ACME-29
Assignment 3 Report

Edoardo Gabrielli (1693726), Davide Quaranta (1715742), Alessio Tullio (1809077)

Initial brainstorming

The ACME network now needs a **forward proxy** and a **reverse proxy**. The former - installed on the Proxy machine - is needed to allow internal clients to access external web services, and the latter - in the Web Server machine - is needed to create a security layer on top of the external **coffee machine.** which has some vulnerabilities (like the weak admin `Passw0rd`).

We use **Squid** to implement the forward proxy, which is also integrated in Zentyal. Later in the report we explain why we decided not to use Zentyal but to configure Squid manually.

In any case, the **forward proxy** should accept, forward and log HTTP requests, and clients should be **authenticated**. The proxy should be the only way to navigate the web. All of that eventually means that Squid needs to be configured for such requirements, and the firewall rules should enforce the policy (already done in previous assignments).

The **reverse proxy** is intended to wrap the webserver of the `fantasticcoffe` machine, should provide **HTTPS** support and act as a **WAF**[1].

Forward proxy configuration

Squid can be enabled and configured either as a Zentyal module or as a standalone program. Unfortunately the Zentyal module specifically requires to have at least one network card configured as a WAN gateway, which is something that we don't have and won't have.

For this reason we opted to uninstall everything related to Squid in the Zentyal interface, and **manually configured Squid** in `/etc/squid/forward.conf` as follows:

[1] WAF stands for Web Application Firewall, specific to filter HTTP(S) traffic.

```
# Port ACLs
acl SSL_ports port 443
acl Safe_ports port 80

# Network ACLs
acl allowed_nets src 100.100.2.0/24
acl allowed_nets src 2001:470:b5b8:1df2::/64

# Authentication settings
auth_param digest program /usr/lib/squid/digest_file_auth -c /etc/squid/users
auth_param digest children 5
auth_param digest realm forward-proxy

acl users proxy_auth REQUIRED

# Policies (allow, deny)
http_access deny !Safe_ports
http_access deny !allowed_nets
http_access allow allowed_nets users
http_access deny all

# Privacy enhancements
forwarded_for delete

# General
http_port 3128
cache_mem 100 MB
coredump_dir /var/spool/squid3

```

Specifically, with this configuration we are instricuting Squid to:

- Use the digest authentication method for the users defined in `/etc/squid/users`.
- Set the user authentication mandatory.
- Only allow the defined users coming from the Clients network, thus denying everything not specifically allowed.
- Remove the `X-Forwarded-For` header to hide from the remote destinations the IP address of the originating client.
- Listen on port 3128.
- Have an in-memory cache of 100 MB max.

Unless defined otherwise, Squid **logs** requests in `/var/log/squid/access.log`.

Users with their respective passwords can be created with Apache's **htdigest**[2] utility; as an example, to create the user `nina` in the `forward-proxy` realm is sufficient to run

```
$ htdigest /etc/squid/users forward-proxy nina
```

[2] https://httpd.apache.org/docs/2.4/programs/htdigest.html

Then the password is taken from the stdin.

Reverse proxy configuration

To set up the Apache reverse proxy you have to activate the needed modules, configure the **virtual hosts** for both HTTP and HTTPS, create SSL **certificates** and a Diffie-Hellman group, then configure the **WAF**.

#1 Activate the required modules

The required `proxy`, `proxy_http`, `proxy_html`, `proxy_connect` modules can be activated with the `a2enmod` utility.

#2 Edit the virtual hosts

It is needed to create a **virtual host** listening on port `443`, with contents:

```
<VirtualHost *:443>
      ProxyPreserveHost On
      ServerName localhost
      SSLEngine On
      SSLCertificateFile /etc/apache2/ssl/certs/cert.crt
      SSLCertificateKeyFile /etc/apache2/ssl/cert.key

      ErrorLog ${APACHE_LOG_DIR}/error.log
      CustomLog ${APACHE_LOG_DIR}/access.log combined

      ProxyPass /coffee/ http://100.100.4.10/
      ProxyPassReverse /coffee/ http://100.100.4.10/
</VirtualHost>
```

This essentially proxies via **HTTPS** the coffee machine at http://100.100.4.10, with a self-signed certificate.

To **redirect the HTTP traffic to the HTTPS** version is sufficient to create another virtual host listening on port 80, with the following directive:
`Redirect "/coffee/" "https://100.100.6.2/".`

Self-signed certificates can be created with:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/apache2/ssl/private/apache-selfsigned.key -out
/etc/apache2/ssl/certs/apache-selfsigned.crt
```

The Diffie-Hellman group can be created with:

```
$ openssl dhparam -out /etc/apache2/ssl/certs/dhparam.pem 2048
```

It is important to put symlinks in `/etc/apache2/sites-enabled/` to point to the configurations done earlier in `/etc/apache2/sites-available/`.

Protection of the coffee machine

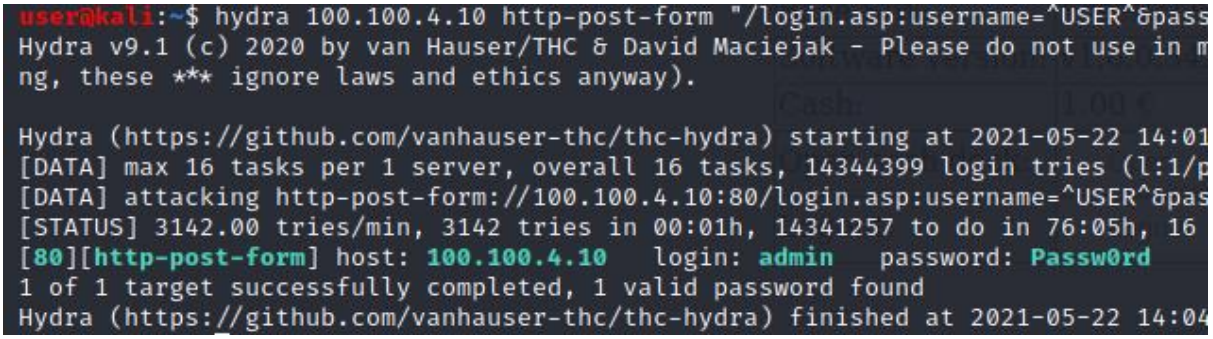
Vulnerabilities

The coffee machine has several vulnerabilities. In fact it has a weak admin password and allows some content disclosure.

Weak administrator password

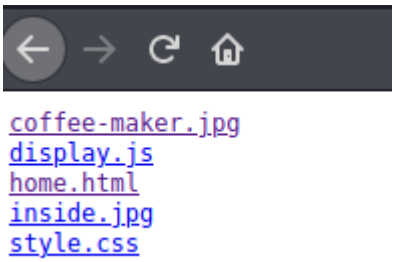
The default password for the `admin` user is **`Passw0rd`**, and can be **easily cracked** with brute force tools like hydra:

```
$ hydra 100.100.4.10 http-post-form
"/login.asp:username=^USER^&password=^PASS^:Wrong" -P
wordlists/rockyou.txt -l admin
```



Directory listing enabled

With web enumeration scanning tools it is possible to find that the custom web server allows directory listing, for example it is possible to see the contents of the `/assets` directory.



The home's template is readable

Following the previous vulnerability, it is possible to read the Go template for the home page, which is `/assets/home.html`. For some reason the code is Go but the extension is `.asp`.

Configuration of the WAF

In order to protect the coffee machine from the vulnerabilities listed above, we can configure Apache's `mod_security`.

The Apache modsecurity is a module that enables a Web Application Firewall, namely

a firewall that operates at the http/https traffic level. `mod_security` rules can be added inside the virtual host configuration.

Block directory listing and home.html

```
SecRule REQUEST_URI "^/coffee/assets/(.*\.html)?$"
id:1005,log,deny,msg:'Web enumeration'
```

Only allow internal IP addresses

Internal connections should be authenticated via the forward proxy.

```
SecRule REQUEST_URI "^/coffee/" "id:'1003',chain,log,deny"
SecRule REMOTE_ADDR "^(?!100\.100\.6\.3$).*$"
```

Block form with unexpected parameters

We should only allow ordering of `coffee`, `tea` and `hotwater`, with sugar `0,1,2,3`.

```
SecRule REQUEST_FILENAME "action.asp"
     "id:'1006',chain,deny,log,msg:'Poduct attack'"
SecRule REQUEST_METHOD "POST" chain
SecRule ARGS:product "@rx ^(?!coffee$|tea$|hotwater$).*"

SecRule REQUEST_FILENAME "action.asp"
     "id:'1007',chain,deny,log,msg:'Sugar attack'"
SecRule REQUEST_METHOD "POST" chain
SecRule ARGS:sugar "@rx ^[^0123].*$|^.{2,}$"
```

Additional security

We enforced the firewalls rules (both in the Main and Internal routers), **blocking** all the HTTP(S) traffic directly destined to the `fantastic_coffee` machine.

Main Firewall Rules

Interface	Direction	Protocol	Source	Destination	Port
ANY	in	IPv4+6 TCP	!proxy_server	fantastic_coffee	HTTP
ANY	in	IPv4+6 TCP	!proxy_server	fantastic_coffee	HTTPS

In the Main we allow only the `proxy_server` to communicate with the `fanstatic_coffee` machine.

Internal Firewall Rules

Interface	Direction	Protocol	Source	Destination	Port
ANY	in	IPv4+6 TCP	*	fantastic_coffee	HTTP
ANY	in	IPv4+6 TCP	*	fantastic_coffee	HTTPS

Tests

In this section we discuss the main tests we did to verify the correctness of the new implemented features.

Forward proxy

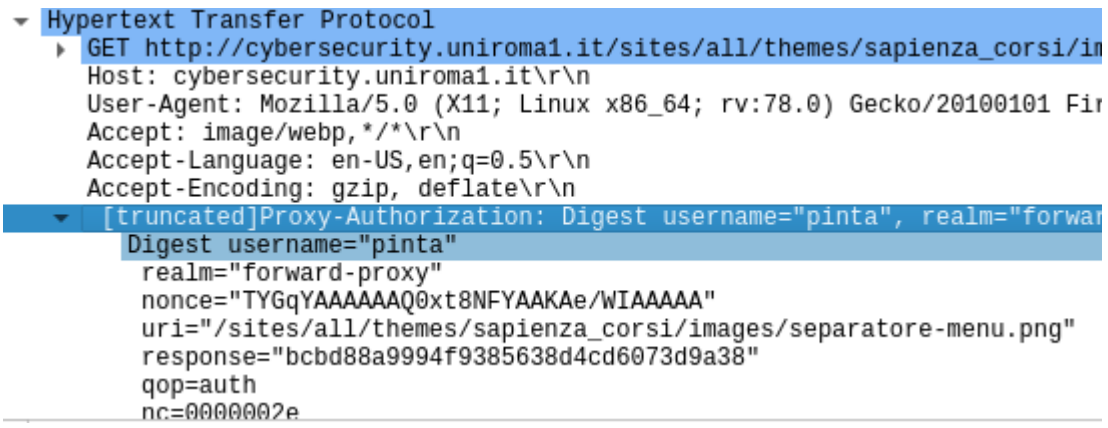
For the forward proxy we had to:

- Try to use the proxy from a host that is not allowed → fail.
- Try to use the proxy to access a port that is not allowed → fail.

Namely, the proxy should only be accessible from the defined authenticated users coming from the `Clients network`, who try to access the allowed ports.

Authentication

HTTP packets from authenticated allowed hosts should have a `Proxy-Authorization` section as shown in the image below:



Logs

Requests should be correctly logged in Squid's access.log. We will discuss how to send these logs to the log-server on assignment#4.

Reverse proxy

The web-server acts as a reverse proxy for the `fantasticcoffee` machine. This is done by accessing the web-server at `https://100.100.6.2/coffee/`.
Through a web browser (or from curl), from the kali machine we executed these tests:

General

- Access the webserver on port 80 → redirect on 443
- Access the webserver on port 443 → ok
- Access the webserver as a forward-proxy-authenticated user → ok
- Access the webserver as a not authenticated host → fail

FantasticCoffee resources

- Access to `/coffee/assets/` → 403 forbidden
- Access to `/coffee/assets/*.html` → 403 forbidden
- Access to `/coffee/` → ok
- Admin login → ok

FantasticCoffee form

- Do `POST /coffee/action.asp` with legal request parameters → ok:
  - `product: {coffee,tea,hotwater}`.
  - `sugar: {0,1,2,3}`.
- Do `POST /coffee/action.asp` with illegal request parameters → fail:
  - `product: {any_other_string}*`.
  - `sugar: {any_other_string}*`
- Do `POST /coffee/action.asp` with mixed legal and illegal parameters → fail

As an example, this is the output of a `curl` call to `action.asp` with a forbidden parameter, such as {**`product=evil`**, `sugar=0`}.

```
user@kali:~$ curl -k -U [redacted] -x http://100.100.6.3:3128 -X
POST "https://100.100.6.2/coffee/action.asp" -d product=evil -d sugar=0 -H
"Cookie: sessionid=Bmik6IXXdUo$" --proxy-digest
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.38 (Debian) Server at 100.100.6.2 Port 443</address>
</body></html>
```

New Firewall Rules

We also tried from all machines to directly access the coffee machine: the firewalls block all direct access.

Final remarks

While realizing the activity we learned about:

- **Squid** authentication mechanisms.
- Apache **virtual host** configuration for proxying.
- **mod_security** configuration.
- mod_security rules syntax and options.

Moreover, securing the coffee machine was a fun activity.

Please refer to the **exported files** for further details on the configuration of all the services, including adjustments to the existing firewall rules.