Practical Network Defense

ASSIGNMENT 1

Firewall Configuration

Group number: **11**

Andrea Fede - 1942562
Chiara Iannicelli - 1957045
Pietro Colaguori - 1936709
Riccardo Tuzzolino - 1954109

Contents

1 Initial Brainstorming

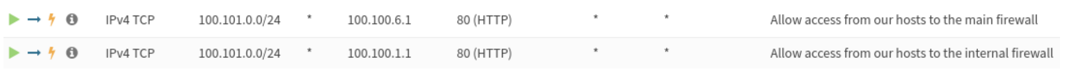
Our first consideration is based on the network architecture, which consists of two firewalls: the main one and the internal one. These two routers are connected via a point-to-point link. Each router is also connected to two separate subnetworks, resulting in a total of four subnetworks. The main router has a WAN interface that connects to the Internet.

Our goal is to configure the two firewalls according to the assigned security policy. To manage firewall rules efficiently, we decided to create aliases to group IPv4 and IPv6 addresses and subnetworks, thereby avoiding an excessive number of individual rules. When determining where to apply a firewall rule, we always chose to apply them as close to the source as possible and for inbound traffic, following the guidelines in the OPNsense documentation. Given that the default policy for each firewall interface is to drop incoming traffic and allow outgoing traffic, we adhered to the procedure of explicitly allowing only the necessary inbound traffic each time.

When rules apply to both the subnets connected to the main firewall and those connected to the internal firewall, we describe the firewall rules and the testing process as follows: we provide an example using one interface for the main firewall, one interface for the internal firewall, and one host in the respective subnets. This approach ensures clarity in demonstrating how the firewall rules were implemented and tested for proper functionality.

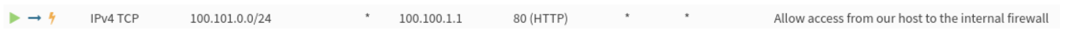
Finally, to easily access the firewalls' web interfaces from our hosts' web browsers, we allowed traffic from the subnet `100.101.0.0/24`, which includes our hosts' IP addresses, to the interfaces of both the main firewall and the internal firewall.

On the WAN interface of the main firewall:



▶ → ϟ ●	IPv4 TCP	100.101.0.0/24	*	100.100.6.1	80 (HTTP)	*	*	Allow access from our hosts to the main firewall
▶ → ϟ ●	IPv4 TCP	100.101.0.0/24	*	100.100.1.1	80 (HTTP)	*	*	Allow access from our hosts to the internal firewall

On the EXTERNAL interface of the internal firewall:



▶ → ϟ	IPv4 TCP	100.101.0.0/24	*	100.100.1.1	80 (HTTP)	*	*	Allow access from our host to the internal firewall

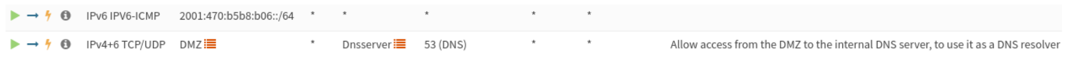
2 Evaluation, implementation and tests of the security policies

2.1 All the ACME hosts must use the internal DNS Server as a DNS resolver

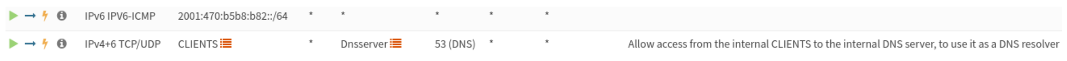
We need to insert two rules in all the interfaces of the main firewall (except WAN) and all the interfaces of the internal firewall (except SERVERS):

- one to allow IPv6-ICMP packets to travel from the respective subnet connected to the firewall interface towards any destination, to permit the packets required by the Neighbour Discovery Protocol (because the connection to the DNS server in our network uses its IPv6 address by defualt);

- another to allow the traffic from the respective subnet connected to the firewall interface towards the DNS server on port 53, of both TCP and UDP protocols.

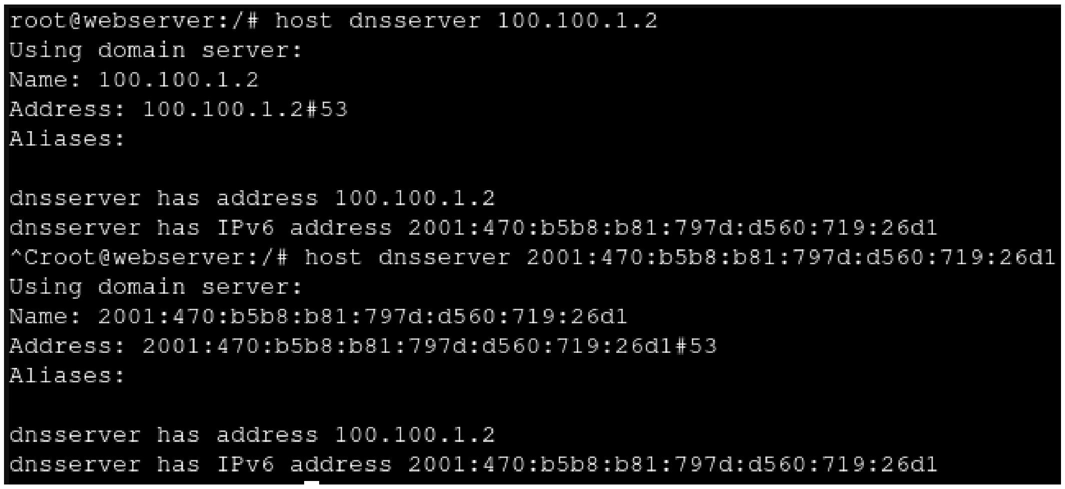


IPv6 IPV6-ICMP	2001:470:b5b8:b06::/64	*	*	*	*	*	
IPv4+6 TCP/UDP	DMZ	*	Dnsserver	53 (DNS)	*	*	Allow access from the DMZ to the internal DNS server, to use it as a DNS resolver



IPv6 IPV6-ICMP	2001:470:b5b8:b82::/64	*	*	*	*	*	
IPv4+6 TCP/UDP	CLIENTS	*	Dnsserver	53 (DNS)	*	*	Allow access from the internal CLIENTS to the internal DNS server, to use it as a DNS resolver

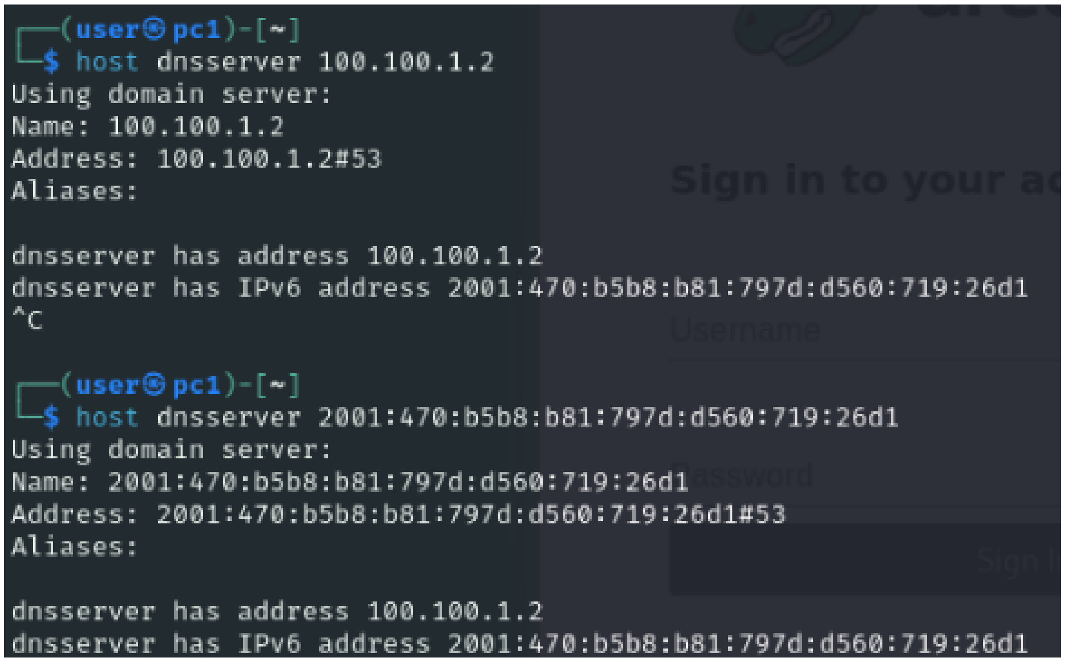
In order to check that everything is working properly for both IPv4 and IPv6, we executed the `host` command, used for DNS lookup operations, from the Webserver and the kali machines.



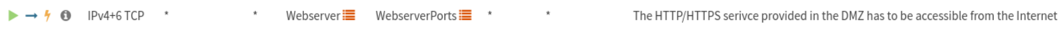
```
root@webserver:/# host dnsserver 100.100.1.2
Using domain server:
Name: 100.100.1.2
Address: 100.100.1.2#53
Aliases:

dnsserver has address 100.100.1.2
dnsserver has IPv6 address 2001:470:b5b8:b81:797d:d560:719:26d1
^Croot@webserver:/# host dnsserver 2001:470:b5b8:b81:797d:d560:719:26d1
Using domain server:
Name: 2001:470:b5b8:b81:797d:d560:719:26d1
Address: 2001:470:b5b8:b81:797d:d560:719:26d1#53
Aliases:

dnsserver has address 100.100.1.2
dnsserver has IPv6 address 2001:470:b5b8:b81:797d:d560:719:26d1
```

2.2 The HTTP/HTTPS service provided in the DMZ has to be accessible from the Internet

Our ACME network communicates with the "outside world" (the Internet) using the WAN interface of the main firewall, thus we need to create a new rule on such interface to allow in all the traffic that has as destination the web server in the DMZ on ports 80 (HTTP) and 443 (HTTPS) to go through.

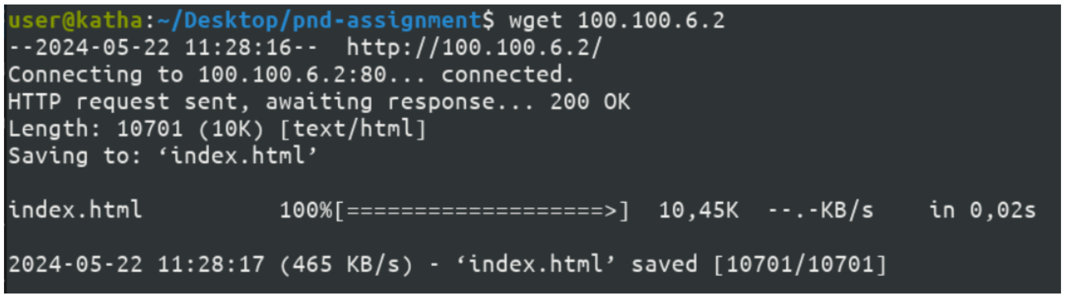


```
┌──(user㉿pc1)-[~]
└─$ host dnsserver 100.100.1.2
Using domain server:
Name: 100.100.1.2
Address: 100.100.1.2#53
Aliases:

dnsserver has address 100.100.1.2
dnsserver has IPv6 address 2001:470:b5b8:b81:797d:d560:719:26d1
^C

┌──(user㉿pc1)-[~]
└─$ host dnsserver 2001:470:b5b8:b81:797d:d560:719:26d1
Using domain server:
Name: 2001:470:b5b8:b81:797d:d560:719:26d1
Address: 2001:470:b5b8:b81:797d:d560:719:26d1#53
Aliases:

dnsserver has address 100.100.1.2
dnsserver has IPv6 address 2001:470:b5b8:b81:797d:d560:719:26d1
```



▶ → ϟ ℹ	IPv4+6 TCP	*	*	Webserver ☰	WebserverPorts ☰	*	*	The HTTP/HTTPS serivce provided in the DMZ has to be accessible from the Internet

To test this we can use the terminal of our virtual machine to perform `wget` on the Webserver.

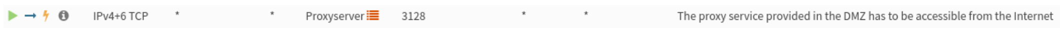


```
user@katha:~/Desktop/pnd-assignment$ wget 100.100.6.2
--2024-05-22 11:28:16--  http://100.100.6.2/
Connecting to 100.100.6.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10701 (10K) [text/html]
Saving to: ‘index.html’

index.html          100%[===================>]  10,45K  --.-KB/s    in 0,02s

2024-05-22 11:28:17 (465 KB/s) - ‘index.html’ saved [10701/10701]
```

We cannot perform the same test with IPv6 since we don't know the IPv6 route to connect to the ACME network from our host.

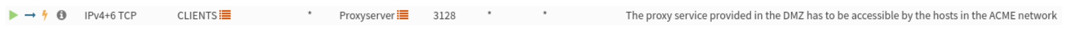
2.3 The proxy service provided in the DMZ has to be accessible by the hosts of the ACME network and from the Internet

Making the proxy service provided in the DMZ accessible from the Internet is analogous to the previous policy.



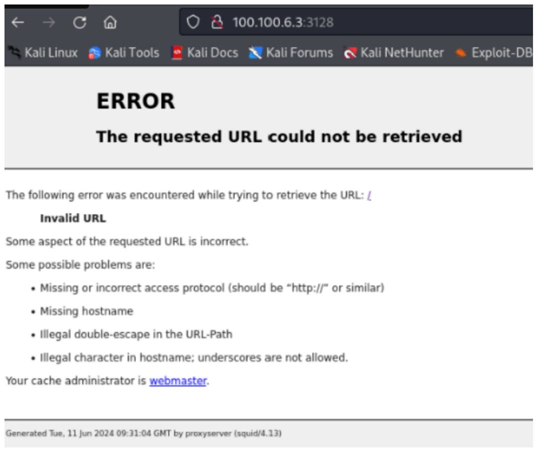
▶ → ϟ ℹ	IPv4+6 TCP	*	*	Proxyserver ☰	3128	*	*	The proxy service provided in the DMZ has to be accessible from the Internet

For all the hosts of the ACME network we need to create a rule for all the interfaces of the main firewall (except DMZ) and all the interfaces of the internal firewall (except INTERNAL) to allow traffic directed to the TCP port 3128 of the proxy server to go through.



```
IPv4+6 TCP   CLIENTS   *   Proxyserver   3128   *   *   The proxy service provided in the DMZ has to be accessible by the hosts in the ACME network
```

To test that everything is working correctly, we can try to reach the SQUID proxy from a web browser (e.g. Firefox) with the URL `http://100.100.6.3:3128`. We will see that the SQUID typical error is shown, but it is up and running and we can reach it. The same result is obtaining by using IPv6.



100.100.6.3:3128

ERROR

The requested URL could not be retrieved

The following error was encountered while trying to retrieve the URL: /

Invalid URL

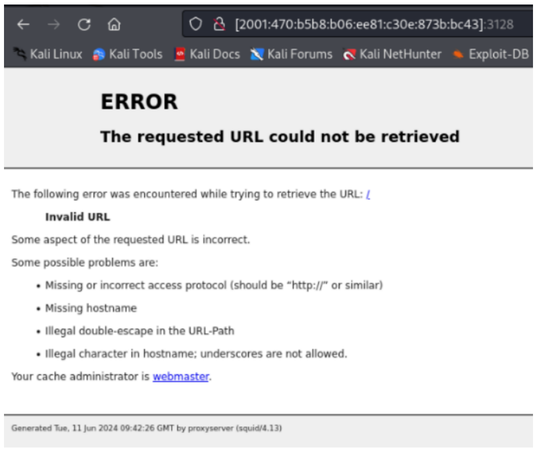
Some aspect of the requested URL is incorrect.

Some possible problems are:

- Missing or incorrect access protocol (should be "http://" or similar)
- Missing hostname
- Illegal double-escape in the URL-Path
- Illegal character in hostname; underscores are not allowed.

Your cache administrator is webmaster.

Generated Tue, 11 Jun 2024 09:31:04 GMT by proxyserver (squid/4.13)



[2001:470:b5b8:b06:ee81:c30e:873b:bc43]:3128

ERROR

The requested URL could not be retrieved

The following error was encountered while trying to retrieve the URL: /

Invalid URL

Some aspect of the requested URL is incorrect.

Some possible problems are:

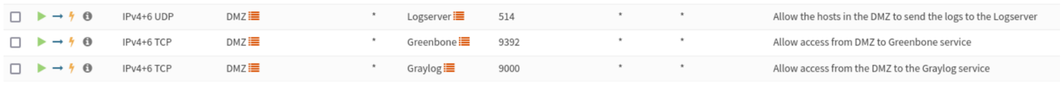
- Missing or incorrect access protocol (should be "http://" or similar)
- Missing hostname
- Illegal double-escape in the URL-Path
- Illegal character in hostname; underscores are not allowed.

Your cache administrator is webmaster.

Generated Tue, 11 Jun 2024 09:42:26 GMT by proxyserver (squid/4.13)

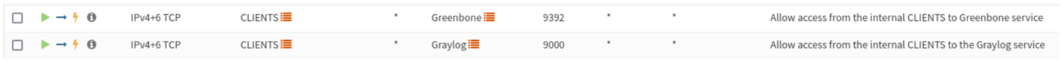
2.4 Besides the DNS resolver, the other services in the Internal server network must be accessible only to hosts of the Client and DMZ networks

Since the default policy for the OPNsense firewall is to drop packets, we don't need to do anything to deny access from the EXTERNAL CLIENTS subnet and the Internet to the SERVERS subnet. We need to explicitly allow instead the traffic coming from the CLIENTS and the DMZ, so that they can reach the following services provided in the SERVERS subnet: Greenbone on TCP port 9392, Graylog on TCP port 9000, and Syslog on UDP port 514.



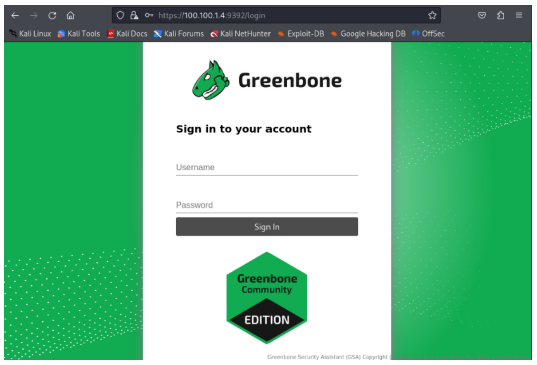
☐	▶ → ϟ ❶	IPv4+6 UDP	DMZ	*	Logserver	514	*	*	Allow the hosts in the DMZ to send the logs to the Logserver
☐	▶ → ϟ ❶	IPv4+6 TCP	DMZ	*	Greenbone	9392	*	*	Allow access from DMZ to Greenbone service
☐	▶ → ϟ ❶	IPv4+6 TCP	DMZ	*	Graylog	9000	*	*	Allow access from the DMZ to the Graylog service

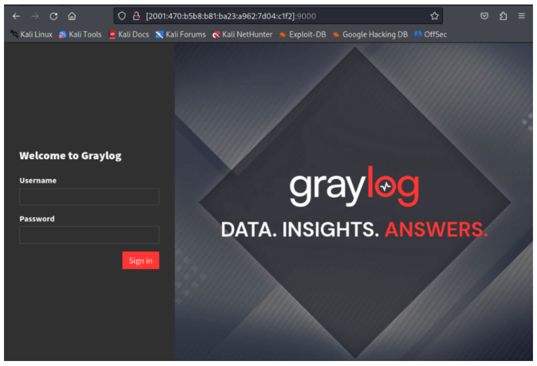
Since in 2.5 it is requested that the hosts in the CLIENTS network don't use the Syslog service, we didn't add this specific rule in the respective internal firewall interface.



☐	▶ → ϟ ❶	IPv4+6 TCP	CLIENTS	*	Greenbone	9392	*	*	Allow access from the internal CLIENTS to Greenbone service
☐	▶ → ϟ ❶	IPv4+6 TCP	CLIENTS	*	Graylog	9000	*	*	Allow access from the internal CLIENTS to the Graylog service

To test this, we can try using the Kali machine to access the services from their web interfaces, using both the IPv4 and IPv6 addresses. In 2.5 also Syslog will be tested.

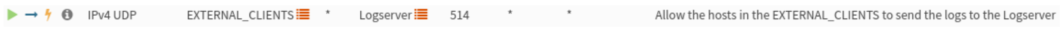




2.5 All the hosts (but the Client network hosts) have to use the syslog and the log collector services on the Log server (syslog) and Graylog server

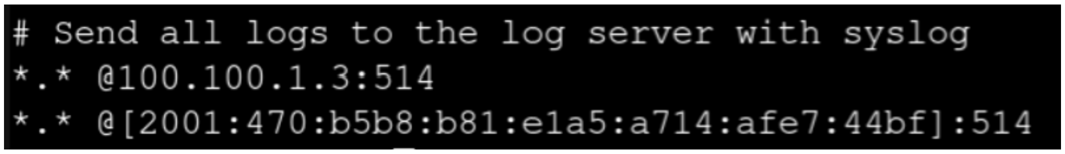
The desired outcome is that the hosts in all the subnets of the ACME network but the CLIENTS subnet send their logs to the Logserver (acting as a log collector) which is running the Syslog service, and then all the logs it receives are sent to the Graylog server for analysis.

In 2.4 we've implemented the rule in the DMZ interface of the main firewall to allow the UDP traffic towards the Logserver on port 514. The same can be done for the EXTERNAL CLIENTS interface.



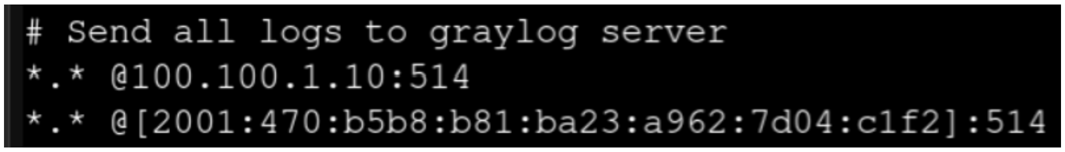
```
IPv4 UDP    EXTERNAL_CLIENTS  *    Logserver  514  *  *    Allow the hosts in the EXTERNAL_CLIENTS to send the logs to the Logserver
```

Now we need to configure the hosts in these subnets to actually send all the logs to the Logserver, using both IPv4 and IPv6 addresses. To do so, we properly configured the syslog service by editing the `/etc/rsyslog.conf` file:



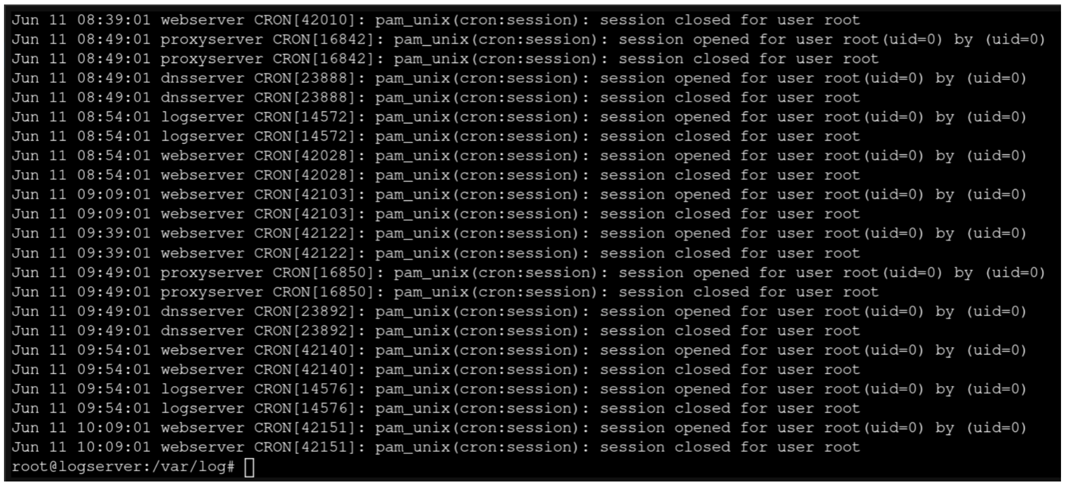
```
# Send all logs to the log server with syslog
*.* @100.100.1.3:514
*.* @[2001:470:b5b8:b81:e1a5:a714:afe7:44bf]:514
```

Finally, we have to configure the Logserver so that it sends all the collected logs to the Graylog server.



```
# Send all logs to graylog server
*.* @100.100.1.10:514
*.* @[2001:470:b5b8:b81:ba23:a962:7d04:c1f2]:514
```

In order to test if this process is working properly, we can check if the logs are effectively being collected on by the Logserver. As an example, we can see that the `/var/log/auth.log` file contains logs coming from the other ACME hosts.

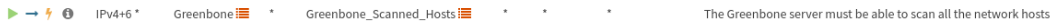


```
Jun 11 08:39:01 webserver CRON[42010]: pam_unix(cron:session): session closed for user root
Jun 11 08:49:01 proxyserver CRON[16842]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 08:49:01 proxyserver CRON[16842]: pam_unix(cron:session): session closed for user root
Jun 11 08:49:01 dnsserver CRON[23888]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 08:49:01 dnsserver CRON[23888]: pam_unix(cron:session): session closed for user root
Jun 11 08:54:01 logserver CRON[14572]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 08:54:01 logserver CRON[14572]: pam_unix(cron:session): session closed for user root
Jun 11 08:54:01 webserver CRON[42028]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 08:54:01 webserver CRON[42028]: pam_unix(cron:session): session closed for user root
Jun 11 09:09:01 webserver CRON[42103]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 09:09:01 webserver CRON[42103]: pam_unix(cron:session): session closed for user root
Jun 11 09:39:01 webserver CRON[42122]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 09:39:01 webserver CRON[42122]: pam_unix(cron:session): session closed for user root
Jun 11 09:49:01 proxyserver CRON[16850]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 09:49:01 proxyserver CRON[16850]: pam_unix(cron:session): session closed for user root
Jun 11 09:49:01 dnsserver CRON[23892]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 09:49:01 dnsserver CRON[23892]: pam_unix(cron:session): session closed for user root
Jun 11 09:54:01 webserver CRON[42140]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 09:54:01 webserver CRON[42140]: pam_unix(cron:session): session closed for user root
Jun 11 09:54:01 logserver CRON[14576]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 09:54:01 logserver CRON[14576]: pam_unix(cron:session): session closed for user root
Jun 11 10:09:01 webserver CRON[42151]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jun 11 10:09:01 webserver CRON[42151]: pam_unix(cron:session): session closed for user root
root@logserver:/var/log#
```

2.6 The Greenbone server must be able to scan all the network hosts

Firstly, we need to configure the Greenbone vulnerability scanner, which is in the SERVERS subnet (the configuration process will be detailed in the report of the third assignment). Secondly, we need to implement firewall rules in the SERVERS interface of the internal firewall and the INTERNAL interface of the main firewall to allow access from the Greenbone server to every host in the AMCE network (any protocol, any port).

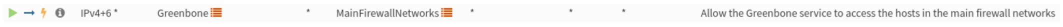
SERVERS interface of the internal firewall:



IPv4+6 *	Greenbone	*	Greenbone_Scanned_Hosts	*	*	*	The Greenbone server must be able to scan all the network hosts

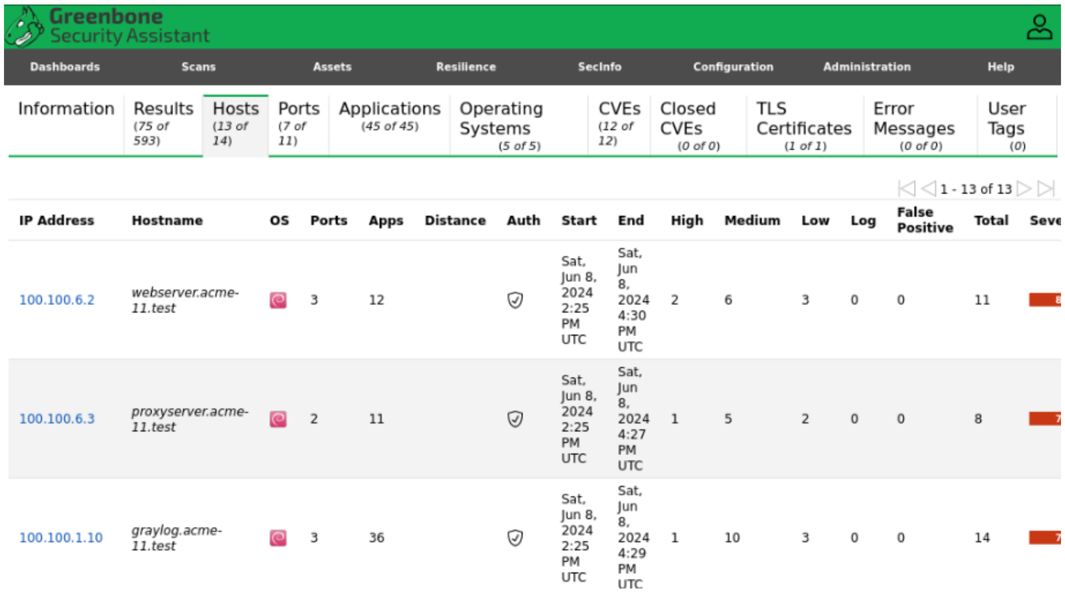
Figure 1: The alias `Greenbone_Scanned_Host` contains IPv4 and IPv6 addresses of the EXTERNAL CLIENTS, DMZ and CLIENTS subnets

INTERNAL interface of the main firewall:



IPv4+6 *	Greenbone	*	MainFirewallNetworks	*	*	*	Allow the Greenbone service to access the hosts in the main firewall networks

In order to test this rule, we can see that in a vulnerability scan (as part of the third assignment), the Greenbone server can reach the hosts of the ACME network, as expected:

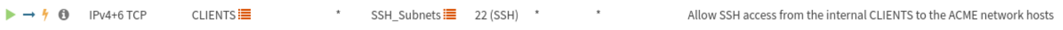


IP Address	Hostname	OS	Ports	Apps	Distance	Auth	Start	End	High	Medium	Low	Log	False Positive	Total	Seve
100.100.6.2	*webserver.acme-11.test*		3	12		✓	Sat, Jun 8, 2024 2:25 PM UTC	Sat, Jun 8, 2024 4:30 PM UTC	2	6	3	0	0	11	8
100.100.6.3	*proxyserver.acme-11.test*		2	11		✓	Sat, Jun 8, 2024 2:25 PM UTC	Sat, Jun 8, 2024 4:27 PM UTC	1	5	2	0	0	8	7
100.100.1.10	*graylog.acme-11.test*		3	36		✓	Sat, Jun 8, 2024 2:25 PM UTC	Sat, Jun 8, 2024 4:29 PM UTC	1	10	3	0	0	14	7

2.7 All network hosts must be managed via SSH only from hosts within the Client network

In order to allow SSH access to the hosts of the ACME network we need to add a line in the `/etc/ssh/sshd_config` file: `PermitRootLogin yes`.

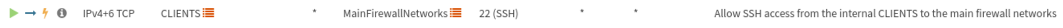
We add a rule on the CLIENTS interface of the internal firewall to permit TCP traffic on port 22 to reach the all the other subnets of the ACME network.



IPv4+6 TCP	CLIENTS	*	SSH_Subnets	22 (SSH)	*	*	Allow SSH access from the internal CLIENTS to the ACME network hosts

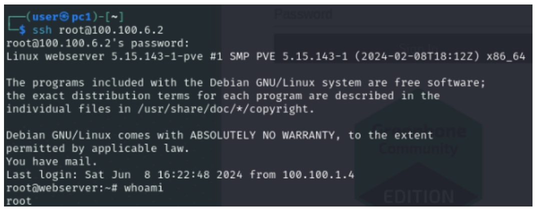
Figure 2: The alias `SSH_Subnets` contains IPv4 and IPv6 addresses of the EXTERNAL CLIENTS, DMZ and SERVERS subnets

We also need to allow the traffic on the INTERNAL interface of the main firewall.

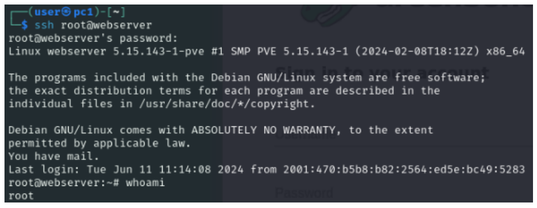


IPv4+6 TCP	CLIENTS	*	MainFirewallNetworks	22 (SSH)	*	*	Allow SSH access from the internal CLIENTS to the main firewall networks

To test this we can try to connect with SSH as the root user from the Kali machine to any other host, for instance, the Webserver, using both IPv4 and IPv6.



```
┌──(user㉿pc1)-[~]
└─$ ssh root@100.100.6.2
root@100.100.6.2's password:
Linux webserver 5.15.143-1-pve #1 SMP PVE 5.15.143-1 (2024-02-08T18:12Z) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
Last login: Sat Jun  8 16:22:48 2024 from 100.100.1.4
root@webserver:~# whoami
root
```

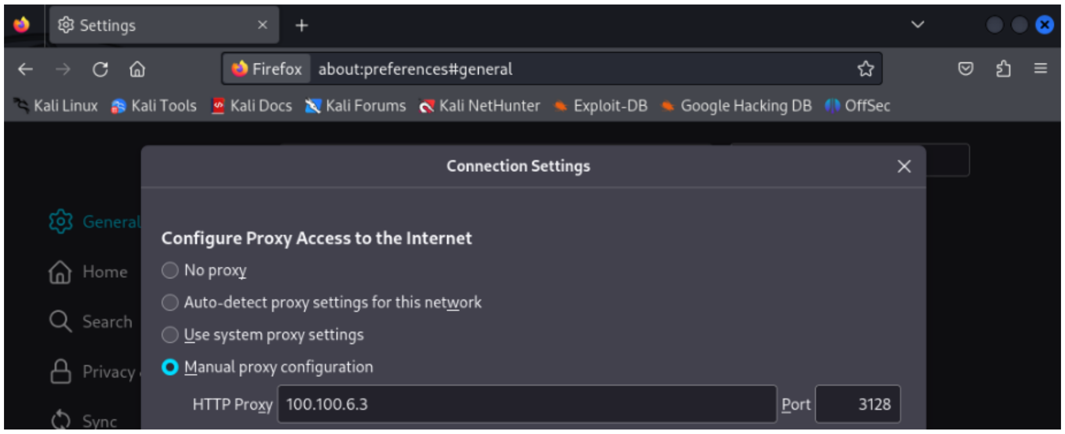


```
┌──(user㉿pc1)-[~]
└─$ ssh root@webserver
root@webserver's password:
Linux webserver 5.15.143-1-pve #1 SMP PVE 5.15.143-1 (2024-02-08T18:12Z) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
Last login: Tue Jun 11 11:14:08 2024 from 2001:470:b5b8:b82:2564:ed5e:bc49:5283
root@webserver:~# whoami
root
```

2.8 The Client network hosts have only access to external web services (HTTP/HTTPS) through the proxy server in the DMZ

Firstly, we need to configure a forward proxy on the Proxyserver. The details about how we configured the SQUID forward proxy can be found in the report for the third assignment.

Secondly, we need to set the proxy configuration in the kali and arpwatch machines so that they use the Proxyserver as a proxy for HTTP and HTTPS traffic:

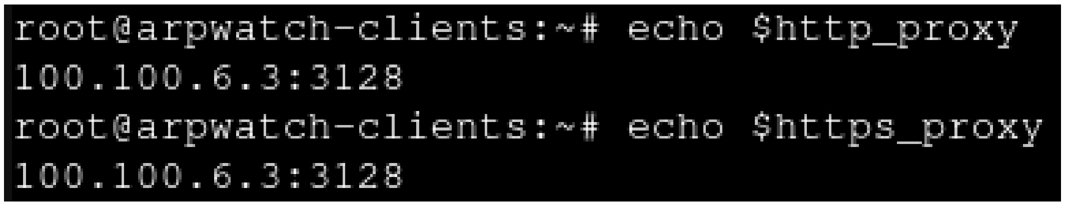


Connection Settings

Configure Proxy Access to the Internet

- No proxy
- Auto-detect proxy settings for this network
- Use system proxy settings
- [✓ Manual proxy configuration]

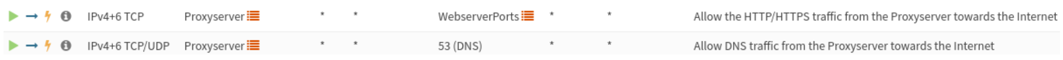
HTTP Proxy: 100.100.6.3 Port: 3128



```
root@arpwatch-clients:~# echo $http_proxy
100.100.6.3:3128
root@arpwatch-clients:~# echo $https_proxy
100.100.6.3:3128
```

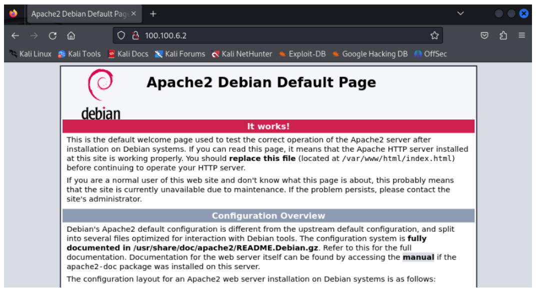
In 2.3 we already allowed access from all the hosts in the ACME network to the Proxyserver, and that includes also the CLIENTS subnet.

Finally, we need to add a rule in the DMZ interface of the main firewall, to allow the Proxyserver to send HTTP and HTTPS (and obviously DNS) traffic towards the Internet.



IPv4+6 TCP	Proxyserver	*	*	WebserverPorts	*	*	Allow the HTTP/HTTPS traffic from the Proxyserver towards the Internet
IPv4+6 TCP/UDP	Proxyserver	*	*	53 (DNS)	*	*	Allow DNS traffic from the Proxyserver towards the Internet

In order to test this rule we can try to reach the HTTP server on the Webserver and Github (using HTTPS).



Apache2 Debian Default Page

It works!

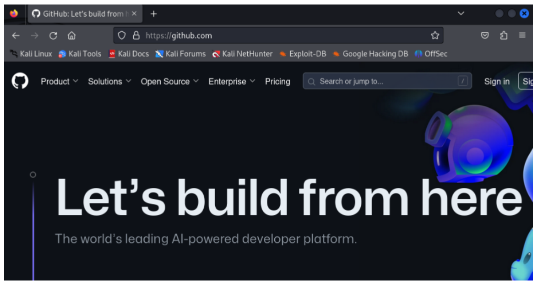
This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:



Let's build from here

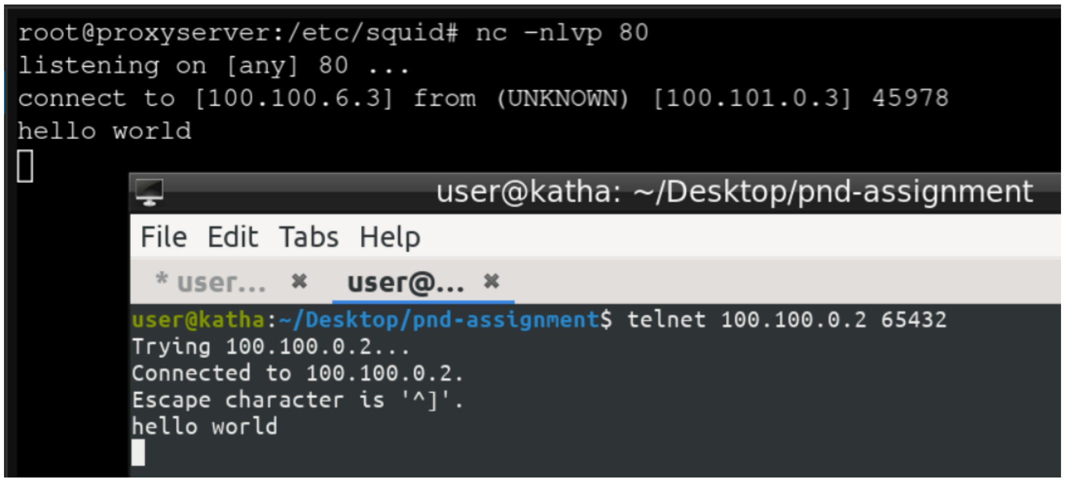
The world's leading AI-powered developer platform.

2.9 Any packet the Main Firewall receives on port 65432 should be redirected to port 80 of the proxy host

To do so we need to perform port forwarding, which can be done by accessing NAT on the main firewall.

⟷	WAN	TCP	*	*	*	65432	Proxyserver	80 (HTTP)	Firewall redirection rule

To check if this is working properly, we set the proxy to listen on port 80 using `netcat` and, from our virtual machine, we send a packet to the main firewall on port 65432.



```
root@proxyserver:/etc/squid# nc -nlvp 80
listening on [any] 80 ...
connect to [100.100.6.3] from (UNKNOWN) [100.101.0.3] 45978
hello world
```

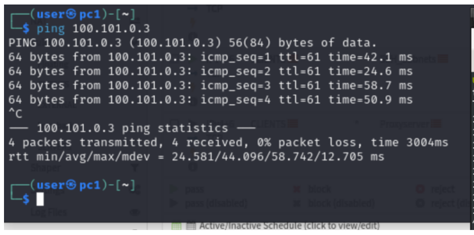
```
user@katha:~/Desktop/pnd-assignment$ telnet 100.100.0.2 65432
Trying 100.100.0.2...
Connected to 100.100.0.2.
Escape character is '^]'.
hello world
```

2.10 All the internal hosts should use the public IP address of the Main Firewall to exit towards the Internet

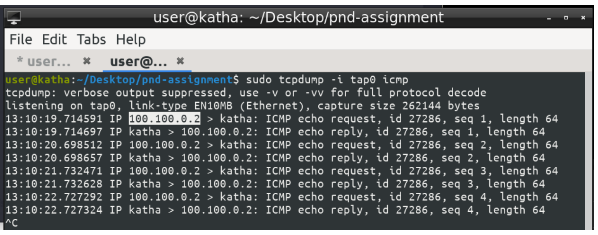
In order to do so, we need an Outbound NAT rule on the main firewall. We have interpreted "internal hosts" as all the hosts located in the subnets connected to the internal firewall, namely CLIENTS and SERVERS.

▶	INTERNAL	InternalFirewallNetworks	*	*	*	Interface address	*	NO

To test this, we can use a host located in the specified subnets to connect to our virtual machine and we can then check the IP that appears as source of the connection.



```
┌──(user㉿pc1)-[~]
└─$ ping 100.101.0.3
PING 100.101.0.3 (100.101.0.3) 56(84) bytes of data.
64 bytes from 100.101.0.3: icmp_seq=1 ttl=61 time=42.1 ms
64 bytes from 100.101.0.3: icmp_seq=2 ttl=61 time=24.6 ms
64 bytes from 100.101.0.3: icmp_seq=3 ttl=61 time=58.7 ms
64 bytes from 100.101.0.3: icmp_seq=4 ttl=61 time=50.9 ms
^C
── 100.101.0.3 ping statistics ──
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 24.581/44.096/58.742/12.705 ms

┌──(user㉿pc1)-[~]
└─$ 
```



```
user@katha:~/Desktop/pnd-assignment$ sudo tcpdump -i tap0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:10:19.714591 IP 100.100.0.2 > katha: ICMP echo request, id 27286, seq 1, length 64
13:10:19.714697 IP katha > 100.100.0.2: ICMP echo reply, id 27286, seq 1, length 64
13:10:20.698512 IP 100.100.0.2 > katha: ICMP echo request, id 27286, seq 2, length 64
13:10:20.698657 IP katha > 100.100.0.2: ICMP echo reply, id 27286, seq 2, length 64
13:10:21.732471 IP 100.100.0.2 > katha: ICMP echo request, id 27286, seq 3, length 64
13:10:21.732628 IP katha > 100.100.0.2: ICMP echo reply, id 27286, seq 3, length 64
13:10:22.727292 IP 100.100.0.2 > katha: ICMP echo request, id 27286, seq 4, length 64
13:10:22.727324 IP katha > 100.100.0.2: ICMP echo reply, id 27286, seq 4, length 64
^C
```

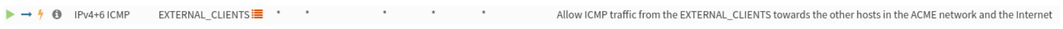
The IP address that appears as source is `100.100.0.2` which is indeed the IP address of the WAN interface of the main firewall.

2.11 All the hosts of the ACME network should be able to ping (and receive replies of) the other hosts and the Internet hosts

We want the hosts inside the ACME network to be able to send ping requests/replies among each other, but we want to block ping requests coming from the Internet. Only the hosts inside ACME network must be able to ping Internet hosts.

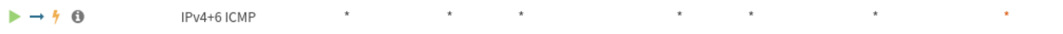
In order to do this, we need to add a rule to allow ICMP traffic on each interface of the main firewall (except WAN) and each interface of the internal firewall.

For instance, on the EXTERNAL CLIENTS interface:



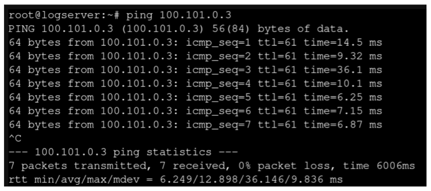
IPv4+6 ICMP EXTERNAL_CLIENTS * * * * * Allow ICMP traffic from the EXTERNAL_CLIENTS towards the other hosts in the ACME network and the Internet

Since the same rule has to be applied on each interface of the internal firewall, we added it to the floating table:

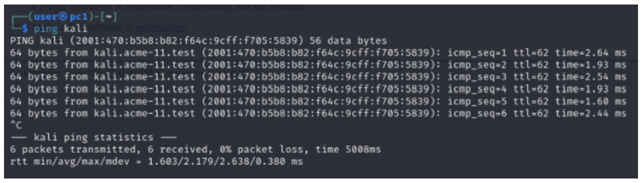


IPv4+6 ICMP * * * * * * *

We can test it by checking the reachability of the Internet from our hosts in the ACME network by pinging our host from the Logserver, and the reachability between hosts of the ACME network by pinging kali from the client-ext-1.



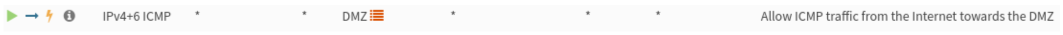
```
root@logserver:~# ping 100.101.0.3
PING 100.101.0.3 (100.101.0.3) 56(84) bytes of data.
64 bytes from 100.101.0.3: icmp_seq=1 ttl=61 time=14.5 ms
64 bytes from 100.101.0.3: icmp_seq=2 ttl=61 time=9.32 ms
64 bytes from 100.101.0.3: icmp_seq=3 ttl=61 time=36.1 ms
64 bytes from 100.101.0.3: icmp_seq=4 ttl=61 time=10.1 ms
64 bytes from 100.101.0.3: icmp_seq=5 ttl=61 time=6.25 ms
64 bytes from 100.101.0.3: icmp_seq=6 ttl=61 time=7.15 ms
64 bytes from 100.101.0.3: icmp_seq=7 ttl=61 time=6.87 ms
^C
--- 100.101.0.3 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6006ms
rtt min/avg/max/mdev = 6.249/12.898/36.146/9.836 ms
```



```
┌──(user㉿pc1)-[~]
└─$ ping kali
PING kali (2001:470:b5b8:b82:f64c:9cff:f705:5839) 56 data bytes
64 bytes from kali.acme-11.test (2001:470:b5b8:b82:f64c:9cff:f705:5839): icmp_seq=1 ttl=62 time=2.64 ms
64 bytes from kali.acme-11.test (2001:470:b5b8:b82:f64c:9cff:f705:5839): icmp_seq=2 ttl=62 time=1.93 ms
64 bytes from kali.acme-11.test (2001:470:b5b8:b82:f64c:9cff:f705:5839): icmp_seq=3 ttl=62 time=2.54 ms
64 bytes from kali.acme-11.test (2001:470:b5b8:b82:f64c:9cff:f705:5839): icmp_seq=4 ttl=62 time=1.93 ms
64 bytes from kali.acme-11.test (2001:470:b5b8:b82:f64c:9cff:f705:5839): icmp_seq=5 ttl=62 time=1.60 ms
64 bytes from kali.acme-11.test (2001:470:b5b8:b82:f64c:9cff:f705:5839): icmp_seq=6 ttl=62 time=2.44 ms
^C
── kali ping statistics ──
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 1.603/2.179/2.638/0.380 ms
```

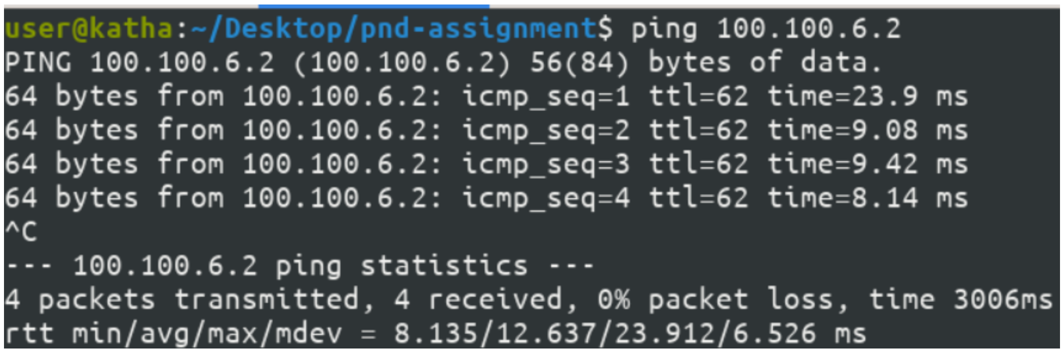
2.12 Only hosts in the DMZ should be reachable using the ping and traceroute tools from the Internet

In order to implement this, we can add a rule on the WAN interface of the main firewall, allowing ICMP packets from the Internet towards the DMZ subnet.



IPv4+6 ICMP	*	*	DMZ	*	*	*	Allow ICMP traffic from the Internet towards the DMZ

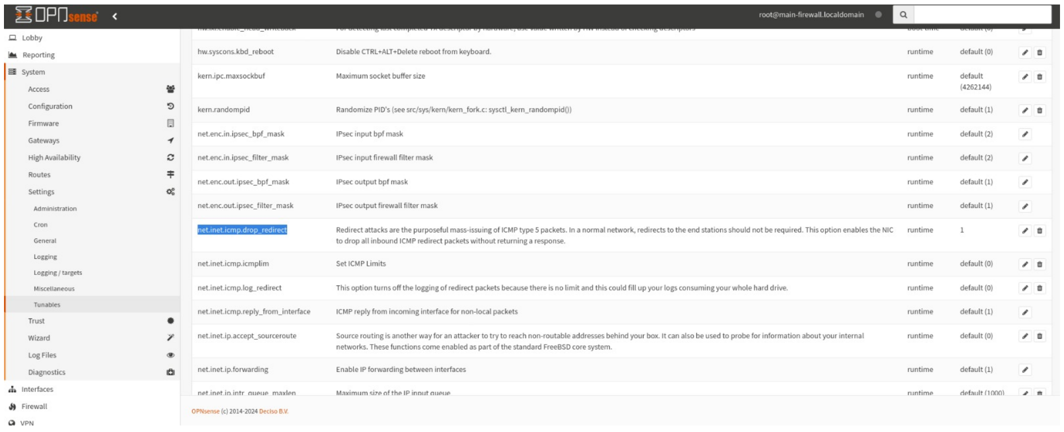
To test this we can use our host to ping the Webserver. Again, we cannot perform the same test with IPv6 since we don't know the IPv6 route to connect to the ACME network from our host.



```
user@katha:~/Desktop/pnd-assignment$ ping 100.100.6.2
PING 100.100.6.2 (100.100.6.2) 56(84) bytes of data.
64 bytes from 100.100.6.2: icmp_seq=1 ttl=62 time=23.9 ms
64 bytes from 100.100.6.2: icmp_seq=2 ttl=62 time=9.08 ms
64 bytes from 100.100.6.2: icmp_seq=3 ttl=62 time=9.42 ms
64 bytes from 100.100.6.2: icmp_seq=4 ttl=62 time=8.14 ms
^C
--- 100.100.6.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 8.135/12.637/23.912/6.526 ms
```

2.13 ICMP redirect packets should not cross any network

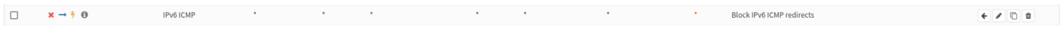
In OPNsense firewalls, IPv4 ICMP redirects are not allowed by default, as we can see from the configuration of `System > Tunables` for the `net.inet.icmp.drop_redirect` of the main firewall (it's the same for the internal firewall):



hw.syscons.kbd_reboot	Disable CTRL+ALT+Delete reboot from keyboard.	runtime	default (0)
kern.ipc.maxsockbuf	Maximum socket buffer size	runtime	default (4262144)
kern.randompid	Randomize PID's (see src/sys/kern/kern_fork.c: sysctl_kern_randompid())	runtime	default (1)
net.enc.in.ipsec_bpf_mask	IPsec input bpf mask	runtime	default (2)
net.enc.in.ipsec_filter_mask	IPsec input firewall filter mask	runtime	default (2)
net.enc.out.ipsec_bpf_mask	IPsec output bpf mask	runtime	default (1)
net.enc.out.ipsec_filter_mask	IPsec output firewall filter mask	runtime	default (1)
net.inet.icmp.drop_redirect	Redirect attacks are the purposeful mass-issuing of ICMP type 5 packets. In a normal network, redirects to the end stations should not be required. This option enables the NIC to drop all inbound ICMP redirect packets without returning a response.	runtime	1
net.inet.icmp.icmplim	Set ICMP Limits	runtime	default (0)
net.inet.icmp.log_redirect	This option turns off the logging of redirect packets because there is no limit and this could fill up your logs consuming your whole hard drive.	runtime	default (0)
net.inet.icmp.reply_from_interface	ICMP reply from incoming interface for non-local packets	runtime	default (1)
net.inet.ip.accept_sourceroute	Source routing is another way for an attacker to try to reach non-routable addresses behind your box. It can also be used to probe for information about your internal networks. These functions come enabled as part of the standard FreeBSD core system.	runtime	default (0)
net.inet.ip.forwarding	Enable IP forwarding between interfaces	runtime	default (1)
net.inet.ip.intr_queue_maxlen	Maximum size of the IP input queue	runtime	default (1000)

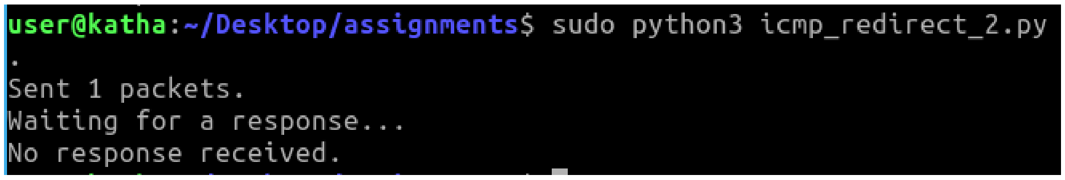
OPNsense (c) 2014-2024 Deciso B.V.

Since IPv6 ICMP redirects are not blocked by default, we need to add a rule in the floating table (because it needs to be applied to all interfaces) of both the main firewall and the internal firewall. We need to explicitly block ICMP redirects because we allowed all ICMP types in sections 2.11 and 2.12.



```
IPv6 ICMP    *    *    *    *    *    *    *    Block IPv6 ICMP redirects
```

In order to test this rule, we created a Python script that generates ICMP redirect packets, and it waits for a response. As we can see, we don't receive any response, so the ICMP redirects are successfully blocked in our ACME network.



```
user@katha:~/Desktop/assignments$ sudo python3 icmp_redirect_2.py
.
Sent 1 packets.
Waiting for a response...
No response received.
```

2.14 Anything that is not explicitly allowed has to be denied.

Since the default policy for OPNsense firewalls is to deny incoming traffic, we explicitly allowed only the traffic specified in the policies. Consequently, no other traffic not explicitly stated in the assigned policies is allowed by our firewalls.

3 Final Remarks

Overall, this assignment was challenging, requiring significant time to understand and apply the concepts learned during lectures. One of the most difficult aspects was finding suitable guides and resources online to troubleshoot the problems we encountered during the process. Despite these challenges, it was rewarding to see everything working properly in the end.