

# HONEYPOT REPORT

## CONTENTS

INTRODUCTION ..... 1

DOMAIN OF KNOWLEDGE ..... 1

ABSTRACT ..... 2

IMPLEMENTATION..... 3

RESOURCE MONITORING ..... 17

PERFORMANCE EVALUATION..... 21

CONCLUSION..... 25

REFERENCING..... 26

## INTRODUCTION

The Raspberry Pi OS is a Debian-based Linux distribution designed to operate on Raspberry Pi devices. It is highly versatile and supports a range of third-party operating systems, including Ubuntu and RISC OS, making it a popular choice for both hobbyists and professionals. Python and Scratch are the two primary programming languages associated with the Raspberry Pi, but the platform also supports a variety of other programming languages, allowing for broad applications in software development, education, and research (Wikipedia, 2024).

For this coursework, I will be using the Raspberry Pi 4, specifically the Raspbian operating system. The Raspberry Pi 4 Model B, which features a 4GB RAM processor, functions as a powerful minicomputer capable of handling a wide array of projects, including networking tasks, IoT applications, and scientific experiments. Its affordability and adaptability make it a valuable tool for exploring cybersecurity concepts, such as creating and analyzing honeypots to enhance system security.

## DOMAIN OF KNOWLEDGE

This coursework incorporates key technical concepts and methodologies, including:

1. **TCP/IP Protocols and Subnetting:** These are foundational to understanding network communication and segmentation. Subnetting allows for more efficient management of IP addresses, which is crucial for isolating and analyzing network traffic.
2. **Understanding Network Subnets:** Subnets enable administrators to divide large networks into smaller, manageable sections, improving security and performance by limiting unnecessary traffic.
3. **Htop for Process Monitoring:** Htop is an interactive process viewer used to monitor the performance of the Raspberry Pi, providing real-time insights into CPU, memory, and process utilization. This is essential for analyzing the system's behavior under different conditions, such as during attacks or stress testing.
4. **Nmap for Network Scanning and Vulnerability Assessment:** Nmap is a powerful open-source tool used to scan networks for open ports, services, and vulnerabilities. In this coursework, Nmap is used to simulate attacks on the honeypot, providing insight into how the system reacts to different forms of reconnaissance and probing. By analyzing the honeypot's logs during these scans, I can assess the system's performance and security measures.

These concepts are crucial to effectively implementing and evaluating the honeypot, as they provide a foundation for understanding how network traffic is managed and how the system's resources are utilized during operation.

---

## ABSTRACT

The primary objective of this coursework is to design and deploy a honeypot on a Raspberry Pi to observe and analyze potential attacks on the system. A honeypot is a decoy server or network setup that mimics a legitimate service to attract and study malicious activity. By redirecting traffic away from the actual server to the honeypot, it provides a layer of security for the real infrastructure. Honeypots serve a dual purpose: improving system security by mitigating threats and offering valuable insights into the behavior of attackers through detailed traffic analysis (Kaspersky, 2024).

In this project, the honeypot will capture and log various forms of malicious activity, with a focus on web logs. Web logs provide a detailed record of interactions with the honeypot, including unauthorized access attempts, the use of malicious tools, and other suspicious behavior. This information can be used to strengthen cybersecurity defenses and understand the tactics, techniques, and procedures (TTPs) of cybercriminals.

Additionally, the coursework involves configuring and analyzing resource monitoring tools, such as Htop, to evaluate the Raspberry Pi's performance under different conditions. This includes monitoring system resources such as CPU, memory, and network usage during normal operation and simulated attacks. Through this process, I aim to demonstrate how a honeypot can act as both a protective and investigative tool, contributing to the broader field of cybersecurity.

---

# IMPLEMENTATION

The implementation of the honeypot involved several steps, including creating user accounts, installing and configuring necessary software, and deploying the DShield honeypot. Below is a detailed step-by-step breakdown:

---

## Creating User Accounts

The first step was to create dedicated user accounts on the Raspberry Pi. These accounts allow better management and isolation of tasks during configuration. User accounts were created using the following commands:

- `sudo useradd rit2311`
- `sudo useradd muhpat18`

By adding these users, we ensured that the system was properly configured for multiple users to access and work on the project securely. This also aligns with best practices for managing system access.

---

## DShield Honeypot

For this project, I utilized DShield, a widely-used low-interaction honeypot designed to log intrusion attempts and firewall activity. The primary goal of using DShield was to gather data about potential malicious activity, including unauthorized access attempts and reconnaissance efforts targeting the system.

DShield is a lightweight solution that simulates a vulnerable system, making it attractive to attackers while logging their activities. The logs generated by DShield provide valuable insights into attacker behavior, which can be analyzed to improve system defenses.

---

## Installing Git

Git is a widely-used version control system created by Linus Torvalds in 2005, and it is maintained by Junio Hamano. It enables tracking code changes and collaborative development, making it a foundational tool for projects requiring software installation or updates from online repositories.

To install Git on the Raspberry Pi, the following command was used: `sudo apt-get -y install git`

This command ensures that all necessary packages and libraries associated with Git are installed. Once Git was successfully installed, it was used to clone the DShield honeypot software from GitHub

```

rit2311@raspberrypi:~ $ sudo apt-get -y install git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.39.5-0+deb12u1).
The following packages were automatically installed and are no longer required:
 chromium-browser chromium-browser-l10n chromium-codecs-ffmpeg-extra
 libqt5qmlworkerscript5 libqt5quickcontrols2-5 libqt5quicktemplates2-5
 qml-module-qtgraphicaleffects qml-module-qtquick-controls2
 qml-module-qtquick-layouts qml-module-qtquick-templates2
 qml-module-qtquick-window2 qml-module-qtquick2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

```

It also shows all the files in execution which are being installed and imported from git. After git is installed we download a copy of DShield from github.

```

rit2311@raspberrypi:~ $ git clone https://github.com/DShield-ISC/dshield.git
Cloning into 'dshield'...
remote: Enumerating objects: 5703, done.
remote: Counting objects: 100% (2245/2245), done.
remote: Compressing objects: 100% (601/601), done.
remote: Total 5703 (delta 1751), reused 2085 (delta 1623), pack-reused 3458 (from 1)
Receiving objects: 100% (5703/5703), 3.52 MiB | 7.67 MiB/s, done.
Resolving deltas: 100% (3591/3591), done.
rit2311@raspberrypi:~ $ cd dshield/bin/

```

This will clone DShield into raspberry file. After DShield is installed we are going to change the directory to bin and it will move all the content of DShield in the bin directory.

After that we run the installation script.

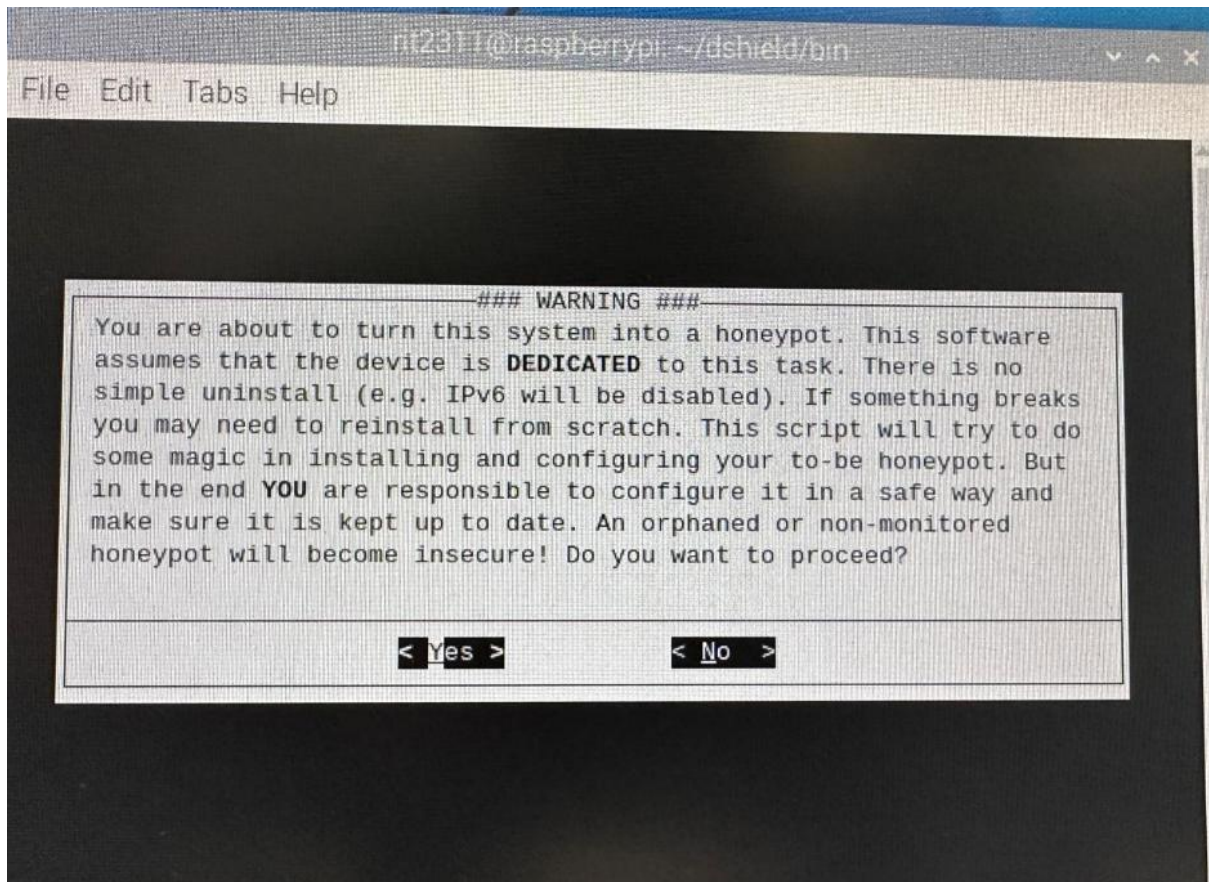
```

rit2311@raspberrypi: ~/dshield/bin
File Edit Tabs Help
#####
Log /srv/log/install_2024-11-21_113458.log started.
ATTENTION: the log file contains sensitive information (e.g. passwords,
           API keys, ...). Handle with care and sanitize before sharing.
Checking Pre-Requisites
ATTENTION: the latest versions of this script have been tested on:
- Raspbian OS
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04
- openSUSE Tumbleweed.
It may or may not work with your distro. Feel free to test and contribute.
Press ENTER to continue, CTRL+C to abort.

```

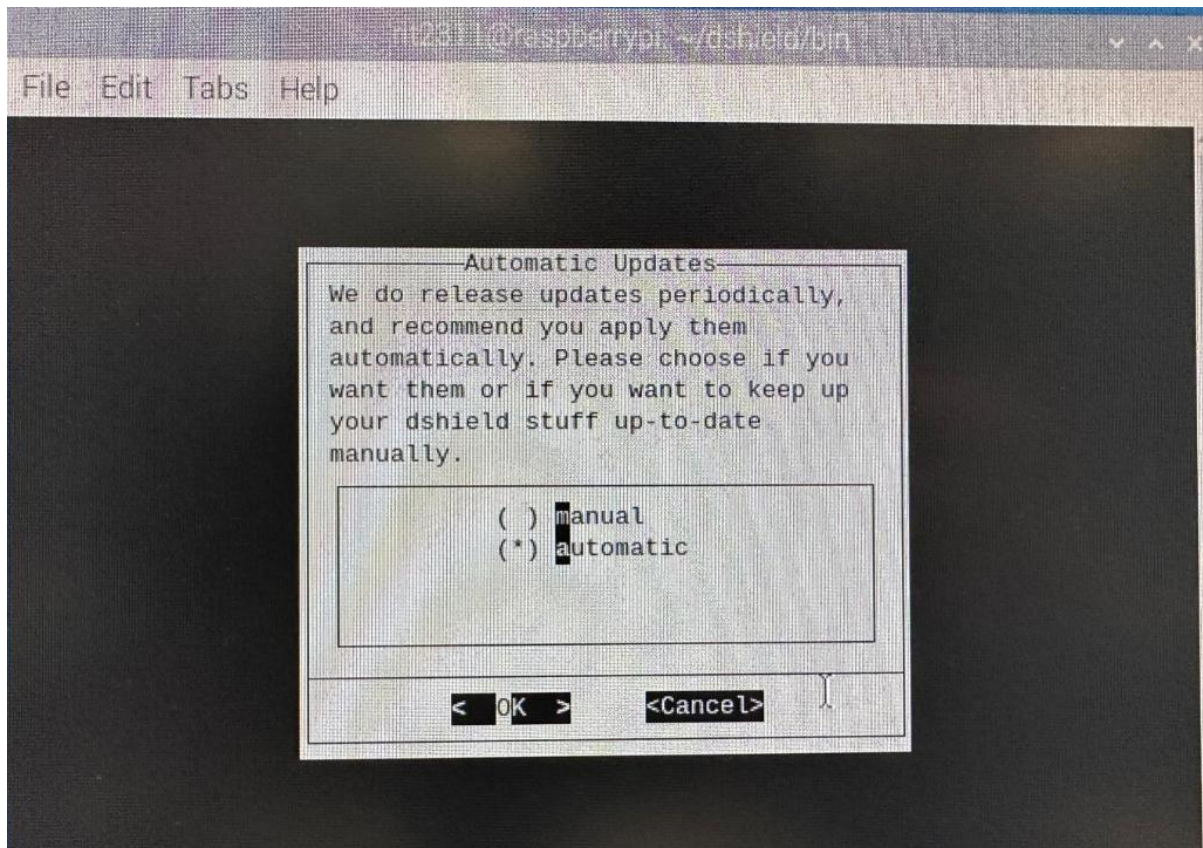
As you can see its installing the server log file which contains the login details like passwords, API keys, etc.



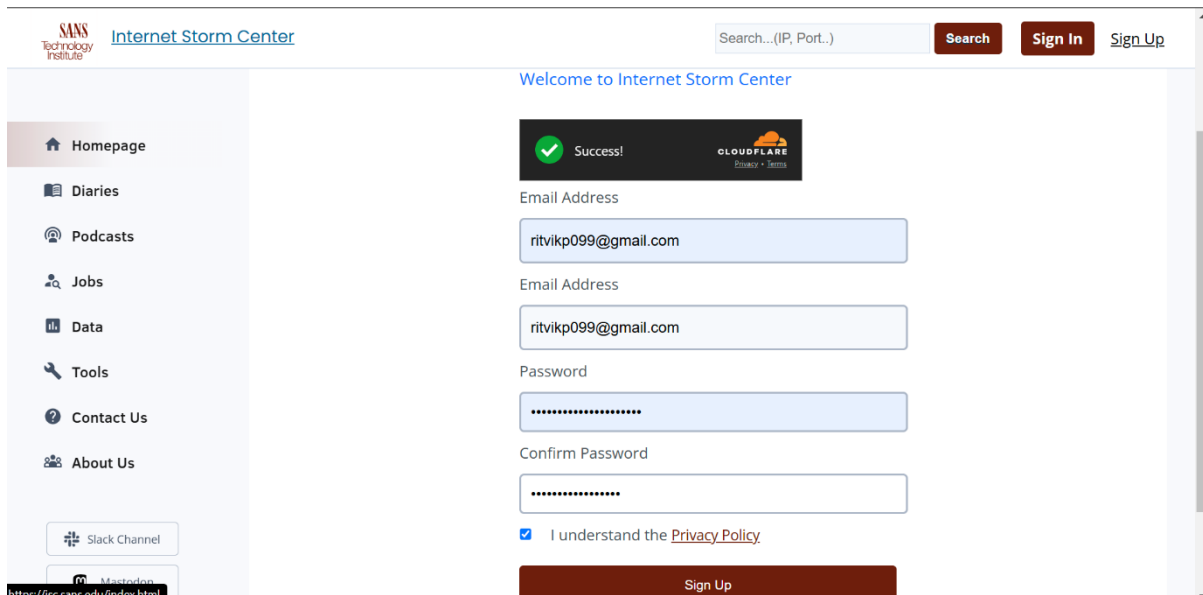


#### Verifying API Key:

The API key was pasted into the honeypot setup, linking the Raspberry Pi to the online DShield account. The system verified the API key and confirmed the successful connection.



After that it asks us if we want to update DShield automatically or manually we click on automatic. After that it asks us for an Email account and an API Key. So we go ahead and create an online account on DShield with the Email address – [ritvikp099@gmail.com](mailto:ritvikp099@gmail.com) and password – IndiaEngland2024.





After the sign up we can see all the information on Myaccount page about the API key and User ID.

## My Account

### Account Information

User ID #:3000056714

API Key: **b0e8cb7dfa0dbf2bc5506c262fcb53fd235a4d34**

(copy/paste to avoid typos. The key may not be visible right after the account is created)

**Reset Key (only use if key is compromised or empty)**

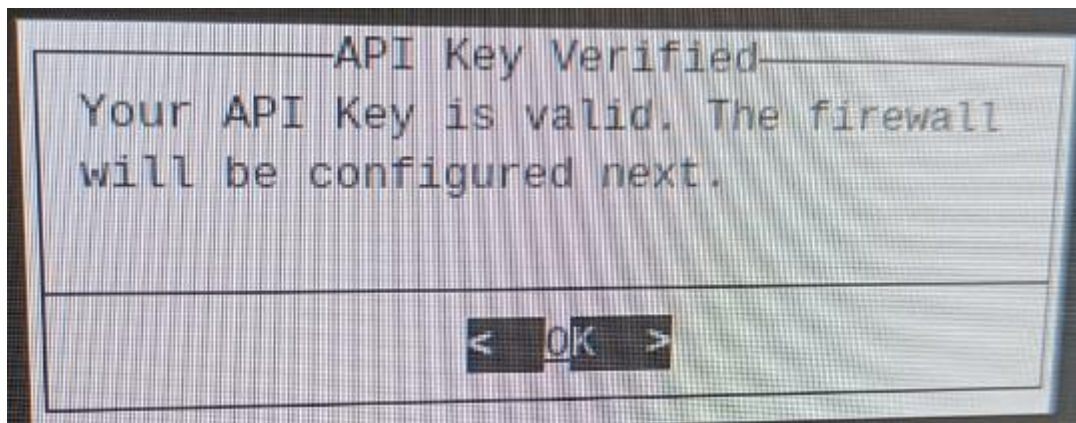
Account Status:**VERIFIED**

Two-Factor Authentication:**Disabled**

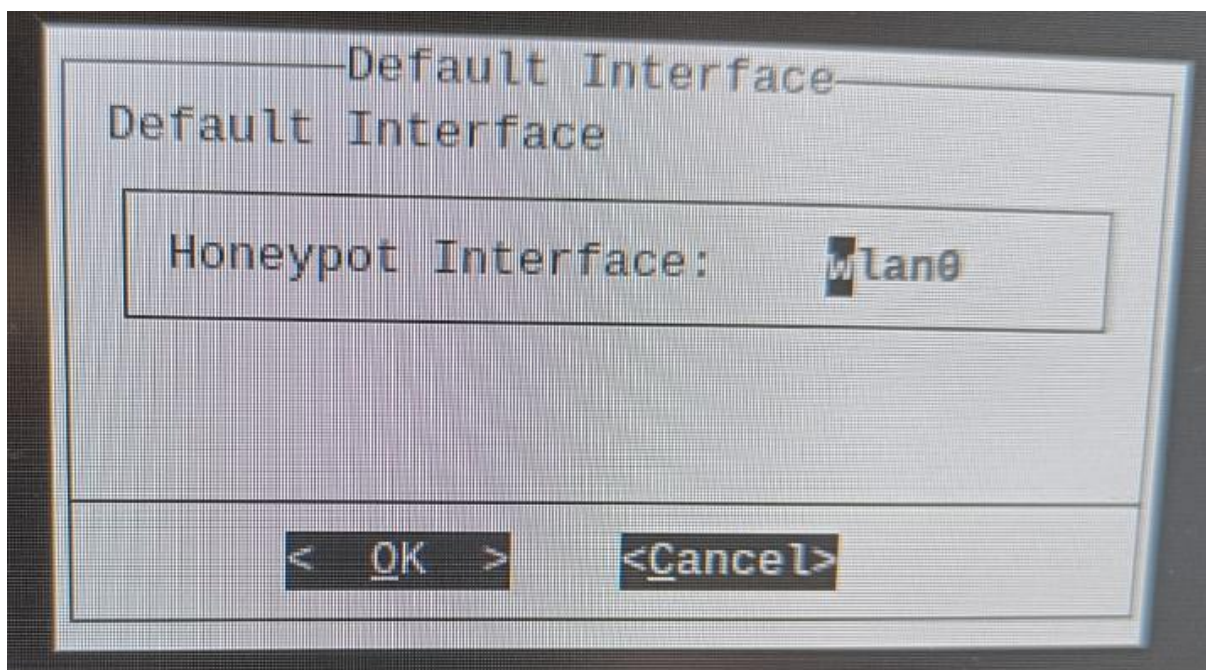
Recovery Phone:**Disabled**

U2F FIDO Token:**Disabled**

We copy and paste this API key into the honeypot API key. Then enter the email address on DShield to link our account with the honeypot on raspberry pi.

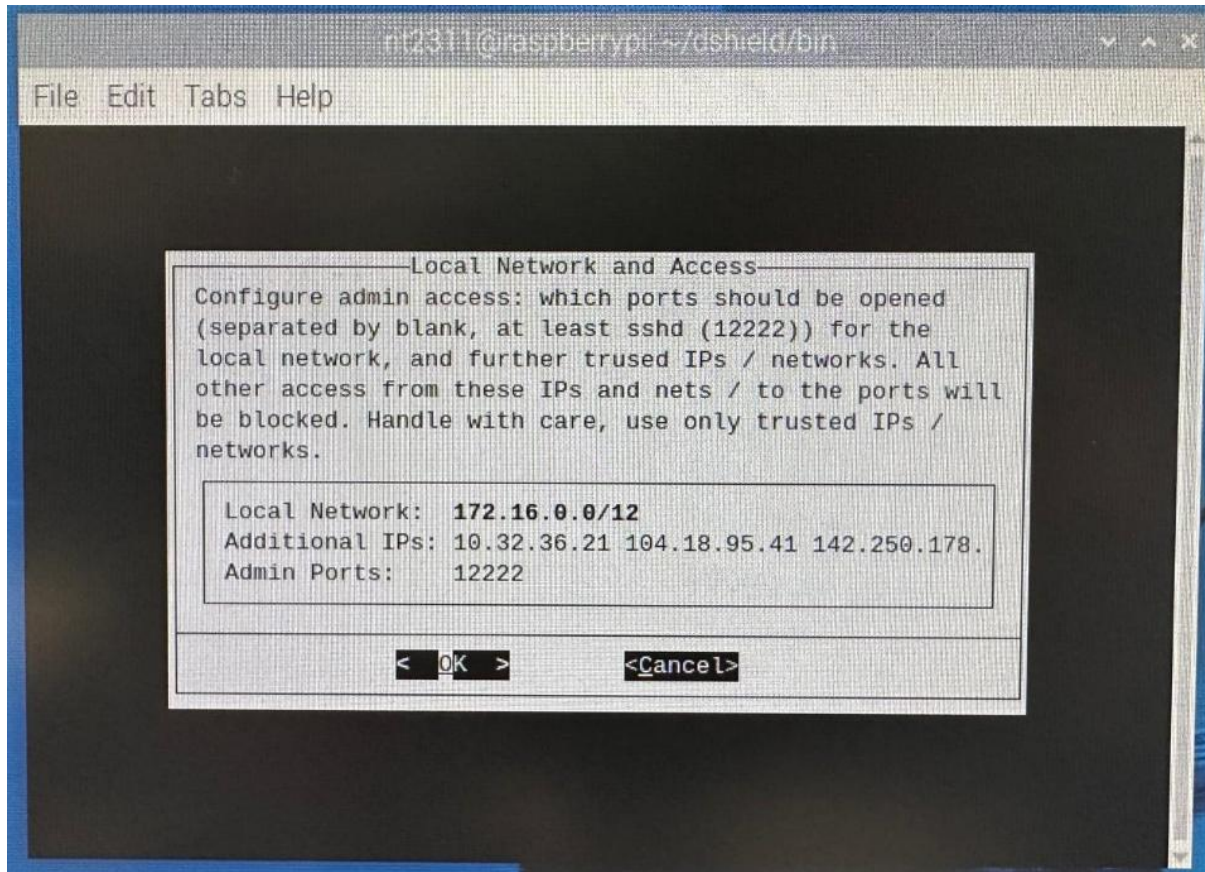


We get a message saying that our API key is valid and it has been verified and the firewall is ready to be configured.



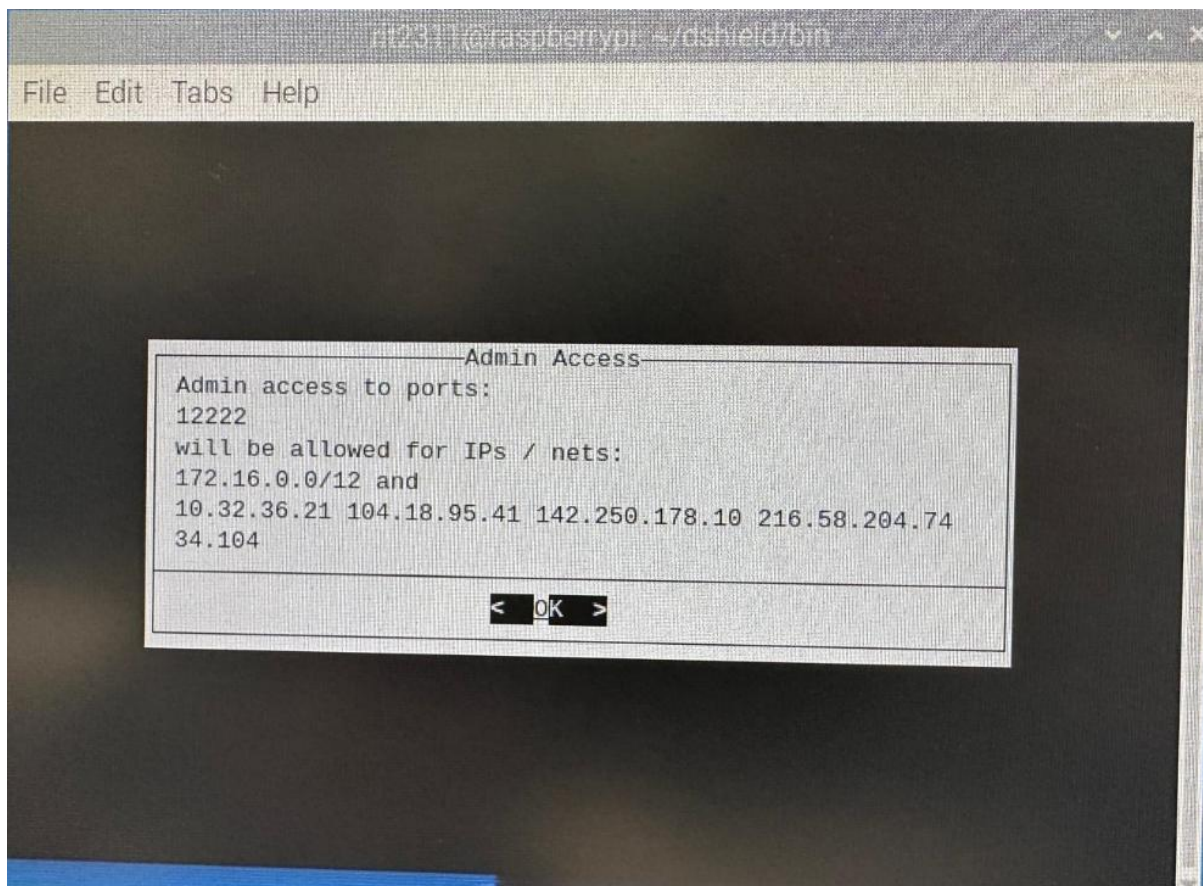
Then it asks us for the Default Interface we want to implement for the honeypot. We go ahead and select wlan0 as it is a wireless connection.

After that it asks us for the configuration of the administrator access and which ports should be opened in the Local network.

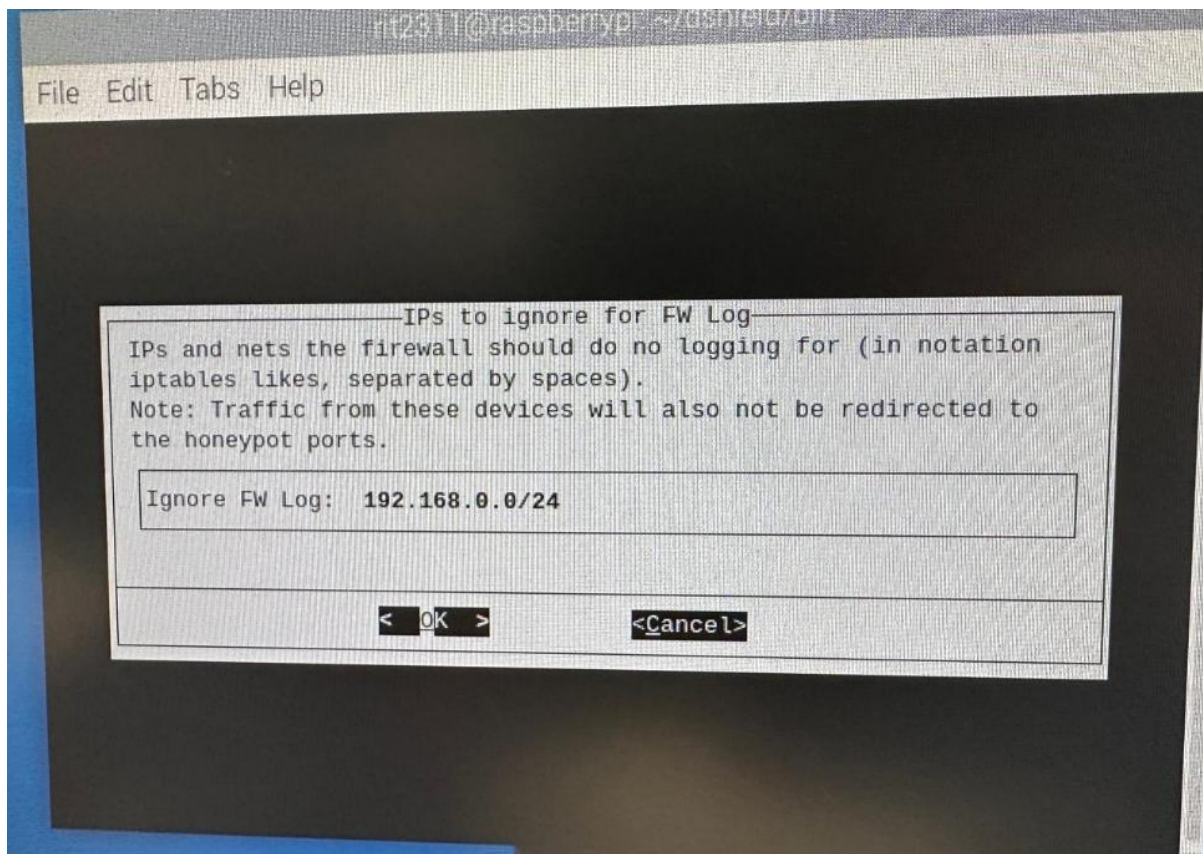


Next it gives us the the ports and the IP addresses the admin is able to access.



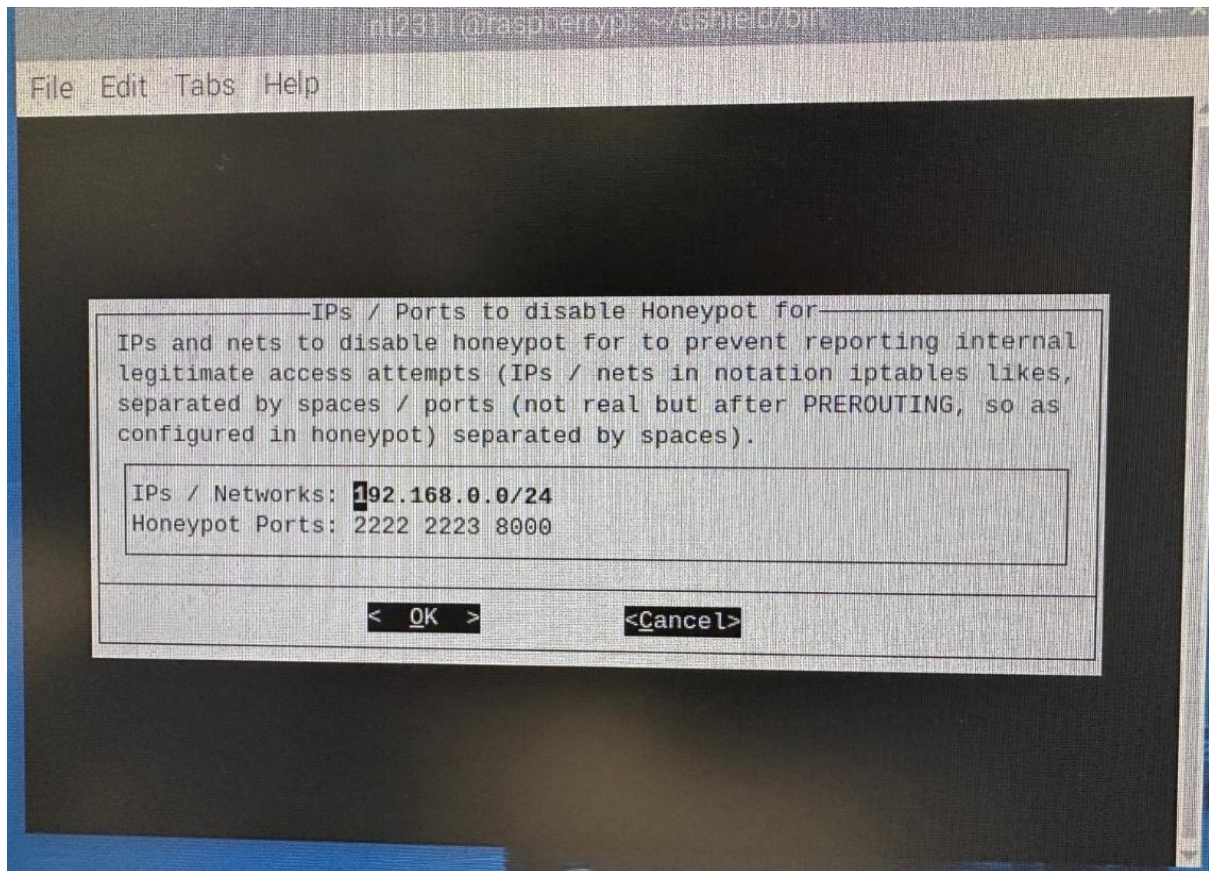


Then we setup an IP to ignore forward logs on that IP where it is not going to log any activity from.



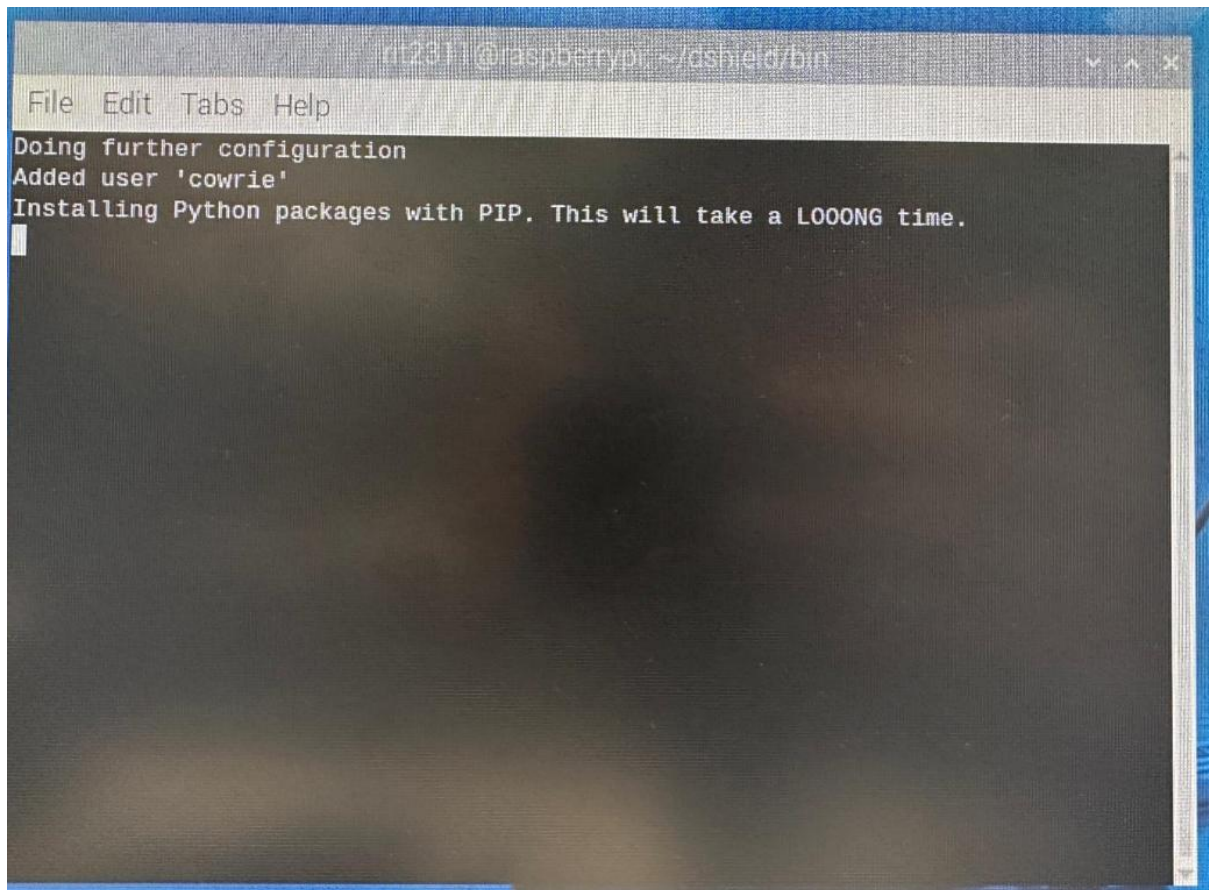
We change the IP to our internal network.

Then we choose the ports to disable any access attempts in the honeypot and the IP address.



After that it does further configuration.

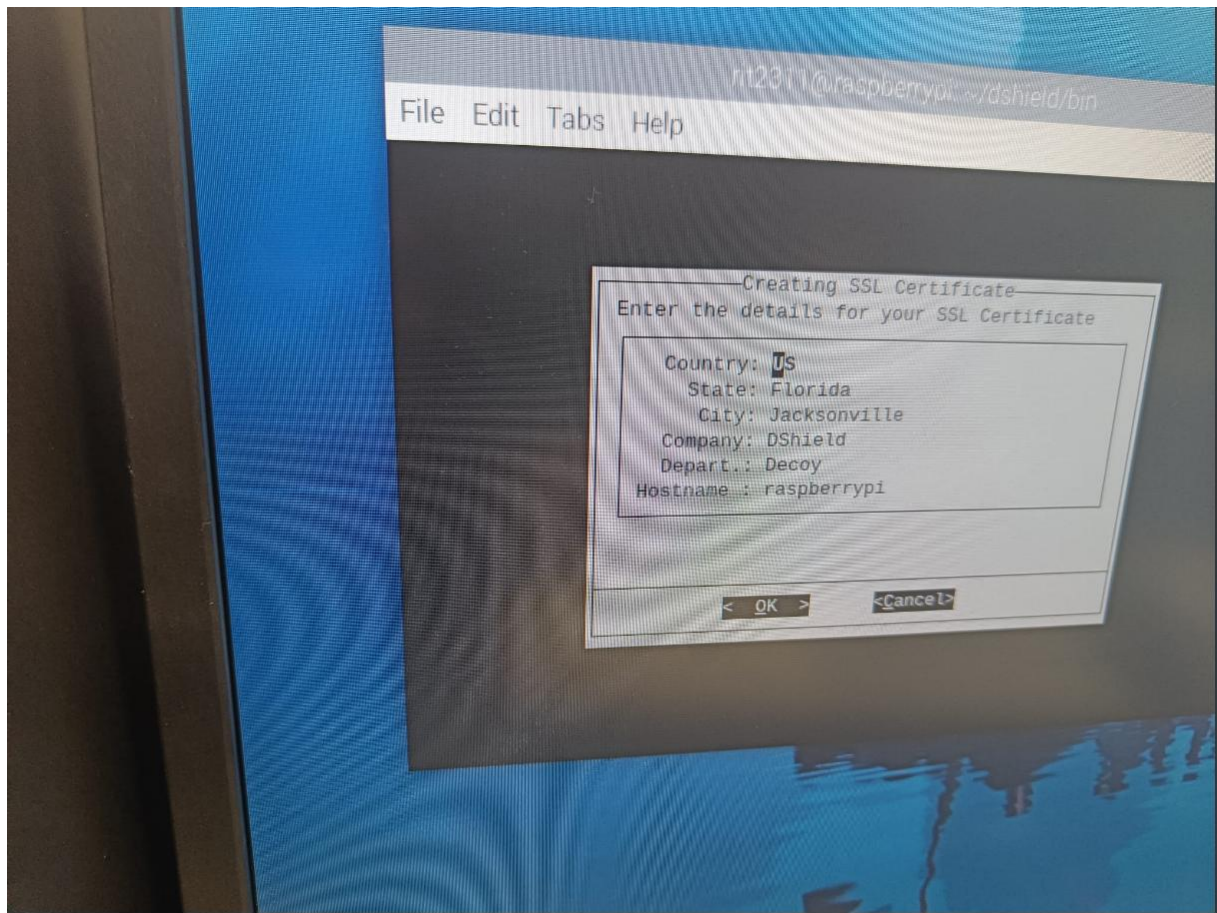




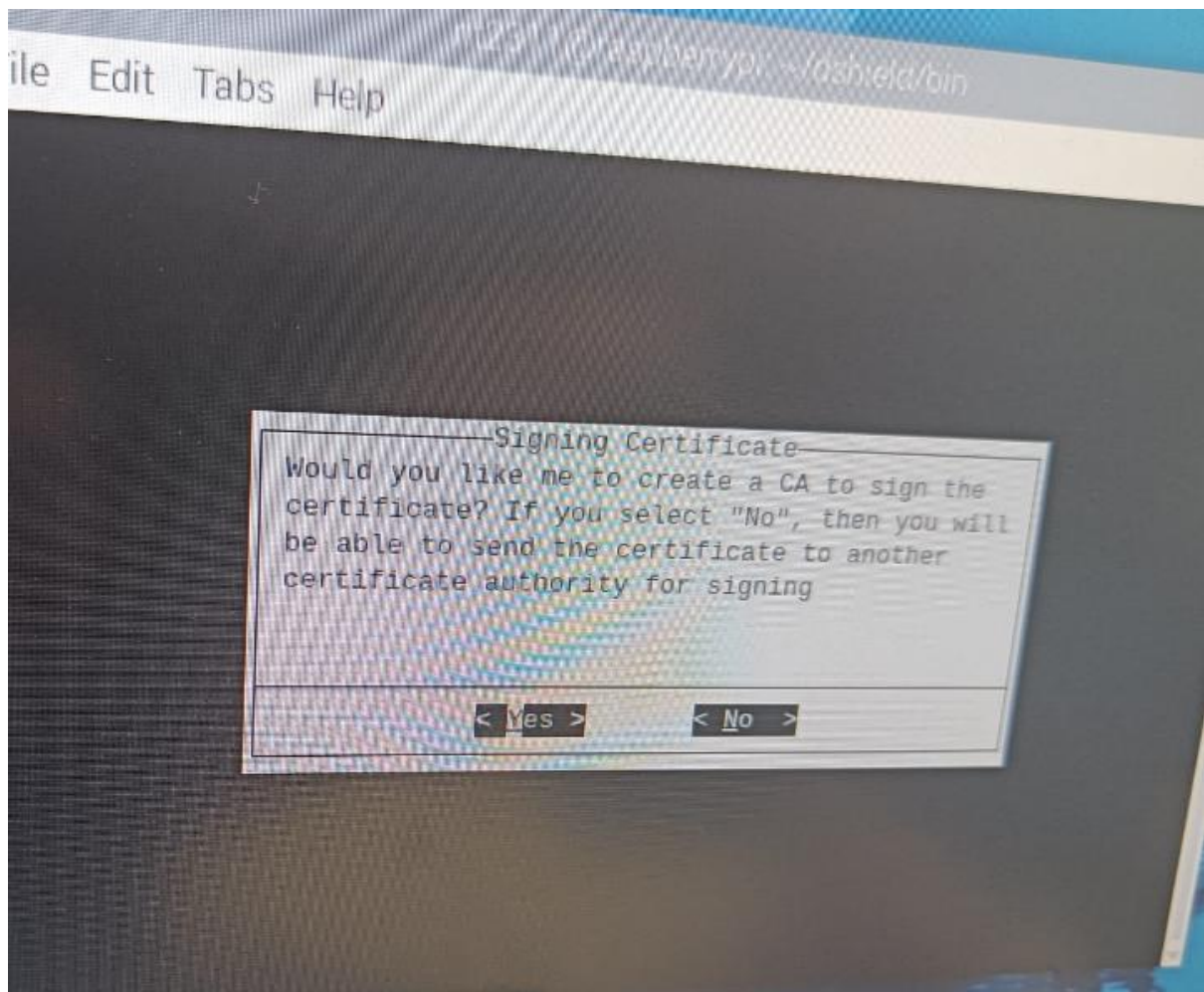
```
root@raspberrypi: ~/dsshield/bin
File Edit Tabs Help
Doing further configuration
Added user 'cowrie'
Installing Python packages with PIP. This will take a LOOONG time.
```

Then it asks us for the SSL certificate we can set it to whatever we want its just a decoy attackers.





After that we sign the SSL certificate.



That's all the configuration done for the honeypot after that we reboot our raspberry pi.

```
File Edit Tabs Help      rit2311@raspberrypi: ~/dshield/bin
Done.

Please reboot your Pi now.

For feedback, please e-mail jullrich@sans.edu or file a bug report on github
Please include a sanitized version of /etc/dshield.ini in bug reports
as well as a very carefully sanitized version of the installation log
(/srv/log/install_2024-11-21_113458.log).

IMPORTANT: after rebooting, the Pi's ssh server will listen on port 12222
           connect using ssh -p 12222 rit2311@172.20.32.32

### Thank you for supporting the ISC and dshield! ###

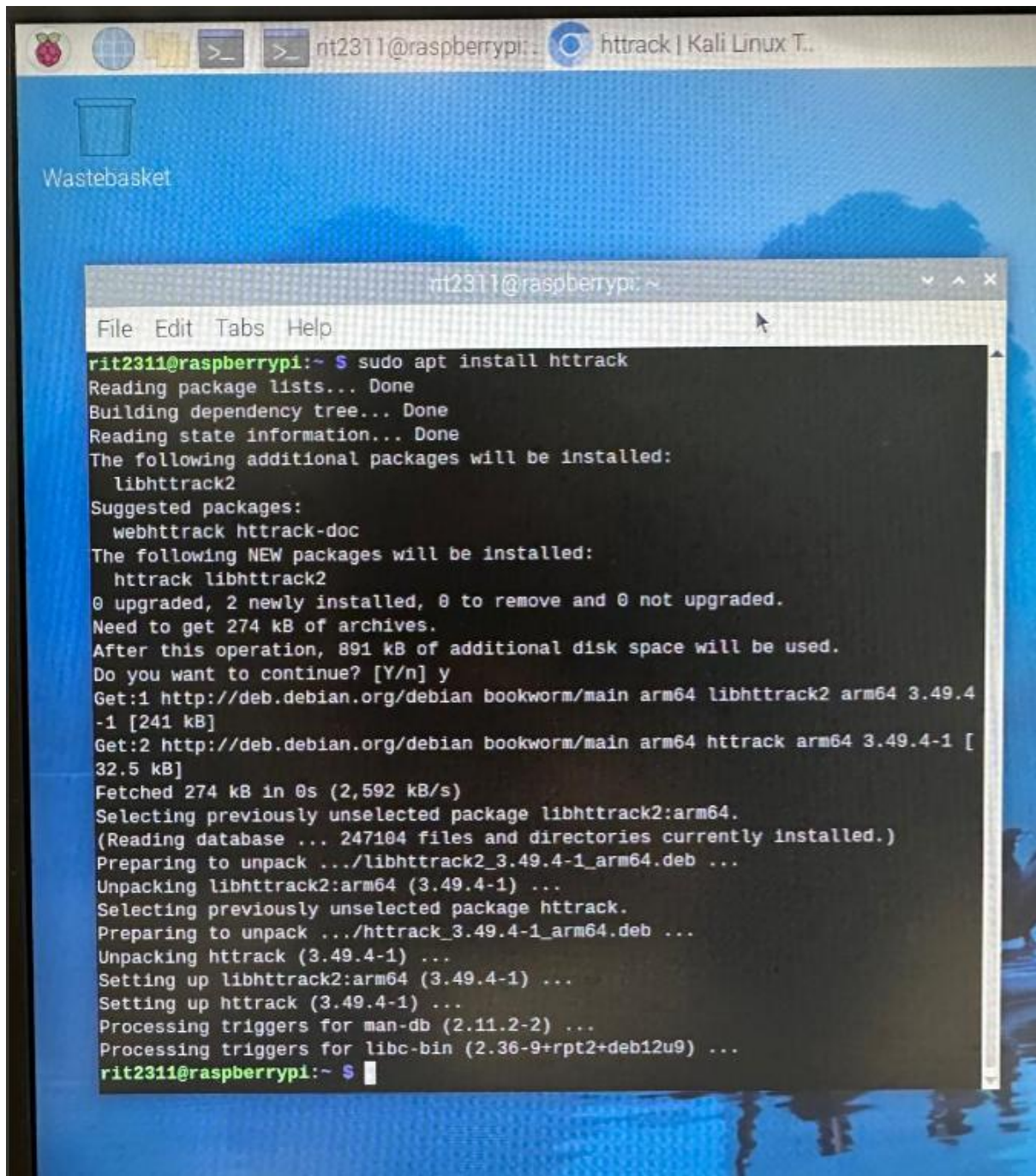
To check if all is working right:
  Run the script 'status.sh' (but reboot first!)
  or check https://isc.sans.edu/myreports.html (after logging in)

for help, check our slack channel: https://isc.sans.edu/slack

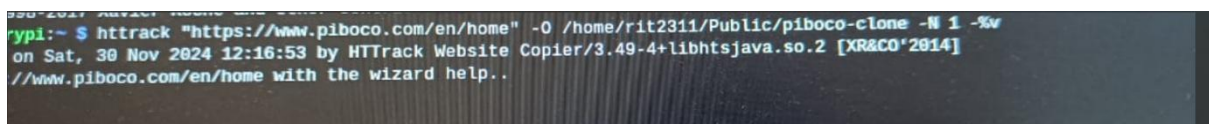
In case you are low in disk space, run /srv/dshield/cleanup.sh
This will delete some backups and logs
Log: /srv/log/install_2024-11-21_113458.log
rit2311@raspberrypi: ~/dshield/bin $
```

After this we need to install htrack to clone a website.





Then I used the command below to clone the front page of a website known as piboco.



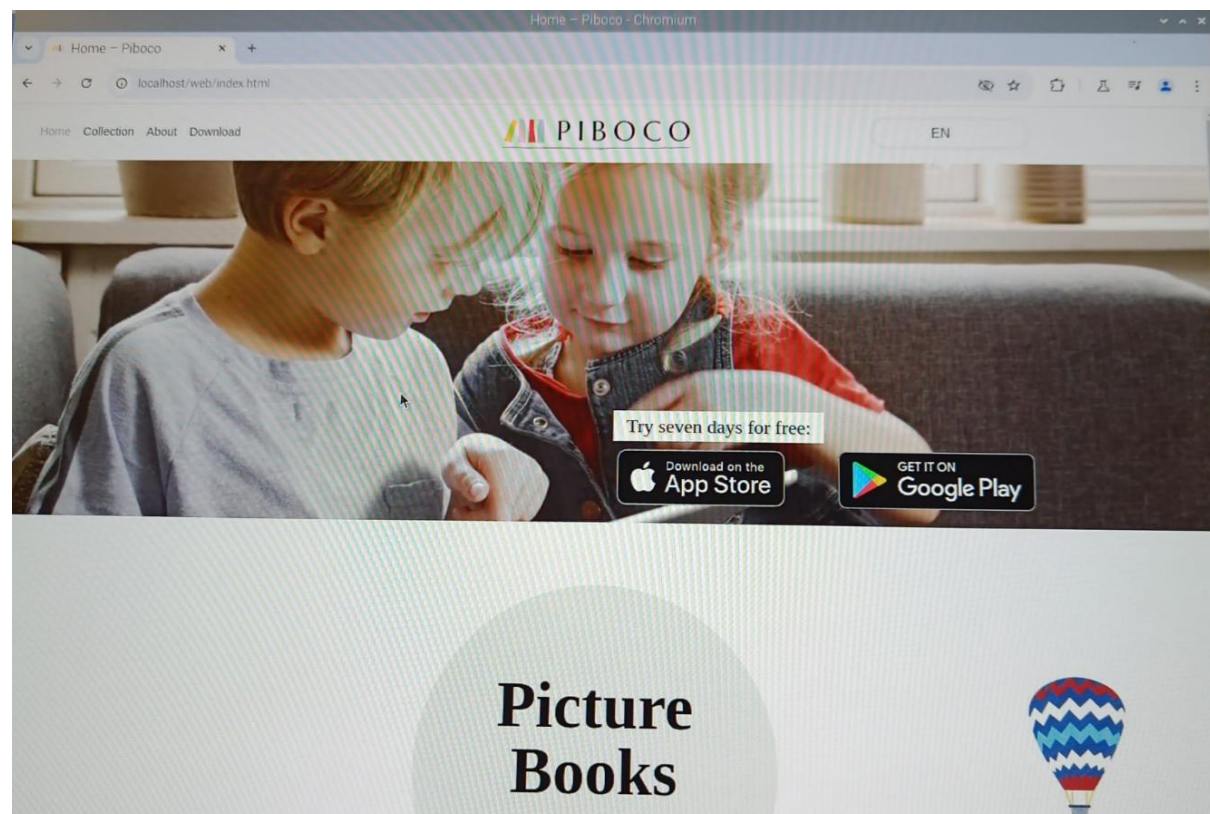
After this I used the command `ifconfig` which displays all the IP addresses on different networks. And I used the `wlan0` one as that is the one we setup earlier.

```
rit2311@raspberrypi:~$ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether e4:5f:01:67:ae:10 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 34 bytes 2902 (2.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 34 bytes 2902 (2.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

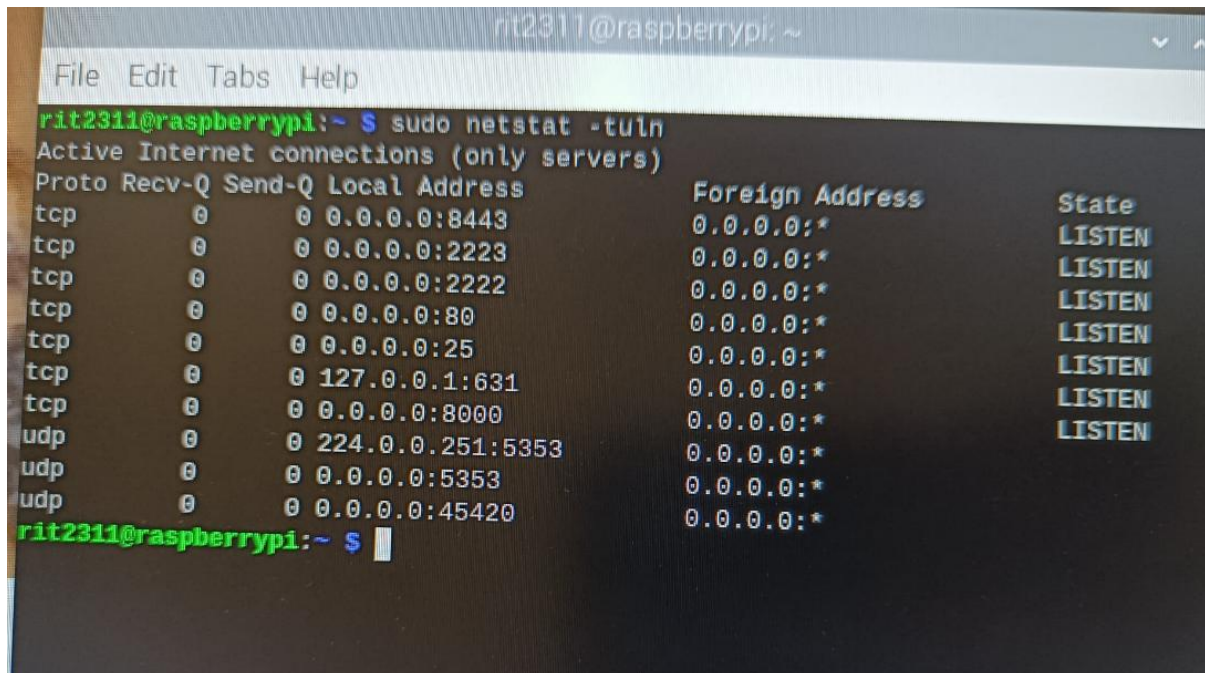
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.67 netmask 255.255.255.0 broadcast 192.168.0.255
    ether e4:5f:01:67:ae:11 txqueuelen 1000 (Ethernet)
    RX packets 5934 bytes 2816826 (2.6 MiB)
    RX errors 0 dropped 12 overruns 0 frame 0
    TX packets 1924 bytes 225455 (220.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

I used <http://192.168.0.67> to access the webpage on the localhost. Below you can see the webpage interface.



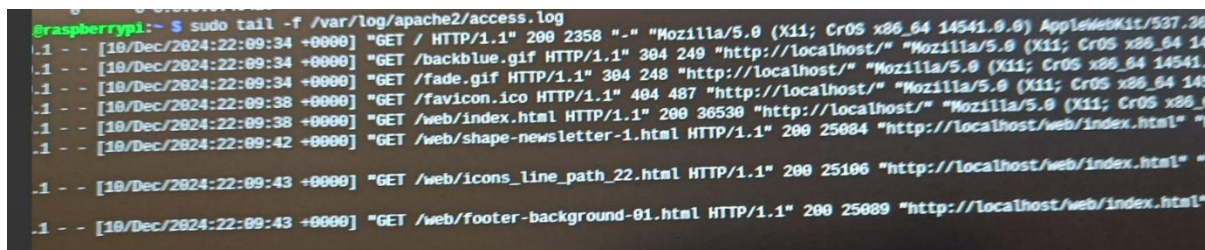


After that we check if apache server is listening on port 80.



```
rit2311@raspberrypi:~  
File Edit Tabs Help  
rit2311@raspberrypi:~ $ sudo netstat -tuln  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State  
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN  
tcp        0      0 0.0.0.0:2223            0.0.0.0:*               LISTEN  
tcp        0      0 0.0.0.0:2222            0.0.0.0:*               LISTEN  
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN  
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN  
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN  
tcp        0      0 0.0.0.0:8000            0.0.0.0:*               LISTEN  
udp        0      0 224.0.0.251:5353       0.0.0.0:*               LISTEN  
udp        0      0 0.0.0.0:5353           0.0.0.0:*               LISTEN  
udp        0      0 0.0.0.0:45420          0.0.0.0:*               LISTEN  
rit2311@raspberrypi:~ $
```

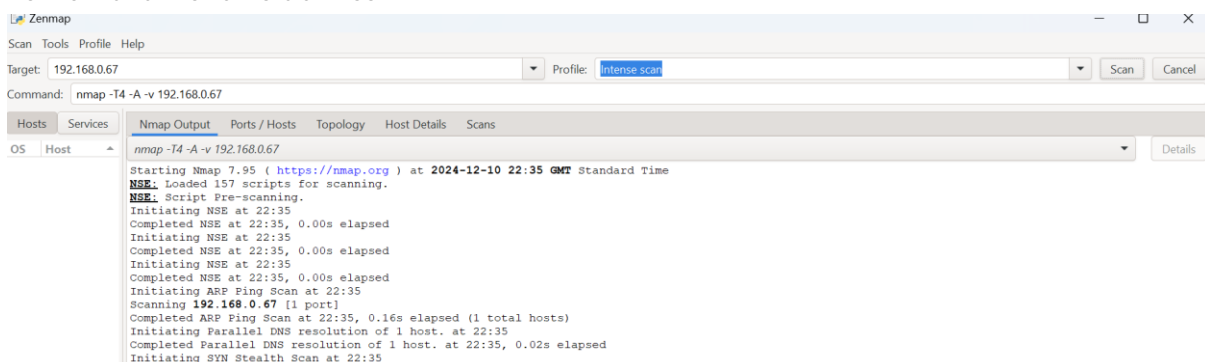
As we can see its Listening on port 80 so we don't need to change anything. Then we use tail command to check if its listening to our request. And as we can see its listening as the echo reply is 200 which means OK.



```
@raspberrypi:~ $ sudo tail -f /var/log/apache2/access.log  
1.1 - - [10/Dec/2024:22:09:34 +0000] "GET / HTTP/1.1" 200 2358 "-" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0) AppleWebKit/537.36  
1.1 - - [10/Dec/2024:22:09:34 +0000] "GET /backblue.gif HTTP/1.1" 304 249 "http://localhost/" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)  
1.1 - - [10/Dec/2024:22:09:34 +0000] "GET /fade.gif HTTP/1.1" 304 248 "http://localhost/" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)  
1.1 - - [10/Dec/2024:22:09:38 +0000] "GET /favicon.ico HTTP/1.1" 404 487 "http://localhost/" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)  
1.1 - - [10/Dec/2024:22:09:38 +0000] "GET /web/index.html HTTP/1.1" 200 36530 "http://localhost/" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)  
1.1 - - [10/Dec/2024:22:09:42 +0000] "GET /web/shape-newsletter-1.html HTTP/1.1" 200 25084 "http://localhost/web/index.html" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)  
1.1 - - [10/Dec/2024:22:09:43 +0000] "GET /web/icons_line_path_22.html HTTP/1.1" 200 25106 "http://localhost/web/index.html" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)  
1.1 - - [10/Dec/2024:22:09:43 +0000] "GET /web/footer-background-01.html HTTP/1.1" 200 25089 "http://localhost/web/index.html" "Mozilla/5.0 (X11; CrOS x86_64 14541.0.0)"
```

## RESOURCE MONITORING

After this I did the attack on the raspberry pi using Nmap. Nmap is a tool that can be used to scan ports on a device and to do an attack. It can be used to detect what devices are running on a network and the vulnerabilities.



```
OS Host  
Nmap Output  
nmap -T4 -A -v 192.168.0.67  
Starting Nmap 7.95 ( https://nmap.org ) at 2024-12-10 22:35 GMT Standard Time  
NSE: Loaded 157 scripts for scanning.  
NSE: Script Pre-scanning.  
Initiating NSE at 22:35  
Completed NSE at 22:35, 0.00s elapsed  
Initiating NSE at 22:35  
Completed NSE at 22:35, 0.00s elapsed  
Initiating NSE at 22:35  
Completed NSE at 22:35, 0.00s elapsed  
Initiating ARP Ping Scan at 22:35  
Scanning 192.168.0.67 [1 port]  
Completed ARP Ping Scan at 22:35, 0.16s elapsed (1 total hosts)  
Initiating Parallel DNS resolution of 1 host. at 22:35  
Completed Parallel DNS resolution of 1 host. at 22:35, 0.02s elapsed  
Initiating SYN Stealth Scan at 22:35
```



And in the honeypot we can access monitor the logs on the raspberry pi using the following commands.

[illegible]

## PERFORMANCE MONITORING USING HTOP

Monitoring the performance of a Raspberry Pi is essential to understand its behavior under various workloads and ensure optimal operation. For this purpose, I used **Htop**, a widely used system monitoring tool for Unix-based systems.

## About Htop

**Htop** is an interactive system-monitoring tool that provides real-time information about system resources such as CPU usage, memory consumption, tasks, and processes. Unlike the standard `top` command, Htop offers an intuitive interface with:

- Color-coded bars to represent resource utilization.
- Easy navigation and filtering for monitoring individual processes.
- A detailed breakdown of tasks and threads running in the system.

It is especially useful for understanding the Raspberry Pi's performance under varying workloads and identifying any potential bottlenecks.

---

### System State Under Normal Load

Here are the observed statistics when the Raspberry Pi was operating under a normal load, as captured using Htop:

#### 1. Memory Utilization:

- **Used Memory:** 486 MB out of the total available 3.70 GB.
- This indicates the system was consuming a modest amount of memory, typical for idle or low-intensity workloads.

#### 2. CPU Frequency and Load Average:

- **Frequency:** The CPU was operating at 600MHz, which is relatively low. This is an indication of the CPU scaling down its frequency to conserve power when the workload is light.
- **Load Average:** The load average was recorded at 0.54. This value represents the average number of active processes in the system over the last minute. A load average below 1 indicates the system has more than enough CPU capacity to handle its current tasks.

#### 3. Tasks and Threads:

- **Total Tasks:** 82 tasks were active in the system.
- **Threads:** These tasks included 183 threads and 138 kilo threads, representing the individual execution paths within the tasks.
- **Running Tasks:** 2 tasks were actively running, with the rest in a waiting or idle state.

#### 4. Swap Usage:

- The swap memory usage was 0 bytes out of the total available 200 MB.

- This shows the system was not under memory pressure, as swap is typically used only when the physical memory is full.

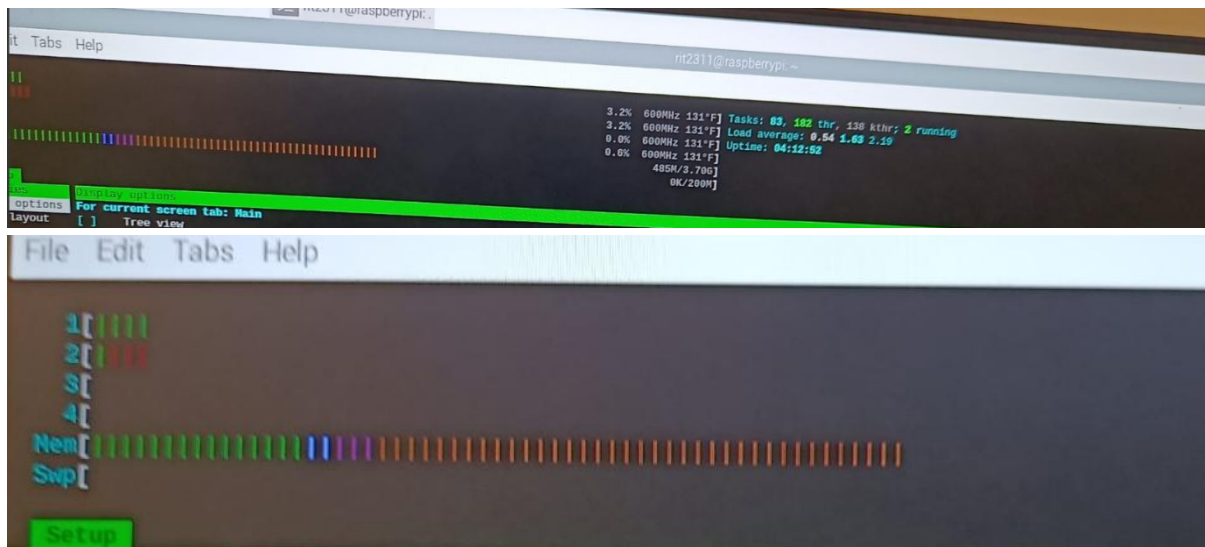
## 5. Temperature:

- The system's temperature was 131°F (approximately 55°C), which is well within safe operational limits.
- The low temperature is consistent with the light workload and lower CPU frequency, as less power consumption leads to reduced heat generation.

---

## Interpretation of Results

- The Raspberry Pi was operating efficiently under a normal load, with low memory usage, minimal CPU workload, and no reliance on swap memory.
- The temperature and CPU frequency remained in a range that ensures stable and energy-efficient operation.
- The system's load average of 0.54 demonstrates its ability to handle significantly more tasks without performance degradation.





```
nt2311@raspberrypi: ~$ top
top - 04:01:39
 29.0% 1800MHz 65°C] Tasks: 117, 363 thr, 136 kthr; 1 running
 26.5% 1800MHz 65°C] Load average: 2.29 2.17 1.98
 29.3% 1800MHz 65°C] Uptime: 04:01:39
 27.5% 1800MHz 65°C]
                    1.09G/3.78G
                    6K/200M

TIME+ Command
08.54 /sbin/init splash
09.60 /lib/systemd/systemd-journald
02.61 /lib/systemd/systemd-udev
00.00 (udev-worker)
00.15 /lib/systemd/systemd-networkd
00.23 /lib/systemd/systemd-timesyncd
00.00 /lib/systemd/systemd-timesyncd
01.77 /usr/libexec/accounts-daemon
01.43 /usr/libexec/accounts-daemon
00.08 /usr/libexec/accounts-daemon
01.25 avahi-daemon: running [raspberrypi.local]
00.00 avahi-daemon: chroot helper
00.10 /usr/libexec/bluetooth/bluetoothd
00.07 /usr/sbin/cron -f
02.98 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --
00.56 /usr/lib/polkit-1/polkitd --no-debug
00.00 /usr/lib/polkit-1/polkitd --no-debug
```

## PERFORMANCE EVALUATION

To evaluate the performance of the Raspberry Pi under attack conditions, I used **Nmap**, a network scanning tool, to simulate an attack on the honeypot. During the attack, I monitored system resources and performance metrics using **htop**, a real-time system monitoring utility. Below is a detailed breakdown of the results and observations.

### Simulated Attack using Nmap

The Nmap (Network Mapper) tool was used to perform a simulated attack on the Raspberry Pi. Nmap scans for open ports, services, and vulnerabilities on a target system, mimicking the reconnaissance phase of a cyberattack.

### Observations in htop

During the Nmap attack, I observed significant changes in system performance via htop. The key metrics were as follows:

#### 1. Task and Thread Utilization:

- The number of **tasks** increased by 34.
- The number of **threads** used reached 363.
- kthreads (kernel threads) increased to 136, indicating that the operating system kernel was actively managing more processes to handle the increased load.

#### 2. CPU Usage:

- The CPU usage increased by 29.6 percent, reflecting the computational demands of processing the incoming attack traffic and maintaining the honeypot's logging operations. The frequency of the CPU increased by 1200Mhz as the new frequency during the nmap scan was 1800Mhz.

### 3. **Memory Utilization:**

- The memory usage rose to 1.09 GB out of the available 3.70GB.
- The swap memory remained unused, indicating that the physical RAM was sufficient to handle the load without resorting to virtual memory.

### 4. **System Temperature:**

- The Raspberry Pi's temperature increased to 65 degree Celsius.
- The temperature spike was due to higher power consumption and the processor handling multiple tasks simultaneously.

### 5. **Running Tasks:**

- There was 1 running in the background, likely related to the DShield honeypot's logging activities and other system processes.

---

## **Analysis**

The performance metrics indicate that the Raspberry Pi managed to handle the simulated attack without significant degradation. Here are the key insights:

#### 1. **CPU and Memory Utilization:**

The increase in CPU and memory usage is expected during an attack, as the honeypot processes incoming traffic and logs details of the intrusion attempts. Despite the increased activity, the system's resource usage remained within acceptable limits.

#### 2. **Swap Memory:**

The swap memory was not utilized, suggesting that the available RAM (3.70 GB) was sufficient for the tasks. However, prolonged or more intense attacks might lead to memory saturation, requiring the use of swap memory.

#### 3. **Temperature:**

The temperature rise to 65 degree celcius indicates that the system was under a moderate workload. While this is within the operating range of the Raspberry Pi, sustained high temperatures can degrade performance and potentially damage hardware. It highlights the need for adequate cooling, such as using heat sinks or external fans.

---

## **STRESS TEST**

To evaluate the maximum capacity of the Raspberry Pi, I used a tool called stress. The stress tool generates a configurable workload that pushes the CPU, memory, and other system resources to their limits, providing insights into the system's breaking point. This test helps determine how the Raspberry Pi performs under heavy computational loads and its maximum sustainable capacity.

```
rit2311@raspberrypi: ~
File Edit Tabs Help
rit2311@raspberrypi:~ $ sudo apt install stress
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
stress is already the newest version (1.0.7-1).
The following packages were automatically installed and are no longer required:
linux-headers-6.6.31+rpt-common-rpi linux-headers-6.6.31+rpt-rpi-2712
linux-headers-6.6.31+rpt-rpi-v8 linux-image-6.6.31+rpt-rpi-2712
linux-image-6.6.31+rpt-rpi-v8 linux-kbuild-6.6.31+rpt
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
rit2311@raspberrypi:~ $ stress --cpu
stress: FAIL: [4045] (167) missing argument to option '--cpu'
rit2311@raspberrypi:~ $ stress --cpu2
stress: FAIL: [4047] (251) unrecognized option: --cpu2
rit2311@raspberrypi:~ $ stress --cpu 2
stress: info: [4050] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
/1
/1
/1
/1
/1
/1
```

```
Fine-grained Person.. rit2311@raspberrypi: ~
rit2311@raspberrypi: ~

||||| 76.2% Tasks: 104, 318 thr, 134 kthr; 4 running
||||| 100.0% Load average: 2.26 0.73 0.54
||||| 100.0% Uptime: 01:37:19
||||| 89.9%
||||| 1.58G/3.70G
||||| 0K/200M

E+ Command
0.00 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.00 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.00 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.13 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.03 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.01 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.25 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
0.36 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
03.67 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
00.00 /usr/lib/chromium/chromium --force-renderer-accessibility --enable-remote-extensions --show-compone
00.84 /usr/lib/chromium/chromium --type=utility --utility-sub-type=network.mojom.NetworkService --lang=en
00.03 /usr/lib/chromium/chromium --type=utility --utility-sub-type=network.mojom.NetworkService --lang=en
00.01 /usr/lib/chromium/chromium --type=utility --utility-sub-type=network.mojom.NetworkService --lang=en
```



## Results and Observations

During the stress test, the Raspberry Pi exhibited the following performance characteristics:

### 1. Tasks and Load Average:

- **Total Tasks:** Increased to 104, representing the total processes active during the test.
- **Running Tasks:** 4 tasks were actively running, corresponding to the 4 CPU threads utilized by the stress test.
- **Load Average:** Increased to 2.26, indicating a significant workload. A load average above 1 suggests that the system is running more tasks than it has CPU cores available.

### 2. CPU Utilization:

- The CPU was pushed to maximum capacity during the test, handling the computational workload generated by the stress tool.

### 3. Memory Utilization:

- 1.58GB of memory was utilized, representing a substantial portion of the Raspberry Pi's total available memory (3.70 GB).
- The memory usage spiked due to the 2 virtual memory workers each consuming 1 GB. The remaining memory was used for system operations and other background processes.

### 4. Temperature and Stability:

- The system temperature rose during the test, reflecting the increased power consumption and sustained CPU activity. While the system remained stable, prolonged operation under such conditions could lead to overheating or throttling.

---

## Analysis

The stress test revealed the Raspberry Pi's ability to handle high computational loads. However, certain limitations and potential bottlenecks were also observed:

### 1. CPU Bottleneck:

With a load average of 2.26, the CPU was nearing its maximum capacity. This indicates that the system was running more tasks than it could process simultaneously, leading to potential delays or task queuing.

### 2. Memory Usage:

The system used 1.58 GB of memory during the test, which is well below the total available memory. This suggests that memory was not a limiting factor for this specific workload. However, more intensive workloads with higher memory requirements could push the system closer to its limits.

### 3. Thermal Considerations:

During the stress test, the temperature of the Raspberry Pi rose significantly, increasing by nearly 20 degree Celsius. This temperature spike occurred as the system reached its maximum processing capacity, with the CPU working at full load. The increased thermal output was a direct result of the high-power consumption required to handle the intensive tasks, pushing the Raspberry Pi to its limits.

---

## TEAMWORK

Teamwork played a pivotal role in the successful completion of this project. My partner and I adopted a systematic approach, dividing responsibilities into two complementary areas to ensure all tasks were addressed efficiently.

My partner focused on configuring the Raspberry Pi, which involved setting up the operating system, enabling SSH access, and creating user accounts essential for the project. Meanwhile, I concentrated on testing the Raspberry Pi's capabilities and performance. My role included using tools like Nmap to simulate network attacks on the honeypot and conducting stress tests to evaluate the system's behavior under various load conditions.

Regular communication and coordination ensured smooth integration between the configuration and testing phases. While my partner worked to ensure the honeypot was operational and properly set up, I analyzed its effectiveness by examining logs and monitoring resource usage during simulated attacks. Together, we addressed challenges such as retrieving logs, resolving network connectivity issues, and interpreting system performance metrics.

Our collaboration not only improved the efficiency of the project but also enhanced our individual learning experiences. By complementing each other's skill sets, we ensured that no aspect of the project was overlooked. This team-oriented approach resulted in a fully functional honeypot and valuable performance data that could be used for further analysis and refinement.

## CONCLUSION

In conclusion, this coursework proved to be both challenging and rewarding. The initial phase of the project involved significant time and effort in evaluating various honeypot options and selecting the most suitable solution to meet our objectives. After deciding on the DShield honeypot, further challenges arose, including configuring the Raspberry Pi, troubleshooting SSH access issues, retrieving logs, and effectively utilizing performance monitoring tools.

Despite these challenges, the project provided me with valuable technical and problem-solving skills. Tools such as htop and stress enhanced my understanding of operating system processes, memory management, and CPU usage. Performing stress tests revealed how the Raspberry Pi

handles resource-intensive tasks and simulated attacks, giving me a practical appreciation for system performance monitoring.

Using Nmap also deepened my knowledge of network security and scanning tools, as I was able to simulate real-world attack scenarios and observe the honeypot's responses. Collecting and analyzing logs further provided critical insights into how the system managed network activity, allowing me to assess the effectiveness of our setup.

Looking forward, there are several areas for improvement. Enhancing the honeypot's security to withstand more sophisticated attacks and capturing more detailed logs could provide deeper insights into network behaviors and vulnerabilities. Refining my knowledge of advanced Linux commands and tools would improve my efficiency and capability in similar projects. Furthermore, incorporating more advanced techniques, such as brute-force attacks, could enhance the realism and depth of the testing process.

Overall, this project gave me a solid foundation in setting up and evaluating cybersecurity tools on embedded systems like the Raspberry Pi. It underscored the importance of thorough testing, effective resource monitoring, and continuous learning in achieving a successful outcome.

## REFERENCING

Fortinet (2024) *DShield v1.0.0*. Available at:

<https://docs.fortinet.com/document/fortisoar/1.0.0/dshield/1/dshield-v1-0-0#:~:text=DShield%20is%20a%20community%2Dbased,feeds%20from%20the%20DShield%20server> (Accessed: 10 December 2024).

SANS Internet Storm Center (2024) *Honeypot*. Available at: <https://isc.sans.edu/honeypot.html> (Accessed: 10 December 2024).

W3Schools (2024) *Git Introduction*. Available at:

[https://www.w3schools.com/git/git\\_intro.asp?remote=github](https://www.w3schools.com/git/git_intro.asp?remote=github) (Accessed: 10 December 2024).

SANS Internet Storm Center (2024) *How to submit logs*. Available at:

<https://isc.sans.edu/howto.html#:~:text=General%20Information%20On%20Submitting%20Logs,T>  
[op%20of%20page](https://isc.sans.edu/howto.html#:~:text=General%20Information%20On%20Submitting%20Logs,T) (Accessed: 10 December 2024).

Reynolds, R. (2023) *What is Nmap and How to Use It: A Tutorial for the Greatest Scanning Tool of All Time*. Available at: <https://www.freecodecamp.org/news/what-is-nmap-and-how-to-use-it-a-tutorial-for-the-greatest-scanning-tool-of-all-time/> (Accessed: 10 December 2024).

Monovm (2024) *What is htop and what does it do?*. Available at: <https://monovm.com/blog/what-is-htop-and-what-does-it-do/> (Accessed: 10 December 2024).

Wikipedia contributors (2024) *Raspberry Pi 4*. Available at:

[https://en.wikipedia.org/wiki/Raspberry\\_Pi\\_4](https://en.wikipedia.org/wiki/Raspberry_Pi_4) (Accessed: 10 December 2024).