Practical Network Defense

ASSIGNMENT 3

Hardening of the ACME network

Group number: **11**

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

Contents

1 Initial Brainstorming

Given the considerable flexibility allowed for this assignment, we have chosen to focus on implementing the network defense elements that were most thoroughly covered in our lectures.
We decided to configure and utilize existing services within our ACME topology, such as Greenbone and Graylog. Additionally, we installed new tools and services to enhance the overall security of the network, including Squid, ModSecurity, and Fail2Ban.
We evaluated the effectiveness and correctness of each implemented security measure by combining the Protection Plan Implementation and the Protection Plan Evaluation into a single section. This approach allowed us to present the tests immediately following the implementation steps.

2 Protection plan definition

To achieve comprehensive protection for our ACME network, we have deployed a range of security measures, including both network infrastructure and application-level defenses. Our strategy encompasses the following key components:

- **Forward Proxy (Squid)**: This proxy server is used to filter web traffic, enhancing security and performance by controlling access to the internet also through authentication and reducing the load on our network.
- **Reverse Proxy (ModSecurity)**: ModSecurity acts as a reverse proxy to secure web applications by filtering and monitoring HTTP traffic, protecting against a variety of attacks such as SQL injection and cross-site scripting (XSS).
- **Host Intrusion Prevention System (IPS) (Fail2Ban)**: Fail2Ban is configured to monitor log files for suspicious activity and automatically block IP addresses that show malicious behavior, thereby preventing potential intrusions.
- **Security Information and Event Management (SIEM) (Graylog)**: Graylog provides centralized logging and real-time analysis of security events, enabling us to detect and respond to threats quickly and efficiently.
- **Vulnerability Scanner (Greenbone)**: Greenbone's scanner is employed to regularly assess our network for vulnerabilities, ensuring that we can identify and patch security weaknesses promptly.
- **General Management, Control, and Data Plane Protection Mechanisms**: We have implemented additional controls across the network to safeguard the management, control, and data planes. These mechanisms are designed to ensure that all aspects of the network are protected against unauthorized access and cyber threats.

Through the deployment of these measures, we aim to establish a robust defense posture for the ACME network, addressing a wide range of security challenges and ensuring the integrity, confidentiality, and availability of our network services.

3 Protection plan implementation and evaluation

3.1 Forward Proxy: SQUID

As requested by the security policy for the first assignment, we have set up the SQUID proxy in the Proxyserver to forward incoming HTTP and HTTPS requests from hosts in the CLIENTS subnet towards their final destination, which can be either the Webserver in the DMZ or other web applications on the Internet.
We installed the SQUID proxy on the Proxyserver and we configured it by editing the `/etc/squid/squid.conf` file.

3.1.1 ACL and Authentication

In order to allow incoming requests, our first step was to define an Access Control List (ACL) for the CLIENTS network and allowed traffic coming from them. We deny access to the traffic not coming from the CLIENTS network, since we do not expect any connection coming from other hosts.
Authentication for the allowed hosts has been provided using the `digest` method. It does not require the exchange of passwords because it is challenge-based and hashes the authentication requests using `md5`, unlike `basic authentication`.
The `user:password` credentials that we have implemented are:

- `al:al`;
- `john:john`;
- `jack:jack`;

3.1.2 Listening and Allowed Ports

As a listening port for our forward proxy, we decided to use the SQUID's default port which is `3128`. Then, we defined the allowed ports for incoming proxy requests (`80`, `443`) and the CONNECT method for `HTTPS` requests.
The access to any other port was denied.

3.1.3 Logging

We enabled logging by specifying the file `/var/log/squid/access.log`, and we specified the `logformat` using the `squid` keyword.

3.1.4 Final configuration file

In Figure 1, the complete configuration of the `squid.conf` file is shown.

```
# Clients Network ACL
acl clients_net src 100.100.2.0/24
acl clients_net src 2001:470:b5b8:b82::/64
acl SSL_ports port 443
acl Safe_ports port 80 # http
acl Safe_ports port 443 # https
acl CONNECT method CONNECT
acl all src all
# Authentication configuration
auth_param digest program /usr/lib/squid3/digest_file_auth -c /etc/squid/passwd
auth_param digest utf8 on
auth_param digest children 5
auth_param digest realm acme11
auth_param digest nonce_garbage_interval 5 minutes
auth_param digest nonce_max_duration 30 minutes
auth_param digest nonce_max_count 50
auth_param digest nonce_strictness on
auth_param digest check_nonce_count on
auth_param digest post_workaround on
acl authenticated proxy_auth REQUIRED
http_access allow authenticated
http_access allow clients_net
# Requests not coming from clients net cannot proceed further
http_access deny !clients_net
# Deny requests to certain unsafe ports
http_access allow Safe_ports
http_access deny !Safe_ports
# Deny CONNECT to other than secure SSL ports
http_access allow CONNECT SSL_ports
http_access deny CONNECT !SSL_ports
# And finally deny all other access to this proxy
http_access deny all
http_port 3128
# Define DNS servers for Squid to use
dns_nameservers 8.8.8.8
# Logging
access_log daemon:/var/log/squid/access.log squid
```

Figure 1: `squid.conf`

3.1.5 Test of the configuration

To perform the testing phase we used `Wireshark`.
First, we specified in the Firefox settings of the Kali machine the Proxyserver IP address and port to use, in order to test the forward proxy functionality from the `kali` machine in the CLIENTS network.

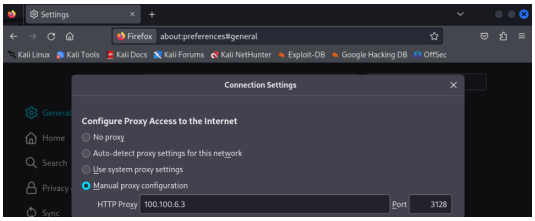


Figure 2: Firefox settings in Kali

Then, we requested a web page, `github.com`, to check that the site was going to ask for credentials:

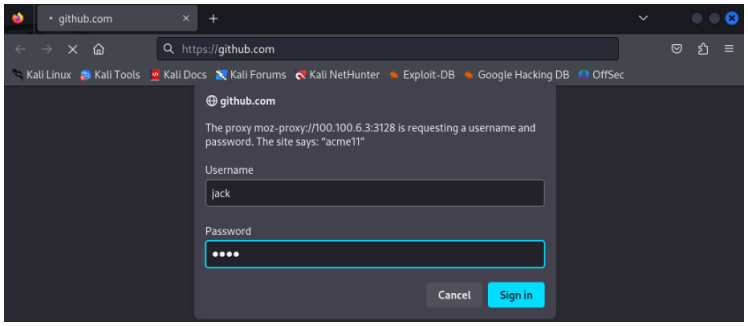


Figure 3: Authentication required

After inserting a correct combination of `username:password` we were able to visualize the page:

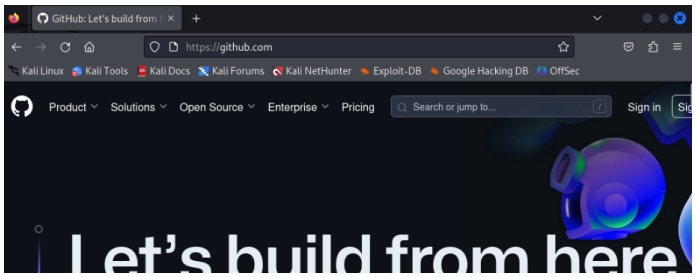


Figure 4: Forward Proxy authentication functioning correctly

In the meantime, we captured the traffic using `Wireshark`, and then we analysed it:

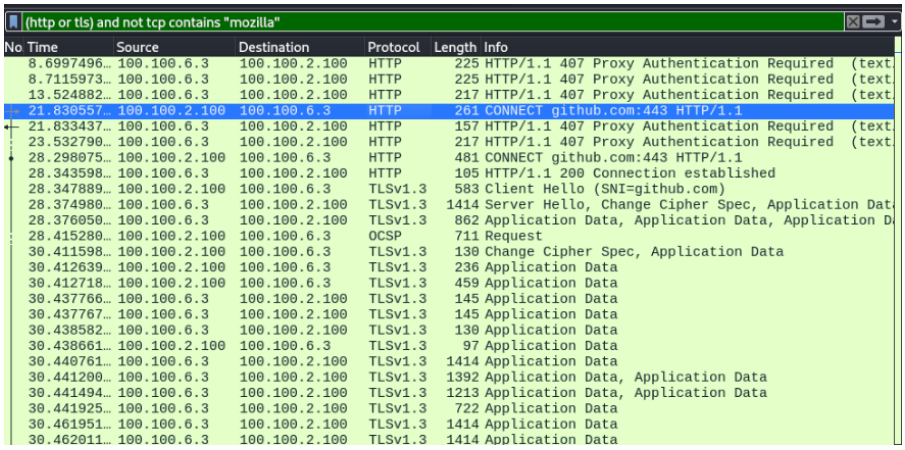


Figure 5: `HTTP CONNECT` method

As we can see, since we enabled the use of the `HTTP CONNECT` method for the SQUID forward proxy, the `HTTPS` connection is established with the `CONNECT` method.

3.2 Reverse Proxy: Apache2 Modsecurity

To implement the reverse proxy we used the `apache2`'s built-in proxy redirection functionality. For this reason, we installed `Modsecurity` on the Webserver.

By editing the apache configuration files we managed to enable all the other functions.

3.2.1 HTTPS Redirection and Proxy Directive

In order to enable `HTTPS` functionality on the fantasticcoffee machine, we configured the Webserver machine to serve as the SSL endpoint. The first step in this process was to generate the necessary certificates and keys, which we accomplished by following these steps:

- `openssl genrsa -out ca.key 2048` to generate the private key;
- `openssl req -new -key ca.key -out ca.csr` to generate the Certificate Signing Request (CSR);
- `openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt` to generate the certificate.

Then, we moved the certificate and the private key into the `/etc/ssl` directory. Finally, we edited `/etc/apache2/sites-enabled/000-default.conf` as follows to enable `HTTPS` endpoint by making the server listen on port `443`.
Our next step was to enable redirection, the core functionality of our reverse proxy.
Here is the resulting `000-default.conf` file:

```
<VirtualHost *:443>
        ServerName webserver

        SSLEngine On
        SSLCertificateFile "/etc/ssl/certs/ca.crt"
        SSLCertificateKeyFile "/etc/ssl/private/ca.key"

        ProxyPreserveHost On
        ProxyPass "/coffee/" "http://100.100.4.10/"
        ProxyPassReverse "/coffee/" "http://100.100.4.10/"

        SecRuleEngine on
</VirtualHost>
```

Figure 6: `000-default.conf`

According to these settings, every request directed towards port `443` of the Webserver will be redirected towards the `fantasticcoffee` machine.

3.2.2 Modsecurity

We spent some time understanding the functionalities of `fantasticcoffee` in order to properly secure the machine. Then, we installed (on the Webserver) and configured `modsecurity` as follows:

- In the file `/etc/modsecurity/modsecurity.conf` we set `SecRuleEngine` to `On` in order to enable its engine.
- Via `Include` we specified the rules path also in `/etc/modsecurity/modsecurity.conf`.
- In `/etc/apache2/sitesavailable/000-default.conf` we enabled the `SecRuleEngine` for our `HTTPS` endpoint.

```
# Two POST Parameters
SecRule REQUEST_FILENAME "action.asp" "id:100,phase:2,chain,deny,status:400,log,msg:'Incorrect Number of Action Params'"
SecRule &ARGS "!^2$"
# Only product or sugar
SecRule REQUEST_FILENAME "action.asp" "id:101,phase:2,chain,deny,status:400,log,msg:'Incorrect Action Params Names'"
SecRule ARGS_NAMES "!^(product|sugar)$"
```

Figure 7: First rule from `modsecurity.conf`

These are the rules that we have decided to implement:
We restricted the `action.asp` endpoint to accept only two parameters: `product` and `sugar`.
We also wanted the inputs to align with the supported values of the machine:

```
# Allowed values for product and sugar
SecRule REQUEST_FILENAME "action.asp" "id:102,phase:2,chain,deny,status:400,log,msg:'Incorrect value for param product'"
SecRule ARGS:product "!^(tea|hotwater|coffee)$"

SecRule REQUEST_FILENAME "action.asp" "id:103,phase:2,chain,deny,status:400,log,msg:'Incorrect value for param sugar'"
SecRule ARGS:sugar "!^(0|1|2|3)$"
```

Figure 8: Second rule from `modsecurity.conf`

Finally, we allowed only the `GET` requests to `open-cash.asp`.

```
# Open Cash (GET /open-cash.asp)
SecRule REQUEST_FILENAME "open-cash.asp" "id:104,phase:1,chain,deny,status:400,log,msg:'Not Supported Method for open-cash'"
SecRule REQUEST_METHOD "!GET"
```

Figure 9: Third rule from `modsecurity.conf`

Changes to the firewall configuration were not necessary since everything was correctly set up in the first assignment.

3.2.3 Test of the configuration

To perform the testing phase, we mainly utilized `tcpdump`, `Wireshark` and `Burpsuite`. To check if the redirection functionality of the reverse proxy is working properly, we can use the Firefox browser on kali to navigate on `https://100.100.6.2/coffee/`.
In this way, the request will go through the forward proxy and then will reach the Webserver, which will forward the request to the `fantasticcoffee` machine.

No	Time	Source	Destination	Protocol	Length	Info
1	0.000000	proxyserver.…	webserver.acm…	TLSv1.2	445	Application Data
2	0.001805	webserver.ac…	fantastic-cof…	HTTP	508	GET /display.asp HTTP/1.1
3	0.002581	fantastic-co…	webserver.acm…	HTTP	182	HTTP/1.1 200 OK (text/plain)
5	0.002911	webserver.ac…	proxyserver.a…	TLSv1.2	291	Application Data
7	0.500498	proxyserver.…	webserver.acm…	TLSv1.2	445	Application Data
8	0.502612	webserver.ac…	fantastic-cof…	HTTP	508	GET /display.asp HTTP/1.1
9	0.503345	fantastic-co…	webserver.acm…	HTTP	182	HTTP/1.1 200 OK (text/plain)
	0.503691	webserver.ac…	proxyserver.a…	TLSv1.2	291	Application Data
	1.001026	proxyserver.…	webserver.acm…	TLSv1.2	445	Application Data
	1.003076	webserver.ac…	fantastic-cof…	HTTP	508	GET /display.asp HTTP/1.1
	1.003970	fantastic-co…	webserver.acm…	HTTP	182	HTTP/1.1 200 OK (text/plain)

Figure 10: Captured traffic on the Webserver

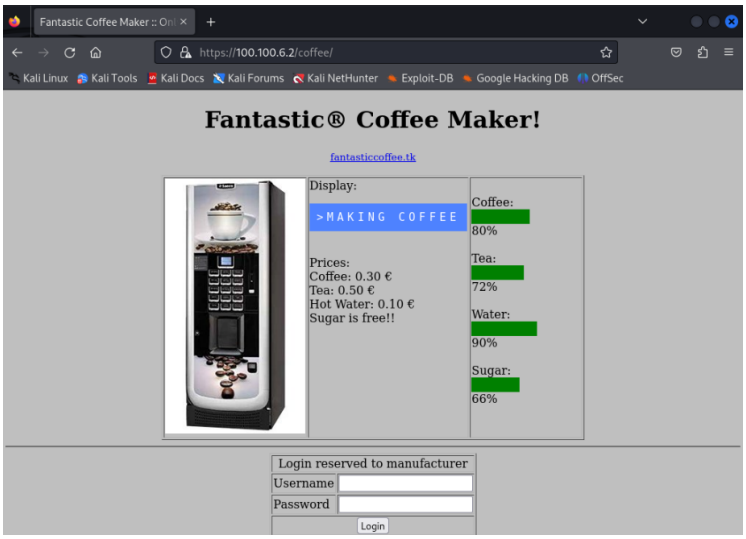


Figure 11: Fantasticcoffee home page

Finally, we tested `modsecurity` using `Burpsuite` to manipulate parameters of the requests we sent and analyse their responses. As shown in Figure 12, sending a request with correct parameters, `product` and `sugar`, and appropriate input values for these parameters results in the expected response, which is a redirection.

Request

Pretty Raw Hex

```
1  POST /coffee/action.asp HTTP/1.1
2  Host: 100.100.6.2
3  Cookie: sessionid=bJyEfoj6fLU$
4  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:123.0)
   Gecko/20100101 Firefox/123.0
5  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
   bp,*/*;q=0.8
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate, br
8  Content-Type: application/x-www-form-urlencoded
9  Content-Length: 22
10 Origin: https://100.100.6.2
11 Referer: https://100.100.6.2/coffee/index.asp
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Te: trailers
18 Connection: keep-alive
19
20 product=coffee&sugar=0
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 302 Found
2  Date: Wed, 19 Jun 2024 11:13:57 GMT
3  Server: Apache/2.4.56 (Debian)
4  Location: index.asp
5  Content-Length: 0
6  Keep-Alive: timeout=5, max=100
7  Connection: Keep-Alive
8
9
```

Figure 12: Valid request

If, instead, we use an invalid input value for the parameters, such as "matcha" for `product` instead of the allowed values "coffee," "tea," and "hotwater," we correctly receive a `400 Bad Request` response code in the response to our request.

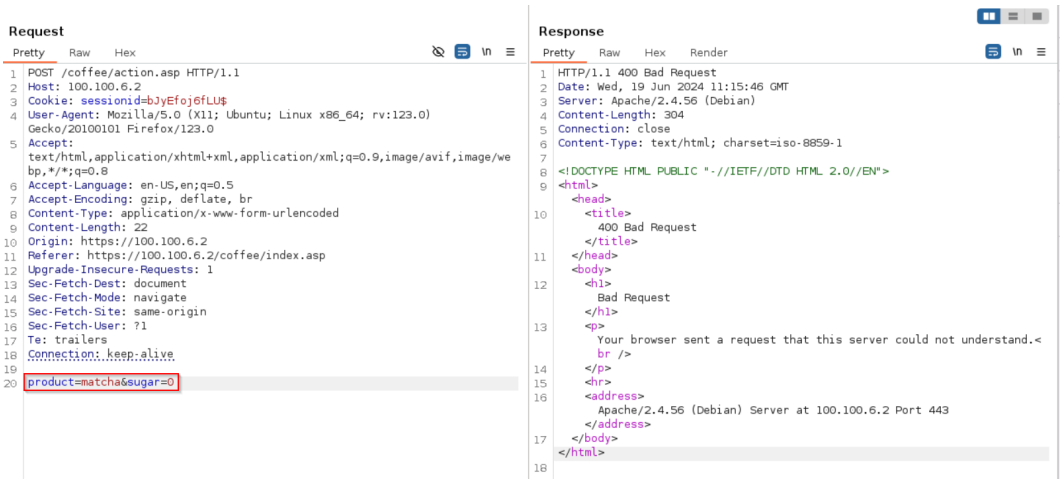


Figure 13: Bad Request

3.3 Host IPS: Fail2ban

In order to protect the hosts in our ACME network against brute-force attacks on SSH passwords, we can implement several countermeasures.
While there are multiple tools available for monitoring login attempts, including some versions of OPNsense, we will use Fail2ban. Fail2ban scans log files (e.g., `/var/log/auth.log`) and bans IP addresses that exhibit malicious behavior, such as too many password failures or exploit attempts. Generally, Fail2ban updates firewall rules to reject the malicious IP addresses for a specified amount of time, though it can also be configured to perform other actions, such as sending an email notification.
Fail2ban comes with pre-configured filters for various services (apache, courier, ssh, etc.), making it easy to set up for SSH monitoring, which is what we have been focusing on.

3.3.1 Ban brute-forcing SSH login attempts on the Webserver

To protect the Webserver from SSH brute-forcing, we installed Fail2ban and then we configured it by editing the `/etc/fail2ban/jail.local` file, where we added the following lines to the predefined `sshd` jail:

```
[sshd]

# To use more aggressive sshd modes set filter parameter "mode" in jail.local:
# normal (default), ddos, extra or aggressive (combines all).
# See "tests/files/logs/sshd" or "filter.d/sshd.conf" for usage example and details.
#mode   = normal
enabled = true
port    = ssh
filter  = sshd
logpath = /var/log/auth.log
backend = auto
maxretry = 10
bantime = 600
findtime = 300
action = %(action_)s
```

Figure 14: `/etc/fail2ban/jail.local` file in the Webserver

The `maxretry` variable sets the maximum number of SSH authentication attempts a client can make within a time window defined by `findtime` before being banned. With the

specified settings, Fail2ban will ban a client that unsuccessfully attempts to log in with SSH 10 times within a 5-minute window. The ban will last for 10 minutes. The default ban action (referred to with the parameter `$(action_)s`) is executed automatically by Fail2ban when triggered. We set it up so that it adds a rule in the iptables firewall to block the source IP address of the failed SSH connections.
To test this configuration, we can perform 10 failed login attempts form the Kali host in the CLIENTS subnet, and here we can see the results:

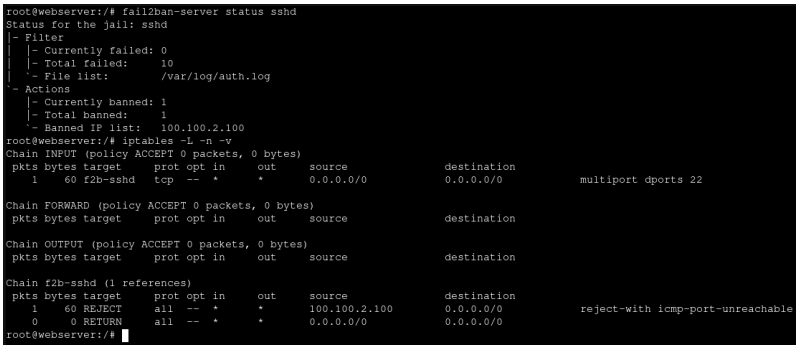


Figure 15: Status of the sshd jail in the Webserver, and iptables rules

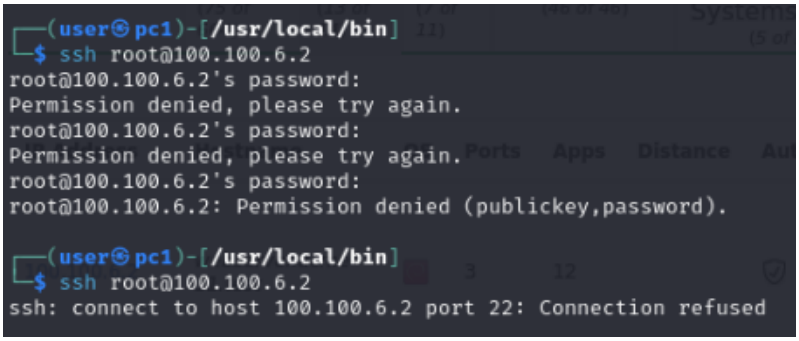


Figure 16: Kali host banned

3.3.2 Ban brute-forcing SSH login attempts from the Logserver

Subsequently, we configured the Fail2Ban IPS on the Logserver to ban or unban hosts that attempt to brute-force other hosts in our ACME network. This is done by directly interacting with the main and internal firewalls using `SSH` and the `pfctl` command.
As before, we installed Fail2ban on the Logserver and then we configured it by editing the `/etc/fail2ban/jail.local` file, adding the following lines to create the `syslog-ssh` custom jail:

```
[syslog-ssh]
enabled  = true
filter   = sshd
logpath  = /var/log/auth.log
maxretry = 10
bantime = 600
findtime = 300
banaction = pfctl
```

Figure 17: `/etc/fail2ban/jail.local` file in the Webserver

We are using the same configuration as before. Fail2ban will ban a client that unsuccessfully attempts to log in via SSH 10 times within a 5-minute window. The ban will last for 10 minutes.
Next, we created a custom action for pfctl, in `/etc/fail2ban/action.d/pfctl.conf`:

```
[Definition]
actionstart = /bin/true
actionstop = /bin/true
actioncheck = /bin/true
actionban = /etc/fail2ban/action.d/pfctl-ban.sh <ip>
actionunban = /etc/fail2ban/action.d/pfctl-unban.sh <ip>
```

Figure 18: `/etc/fail2ban/action.d/pfctl.conf` file

The `actionban` function invokes a custom script we've developed, passing the IP address of the host to be banned as an argument. Similarly, `actionunban` functions in an equivalent way, calling another custom script to unban a previously banned host's IP address, provided as an argument.
More precisely, the `pfctl-ban.sh` script is designed to ban an IP address by adding it to a firewall table (`bruteforce`) on either the main or internal firewall, depending on which subnet the IP address belongs to. It handles both IPv4 and IPv6 addresses.

```
1  #!/bin/bash
2
3  IP=$1
4
5  # Function to check if an IP address is within a subnet
6  ip_in_subnet() {
7      IP=$1
8      SUBNET=$2
9      python3 -c "import ipaddress; print(ipaddress.ip_address('$IP') in
           ipaddress.ip_network('$SUBNET'))"
10 }
11
12 # IPv4 subnets for main firewall
13 MAIN_IPV4_SUBNETS=("100.100.6.0/24" "100.100.4.0/24")
14 # IPv4 subnets for internal firewall
15 INTERNAL_IPV4_SUBNETS=("100.100.2.0/24" "100.100.1.0/24")
16
17 # IPv6 subnets for main firewall
18 root@logserver:/etc/fail2ban/action.d# cat pfctl-ban.sh
19 #!/bin/bash
20
21 IP=$1
```

```
22
23  # Function to check if an IP address is within a subnet
24  ip_in_subnet() {
25      IP=$1
26      SUBNET=$2
27      python3 -c "import ipaddress; print(ipaddress.ip_address('$IP') in
            ipaddress.ip_network('$SUBNET'))"
28  }
29
30  # IPv4 subnets for main firewall
31  MAIN_IPV4_SUBNETS=("100.100.6.0/24" "100.100.4.0/24")
32  # IPv4 subnets for internal firewall
33  INTERNAL_IPV4_SUBNETS=("100.100.2.0/24" "100.100.1.0/24")
34
35  # IPv6 subnets for main firewall
36  MAIN_IPV6_SUBNETS=("2001:470:b5b8:b06::/64" "2001:470:b5b8:b04::/64")
37  # IPv6 subnets for internal firewall
38  INTERNAL_IPV6_SUBNETS=("2001:470:b5b8:b81::/64" "2001:470:b5b8:b82::/64")
39
40  # Determine which firewall to use based on the IP
41  if [[ $IP =~ : ]]; then
42      # IPv6 address
43      for subnet in "${MAIN_IPV6_SUBNETS[@]}"; do
44          if [[ $(ip_in_subnet $IP $subnet) == "True" ]]; then
45              ssh root@100.100.6.1 "pfctl -t bruteforce -T add $IP && pfctl -f
                    /etc/pf.conf"
46              exit 0
47          fi
48      done
49      for subnet in "${INTERNAL_IPV6_SUBNETS[@]}"; do
50          if [[ $(ip_in_subnet $IP $subnet) == "True" ]]; then
51              ssh root@100.100.1.1 "pfctl -t bruteforce -T add $IP && pfctl -f
                    /etc/pf.conf"
52              exit 0
53          fi
54      done
55  else
56      # IPv4 address
57      for subnet in "${MAIN_IPV4_SUBNETS[@]}"; do
58          if [[ $(ip_in_subnet $IP $subnet) == "True" ]]; then
59              ssh root@100.100.6.1 "pfctl -t bruteforce -T add $IP && pfctl -f
                    /etc/pf.conf"
60              exit 0
61          fi
62      done
63      for subnet in "${INTERNAL_IPV4_SUBNETS[@]}"; do
64          if [[ $(ip_in_subnet $IP $subnet) == "True" ]]; then
65              ssh root@100.100.1.1 "pfctl -t bruteforce -T add $IP && pfctl -f
                    /etc/pf.conf"
66              exit 0
67          fi
68      done
69  fi
70
71  echo "IP $IP does not match any known subnets."
72  exit 1
```

The script runs an SSH command on the appropriate firewall to add the IP to the `bruteforce` table using the `pfctl` command, and then reloads the `pf` configuration. Specifically:

- `pfctl -t bruteforce -T add $IP`: Adds the IP address to the `bruteforce` table.
- `pfctl -f /etc/pf.conf`: Reloads the `pf` firewall configuration to apply the changes.

The `pfctl-unban.sh` script does the exact same thing, but instead of adding the IP address to the bruteforce table, it removes it with the following command: `pfctl -t bruteforce -T delete $IP`.
Our next step was to set up the SSH keys for passwordless login on both firewalls, using the following commands:

- `ssh-keygen -t rsa`, which generates an RSA type SSH key pair for the Logserver.
- `ssh-copy-id root@<firewall_IP>`, which copies the generated public SSH key to the specified firewall (done for both the main and the internal firewall), for the user root. This allows for passwordless authentication when logging into the firewall using SSH, as the SSH server on the firewalls can verify the private key stored on the Logserver.

Finally, we need to configure the pf bruteforce table on both firewalls, by editing the `/etc/pf.conf` and then reloading the `pf` configuration:

```
root@main-firewall:~ # cat /etc/pf.conf
table <bruteforce> persist
block in quick from <bruteforce> to any
block out quick from any to <bruteforce>
root@main-firewall:~ #
```

Figure 19: `/etc/pf.conf` file

To test if everything is working properly, we try to perform several failed SSH login attempts from the Kali host to the Proxyserver. As expected, the IP address of the Kali machine is added to the `bruteforce` table of the internal firewall:

```
root@logserver:/etc/fail2ban# fail2ban-client status syslog-ssh
Status for the jail: syslog-ssh
|- Filter
|  |- Currently failed: 1
|  |- Total failed:     12
|  `- File list:        /var/log/auth.log
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   100.100.2.100
root@logserver:/etc/fail2ban# ssh root@100.100.1.1 "pfctl -t bruteforce -T show"
   100.100.2.100
You have new mail in /var/mail/root
root@logserver:/etc/fail2ban# ssh root@100.100.6.1 "pfctl -t bruteforce -T show"
root@logserver:/etc/fail2ban#
```

Figure 20: Status of the sshd jail in the Webserver, and bruteforce table in the internal firewall

Figure 21: Kali host banned, again

3.4 SIEM: Graylog

3.4.1 Syslog configuration

In our ACME network, all hosts except those in the CLIENTS subnet (as per firewall policy requirements of the first assignment) forward their logs to the Logserver. The Logserver hosts the syslog service and functions as a centralized log collector. Subsequently, the Logserver sends all received logs, including its own, to a Graylog server, which serves as a Security Information and Event Management (SIEM) system. This setup enables comprehensive monitoring of our ACME network.
To allow a host to send its logs to the Logserver we have to modify the `/etc/rsyslog.conf` file by adding the following lines:

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

Figure 22: `/etc/rsyslog.conf` file in the Webserver

We decided to include the IPv6 address as an additional layer of security, following the principle of redundancy. In this way, if accessing the log server fails via either IPv4 or IPv6, the other protocol may still provide a functional alternative.
Something similar has been done to allow the Logserver to send all the collected logs to Graylog, the only difference is in the destination IP address.

```
# Send all logs to graylog server
*.* @100.100.1.10:514;RSYSLOG_SyslogProtocol23Format
*.* @[2001:470:b5b8:b81:ba23:a962:7d04:c1f2]:514;RSYSLOG_SyslogProtocol23Format
```

Figure 23: `/etc/rsyslog.conf` file in the Logserver

The transport layer protocol employed is the standard one used by Syslog, which is UDP: it is preferred over TCP since it is faster and will not perform a three-way handshake for every log transfer.
All the necessary firewall rules to permit this traffic were implemented as part the first assignment.

3.4.2 Graylog Web Interface setup

Now that all the logs are correctly being sent to the Logserver and to Graylog, our next step is to configure the Graylog server. We begin by setting up the web interface, which

will be accessible on port 9000. To accomplish this, we have to edit the configuration file `/etc/graylog/server/server.conf` as follows:

- `http_bind_address` must be set to `0.0.0.0:9000`.
- `http_publish_uri` must be set to `http://100.100.1.10:9000/`.
- `enable_cors=true` to guarantee compatibility with modern browsers.

To enable users to query the logs, we need to configure the Graylog server to send all logs to itself on port 5555, to process and store them in an Elasticsearch database. In order to do so, we first need to modify the `/etc/rsyslog.conf` of Graylog, and then on the web interface go to `System` → `Inputs` and add a Syslog UDP input to listen on port 5555 of localhost, `127.0.0.1`.

```
#
# Include all config files in /etc/rsyslog.d/
#
$IncludeConfig /etc/rsyslog.d/*.conf
*.* @127.0.0.1:5555;RSYSLOG_SyslogProtocol23Format
```

Figure 24: Graylog sending the logs to itself on port 5555

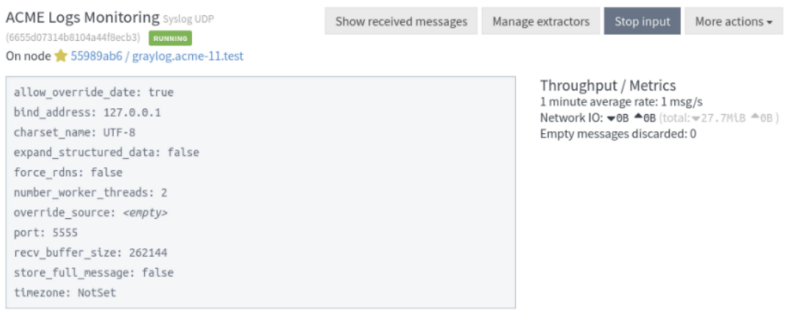


Figure 25: Local input configuration for ACME network logs

In order to check whether the local input is working properly we can use the command `logger` to write something inside the logs and then look for it with the search bar of Graylog. For instance, we executed `logger "Hello Graylog!"` on the Webserver and we were able to query the SIEM to find it, as shown below.

Figure 26: Local input query check

This ensures that all logs are correctly flowing from the ACME network hosts to Graylog through the Logserver, and that we are able to access them from its web interface.

3.4.3 Alerts

One of the main functions of a SIEM is to define alerts that will notify the administrator if certain events that need to be monitored occur. We have identified two events that we believe would be interesting to monitor:

- User fails authentication too many times, it might indicate an ongoing brute-force attack.
- CPU usage goes above 90% for a host, it might indicate a DoS attack.

3.4.4 Authentication Failure

We navigate to `Alerts` → `Event Definition` and click on "Create event definition". We decided to make Graylog continuously monitor for events matching the string "FAILED LOGIN" within a 2-minute time window. The search will be re-executed every 10 seconds to detect real-time changes or occurrences. If the string is matched 4 times or more by logs from the same host, a notification will be triggered.

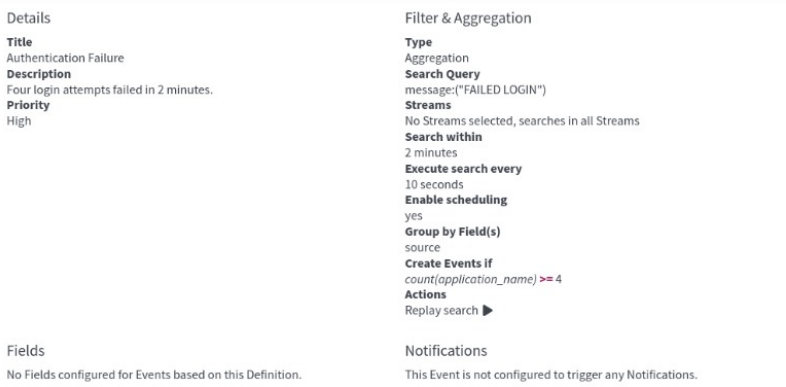


Figure 27: Authentication event summary

To test this we can try to log out, then log in again inside a host (e.g., Webserver) and deliberately fail the authentication 4 times.

Figure 28: Authentication alert has been raised

3.4.5 CPU Usage

In order to perform this kind of check, we need to create a bash script that automatically writes a log when the condition is verified. We decided to insert this script only in two hosts, the Webserver and the Logserver, probably two of the most critical ones.
The following is the bash script we wrote:

```
1 #!/bin/bash
2
3 THRESHOLD=90
```

```
CPU_USAGE=$(top -bn1 | awk '/%Cpu/ { print $2 + $4 }')
DATE=$(date +"%b %d %H:%M:%S")
RESULT=$(echo "$CPU_USAGE > $THRESHOLD" | bc)

if [ $RESULT ]; then
echo "$DATE $(hostname) cpu usage is ${CPU_USAGE}%" | logger -p local0.
    warning
fi
```

We then used Crontab to make the script run every minute, appending its output to a dedicated log file.
To test if the alert is triggered we modified the value of the threshold from 90 to 0, which generates the following notification on Graylog as soon as the `cpu_check.sh` script is executed.

Figure 29: CPU usage alert has been raised

3.5 Graylog Dashboard

Another useful feature of SIEMs is their ability to generate dashboards. Users can add widgets to these dashboards to display information of interest in a convenient format, typically as graphs or tables.

To do so, go to Dashboard and then click "Create new dashboard". Once a dashboard is created, we can create a series of widgets.
We started by creating a histogram showing the amount logs that are generated by each host, this is done using the function `count()`. It also seemed interesting for us to couple this graph with a line graph showing the number of logs received over time.

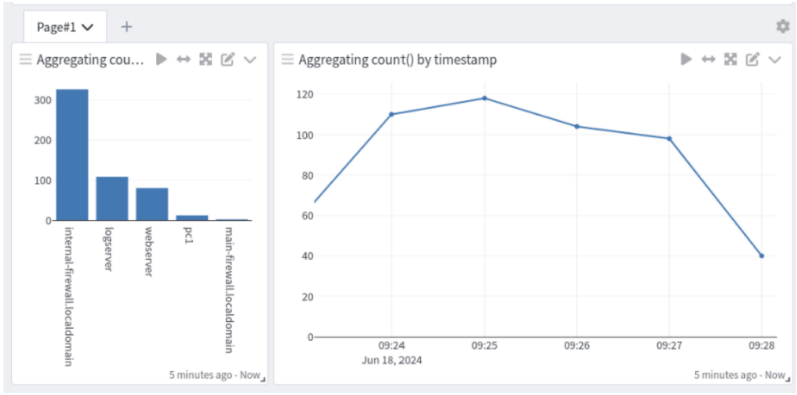
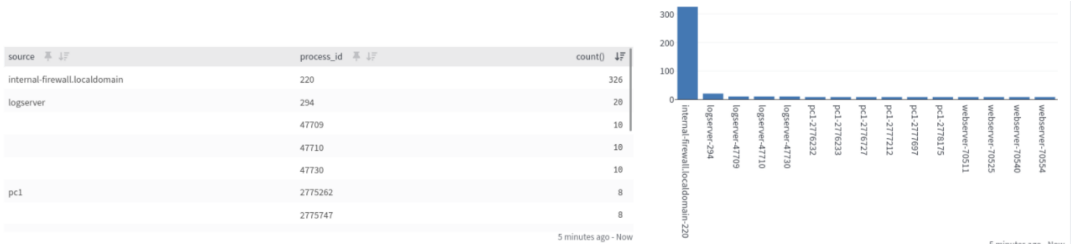


Figure 30: Counting logs per host over time

Moreover, we have decided to create a detailed histogram displaying the number of logs generated by active processes on each host.

This histogram will be sorted from most active to least active, allowing us to quickly identify processes generating a high volume of entries. This capability is crucial for identifying potentially malicious programs, such as those involved in activities like cryptojacking, which may operate discreetly on our machines.



We thought it would be a good idea to present the histogram alongside a data table on the left to help users understand the information better.

3.6 Greenbone Vulnerability Management (GVM)

Greenbone is a widely-used open source vulnerability management tool which uses OpenVAS as a vulnerability scanner. We are able to access the web interface of this service on its default port at the address `https://100.100.1.4:9392`.
From the web interface, we are able to create tasks, set up schedules and manage scan configurations. Another important feature is being able to update NVT, CERT and SCAP feeds to ensure having the latest vulnerability definitions (something that should be done frequently) with the following commands:

```
root@greenbone:/etc/gvm# greenbone-nvt-sync
Running as root. Switching to user '_gvm' and group '_gvm'.
Trying to acquire lock on /var/lib/openvas/feed-update.lock
Acquired lock on /var/lib/openvas/feed-update.lock
i. Downloading Notus files from rsync://feed.community.greenbone.net/community/vulnerability-feed/22.04/vt-data/notus/ to
/var/lib/notus
i. Downloading NASL files from rsync://feed.community.greenbone.net/community/vulnerability-feed/22.04/vt-data/nasl/ to
/var/lib/openvas/plugins
Releasing lock on /var/lib/openvas/feed-update.lock
```

Figure 31: NVT feed synchronized

To perform our first scan of the ACME network we first need to define a target, by going to `Configuration` → `Targets`. We created a target that includes all hosts within the ACME network. It is important to note that for our first assignment, we have already implemented firewall rules that grant the Greenbone server access to every port on every host. This access is crucial for the vulnerability scanner to function correctly.

Name ▲	Hosts	IPs	Port List	Credentials	Actions
ACME network (All the hosts in the ACME network)	100.100.6.1-100.100.6.3, 100.100.4.1, 100.100.4.10, 100.100.4.100, 100.100.1.1-100.100.1.4, 100.100.1.10, 100.100.2.1, 100.100.2.254, 100.100.2.100	14	All IANA assigned TCP and UDP	SSH:root	

Figure 32: ACME target

Our next step was setting up a scan task by going to `Scans` → `Tasks` and pressing "New Task". For each task created we need to select a name, a target (in our case the ACME

target previously created), a scanner (OpenVAS by default) and a configuration. After this is done, we initiated a scan by clicking on "Start".
Once the vulnerability scanning is completed, the same page displays statistics and a summary of the generated reports, including the severity levels of the vulnerabilities identified.

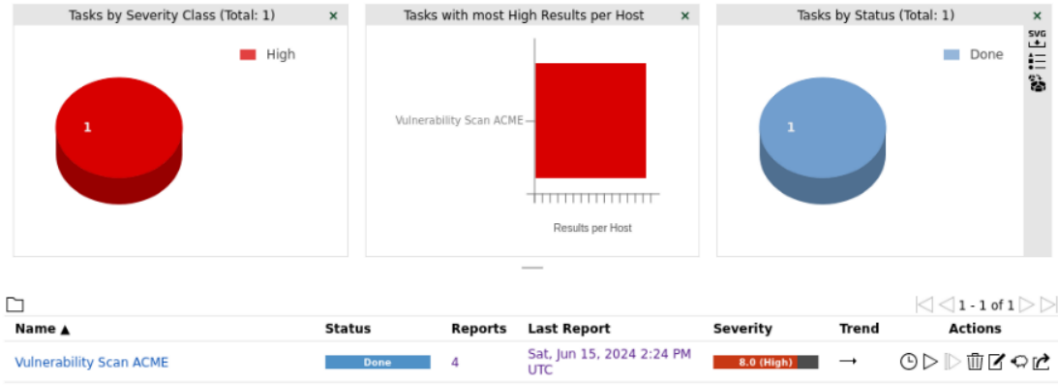


Figure 33: Task completed

By reading the reports we can gather plenty of useful information, such as:

- A list of the vulnerabilities found, together with their severity, their Quality of Detection (QoD), the host and the port to which they are related.
- List of the hosts scanned, enabling us to check which hosts are the most vulnerable, in our case the Webserver and the Proxyserver.
- List of the ports scanned, enabling us to check which ports are the most vulnerable.
- List of the applications scanned, enabling us to check which applications introduce the most vulnerabilities, in our case the SQUID forward proxy.
- A list of Common Vulnerabilities and Exposures (CVEs) found.

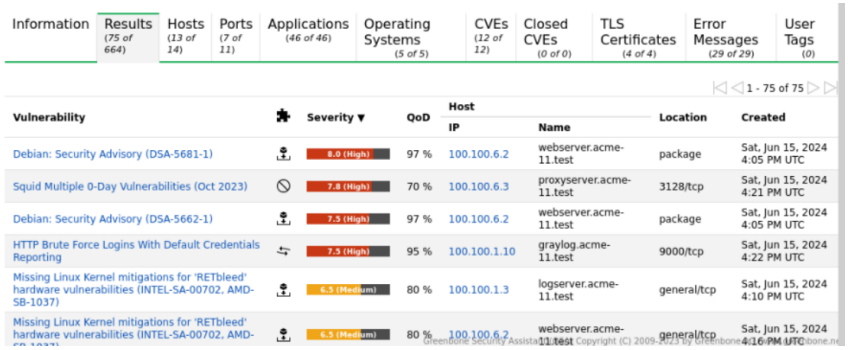


Figure 34: Results of the Greenbone scan

3.7 Network Hardening

Network hardening is the process of reducing the surface of vulnerability of a network in order to minimise the risk of unwanted events. To achieve this objective, various measures can be implemented at different levels, and we will accomplish this using our OPNsense firewalls.

3.7.1 Management Plane

The Management Plane is related to configuring and monitoring network devices.

- Change the default password for the root user to access both firewalls to a randomly generated strong password consisting of uppercase letters, lowercase letters, numbers, and special characters. This new password is 16 characters long it is now used on both firewalls: `lQwG@nVF6M!AlCoe`
- Make sure to log in only using `HTTPS`, which is a secure protocol since it makes use of cryptography, unlike `HTTP`. We go to `System` → `Settings` → `Administration` and select `HTTPS` as protocol:

Figure 35: `HTTPS` protocol

- Configure `NTP` (Network Time Protocol) service on internal interfaces. This was already configured in OPNsense:

Figure 36: `NTP` service

- Sending logs to syslog on the Logserver. We go to `System` → `Settings` → `Logging / Targets` and add an appropriate rule.

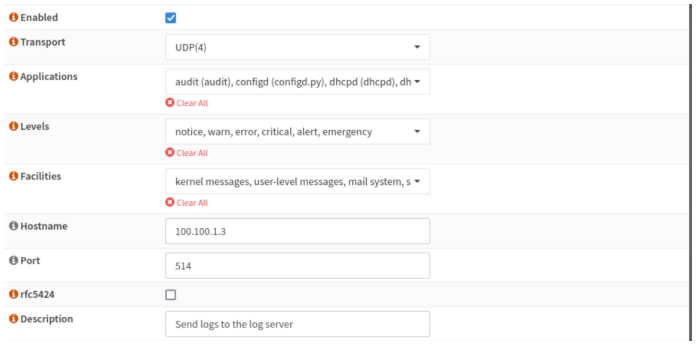


Figure 37: Sending the logs to the Logserver

3.7.2 Control Plane

The Control Plane is related to managing network traffic routing and switching, and it makes decisions about how to forward data packets.

- Limiting and blocking potentially disruptive ICMP traffic, which can be used to perform attacks such as DoS and address spoofing. We go to `Firewall` → `Shaper` → `Settings`, from here we can use the "Pipes" and the "Rules" tabs to add a limitation to `10 Kbit/s` on incoming ICMP traffic from the WAN interface of the main firewall.

Figure 38: In the "Pipes" tab, add limitation without selecting mask

Figure 39: In the "Rules" tab, add a rule for the WAN

- Block all the ICMP redirect packets, which are often used for malicious purposes. ICMPv4 redirects are blocked by default by OPNsense, so we just needed to add a rule regarding ICMPv6 redirects, in the Floating table of both firewalls (it has to be applied to each interface).

Figure 40: Firewall rule to block ICMPv6 redirects

- Block all ICMPv4 type Destination Unreachable messages in outbound direction from the WAN interface of the main firewall.

Figure 41: Block ICMPv4 Destination Unreachable messages

3.7.3 Data Plane

The Data Plane is responsible for the actual forwarding of data packets based on the decisions made by the Control Plane. It handles the user traffic moving through the network. Similar to many other firewalls, the Data Plane in OPNsense inherently protects itself by implementing traffic filtering rules. In addition to the automatic ones, we have already extensively configured these rules during the first assignments.

For instance: allow access in the DMZ only to packets that require the services provided by either the Webserver or the Proxyserver. To accomplish this, we can find the following rules to the WAN interface of the main firewall:

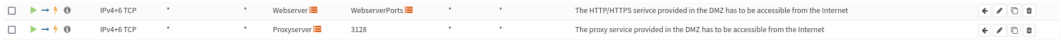


Figure 42: Rules from the WAN interface of the main firewall

4 Final Remarks

It was very fascinating to apply the theoretical concepts discussed in lectures to practical scenarios. The most challenging service to implement for us was definitely Fail2ban, particularly the process of integrating it with our OPNsense firewalls. If we had more time at our disposal, we would have continued the network hardening process by configuring a Network Intrusion Detection System, such as Suricata (supported by OPNsense), and implementing link-local attack protections, such as arpwatch.