allow for outgoing traffic.

```
sudo iptables -A FORWARD -i eth1 -o eth0 -m conntrack –ctstate
NEW,ESTABLISHED -j ACCEPT

sudo iptables -A FORWARD -i eth1 -o eth0 -j DROP
```

Finally, after all that, you can save your new firewall settings and restart your webserver