Homework 3 Report - Group number 27

Name	Student ID	email
Simone di Biasio	1823213	dibiasio.1823213@studenti.uniroma1.it
Leonardo Persiani	1795525	persiani.1795525@studenti.uniroma1.it

- 0. Brainstorming
- 1. SQUID Configuration
  - 1.1 Clients Network ACL
  - 1.2 Authentication
  - 1.3 Listening Port and Allowed Ports
  - 1.4 Caching and Logging
  - 1.5 Blocking everything else
- 2. Reverse Proxy
  - 2.1 HTTPS Redirection
  - 2.2 Proxy Directive
  - 2.3 Modsecurity
    - 2.3.1 Fantasticcoffee and its functionalities
    - 2.3.2 Modsecurity installation and configuration
    - 2.3.3 Modsecurity Rules
- 3.Firewall adjustments
- 4. Test of the configuration
  - 4.1 Forward Proxy Test
  - 4.2 Reverse Proxy Test
  - 4.3 Modsecurity test
- 5. Final Remarks

0. Brainstorming

The idea is to first configure `squid` to forward HTTP and HTTPS requests and then setup the reverse proxy on webserver machine.

1. SQUID Configuration

As Requested, we used `SQUID` as Forward Proxy: incoming requests from hosts in clients network are forwarded to the final endpoint.

1.1 Clients Network ACL

To allow incoming requests, we first defined an `acl` for the Clients Network and allowed traffic coming from them:

```
# Clients Network ACL
acl clients_net src 100.100.2.0/24
acl clients_net src 2001:470:b5b8:1b12::/64

# Requests not coming from clients net cannot proceed further
http_access deny !clients_net
```

This, in conjunction with firewall rules which restrict the proxy requests, ensures that the `fantasticcoffee` machine is reachable only from hosts in Clients Network.

1.2 Authentication

Authentication for clients is provided via `digest` method: it is challenge-based (so doesn't need the password to be sent) and hashes authentication requests using `md5`, contrary to `basic authentication`.

We first created `/etc/squid/passwd` which is the file credential will be stored, then we created the three requested users using `htdigest` command:

```
htdigest /etc/squid/passwd acme27 nina
htdigest /etc/squid/passwd acme27 pinta
htdigest /etc/squid/passwd acme27 maria
```

The `user:password` credentials are simply:

- nina:nina
- pinta:pinta
- maria:maria

Then we specified authentication details in `squid.conf` file:

```
# Authentication using digest
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 acme27
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
```

1.3 Listening Port and Allowed Ports

We decided to make SQUID listen on its default port, which is `3128`

```
http_port 3128
```

Next we defined the allowed ports for incoming proxy requests and defined the CONNECT method for `https` requests:

```
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 443         # https
acl CONNECT method CONNECT
```

And we denied access to any other port/method:

```
# Deny requests to certain unsafe ports
http_access deny !Safe_ports

# Deny CONNECT to other than secure SSL ports
http_access deny CONNECT !SSL_ports
```

1.4 Caching and Logging

We enabled caching and logging by specifying the following directives:

```
# Caching directory.
cache_dir ufs /var/cache/squid 100 16 256

# Logging
access_log daemon:/var/log/squid/access.log squid
```

The `squid` keyword specifies the defaul `logformat`, which is:

```
timestamp duration source_ip  cache_result/server_response_code size method
remote_resource user hierarchy_route/dest_ip content_type
```

1.5 Blocking everything else

Since we do not expect any connection coming from other hosts, we simply denied access not coming from clients net:

```
acl all src all
[...]
# And finally deny all other access to this proxy
http_access deny all
```

2. Reverse Proxy

The reverse proxy was implemented using `apache2`'s builtin proxy redirection functionality, while the `modsecurity` feature had to be installed via `apt`:

```
sudo apt install libapache2-mod-security2
```

All the other functions were enabled by editing the apache configuration files.

2.1 HTTPS Redirection

In order to offer HTTPS functionality to the fantasticcoffee machine, the `webserver` machine had to act as the `SSL` endpoint: the first thing to do was generate certificates and keys:

- first generate the private key: `openssl genrsa -out ca.key 2048`
- then generate a Certificate Signing Request: `openssl req -new -key ca.key -out ca.csr`
- and finally the certificate: `openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt`

Then move the Certificate and the Private key to the `/etc/ssl` directory.

Then it was time to enable the HTTPS endpoint by making the server listen on port 443. In `/etc/apache2/sites-enabled/000-default.conf`:

```
<VirtualHost *:443>
    ServerName webserver.acme27.com

    SSLEngine on
    SSLCertificateFile "/etc/ssl/certs/ca.crt"
    SSLCertificateKeyFile "/etc/ssl/private/ca.key"
</VirtualHost>
```

2.2 Proxy Directive

Next we enabled the core functionality of the reverse proxy, the redirection: in the same `000-default.conf` file, we added:

```
<VirtualHost *:443>
    ServerName webserver.acme27.com

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

    ProxyPass "/" "http://100.100.4.10:80"
```

```
    ProxyPassReverse "/" "http://100.100.4.10:80"

</VirtualHost>
```

Every request coming on port 443 will be redirected to the `fantasticcoffee` machine.

2.3 Modsecurity

In order to properly secure the `fantasticcoffee` machine, we first had to understand how it worked. To do so, we reverse-engineered it.

2.3.1 Fantasticcoffee and its functionalities

The main page of the `fantasticcoffee` machine doesn't offer much, but we see a login form. After a few attempts, and thanks to some hints given during class, we found valid credentials: `admin:Passw0rd`.

After logging in, the website allows the admin to either make a drink or empty the cash dock. By playing around with `Burp` we found out that:

- the only supported beverages are `coffee`, `tea` and `hotwater`
- the allowed amounts of sugar are either 0, 1, 2 or 3
- the `/open-cash.asp` endpoint supports only `GET` requests

2.3.2 Modsecurity installation and configuration

We installed `modsecurity` via `apt install libapache2-mod-security2` and then we proceeded to configure it.

First we enabled its engine by setting `SecRuleEngine` to `On` in `/etc/modsecurity/modsecurity.conf`. Then we specified the rules path via `Include /etc/modsecurity/fanstasticcoffee_rules.conf` And finally we enabled the `SecRuleEngine` for our HTTPS endpoint in `/etc/apache2/sites-available/000-default.conf`

2.3.3 Modsecurity Rules

We specified our rules for fantasticcoffee in `/etc/modsecurity/fantasticcoffee_rules.conf`:

- We want to allow only two parameters to `/action.asp` endpoint, and we want them to be `product` and `sugar`:

```
# 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)$" ""
```

- We want the values to be as described above

```
# 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)$" ""
```

- Allow only `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" ""
```

3.Firewall adjustments

The setup would not work without modifying firewall rules, and in particular we had to:

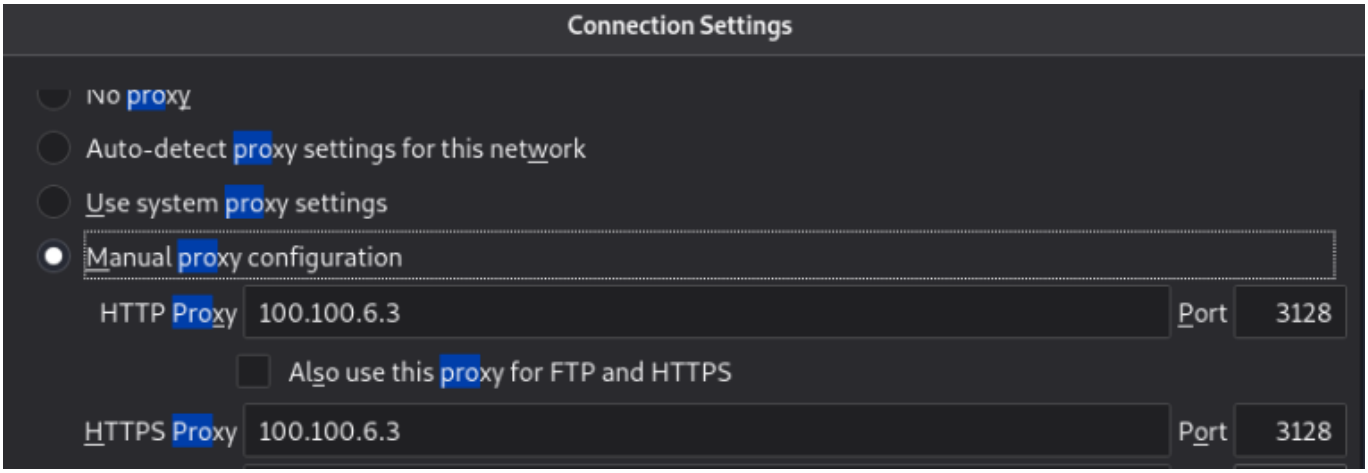
1. Edit proxy rules on Internal Firewall: change the allowed port from `80` and `443` to SQUID's default `3128`
2. Allow HTTP traffic from `webserver` towards `fantasticcoffee` in Main Firewall

4. Test of the configuration

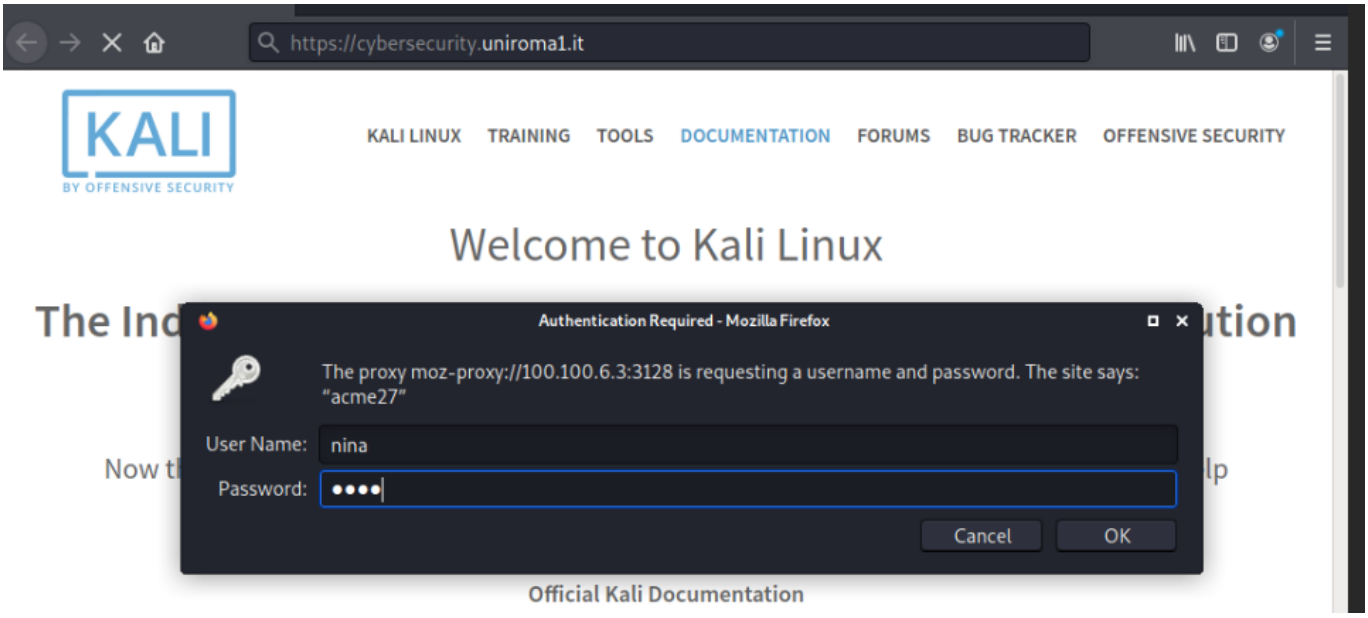
For the testing phase we used `tcpdump`, `wireshark` and `curl`

4.1 Forward Proxy Test

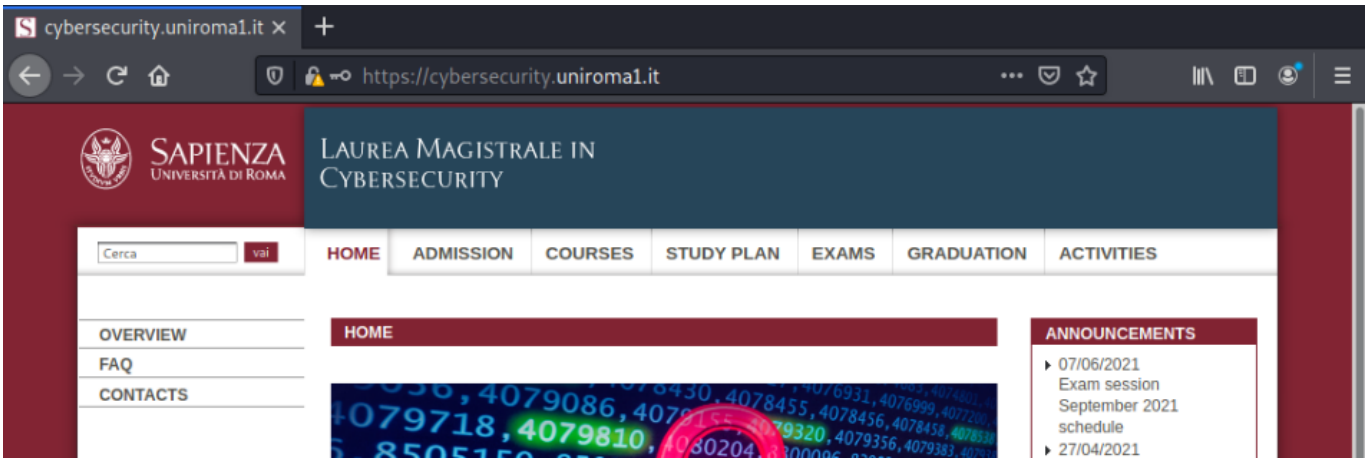
We tested the forward proxy functionality from the `kali` machine in clients network, both from `firefox` and command line interface. In `firefox` settings specify the proxy address and port:



When we request a page we're prompted for credentials:



Then we get the correct page:



We can see the traffic flow in `wireshark` too:

(http or tls) and not tcp contains "mozilla"

No.	Time	Source	Destination	Protocol	Length	Info
4	0.003043	100.100.2.100	100.100.6.3	HTTP	289	CONNECT cybersecurity.uniroma1.it:443 HTTP/1.1
9	0.003472	100.100.6.3	100.100.2.100	HTTP	481	HTTP/1.1 407 Proxy Authentication Required (text/html)
14	5.138311	100.100.2.100	100.100.6.3	HTTP	525	CONNECT cybersecurity.uniroma1.it:443 HTTP/1.1
18	5.141140	100.100.6.3	100.100.2.100	HTTP	105	HTTP/1.1 200 Connection established
20	5.152761	100.100.2.100	100.100.6.3	TLSv1.3	579	Client Hello
21	5.152858	100.100.6.3	151.100.17.12	TLSv1.3	579	Client Hello
23	5.160008	151.100.17.12	100.100.6.3	TLSv1.3	1414	Server Hello, Change Cipher Spec, Application Data
25	5.160109	100.100.6.3	100.100.2.100	TLSv1.3	1334	Server Hello, Change Cipher Spec, Application Data
26	5.160124	151.100.17.12	100.100.6.3	TLSv1.3	1047	Application Data, Application Data, Application Data
29	5.160167	100.100.6.3	100.100.2.100	TLSv1.3	1047	Application Data, Application Data, Application Data
33	5.182869	100.100.2.100	100.100.6.3	TLSv1.3	146	Change Cipher Spec, Application Data
34	5.182985	100.100.6.3	151.100.17.12	TLSv1.3	146	Change Cipher Spec, Application Data
35	5.183710	100.100.2.100	100.100.6.3	TLSv1.3	236	Application Data
37	5.183767	100.100.6.3	151.100.17.12	TLSv1.3	236	Application Data
38	5.184139	100.100.2.100	100.100.6.3	TLSv1.3	313	Application Data
39	5.184188	100.100.6.3	151.100.17.12	TLSv1.3	313	Application Data
40	5.185263	151.100.17.12	100.100.6.3	TLSv1.3	337	Application Data
41	5.185285	151.100.17.12	100.100.6.3	TLSv1.3	337	Application Data
43	5.185360	100.100.6.3	100.100.2.100	TLSv1.3	608	Application Data, Application Data
44	5.185675	151.100.17.12	100.100.6.3	TLSv1.3	128	Application Data
45	5.185715	100.100.6.3	100.100.2.100	TLSv1.3	128	Application Data

In this case a `HTTPS` connection is established with the `CONNECT` method.

We can also see that the forward proxy also works on IPv6 (we used manual name resolution in wireshark for clarity):

not dns and (http or tls)

No.	Time	Source	Destination	Protocol	Length	Info
4	0.002847	kali.ipv6	proxy.ipv6	HTTP	210	CONNECT pkg.opnsense.org:443 HTTP/1.1
10	0.004452	proxy.ipv6	kali.ipv6	HTTP	453	HTTP/1.1 407 Proxy Authentication Required (text/html)
16	0.008965	kali.ipv6	proxy.ipv6	HTTP	465	CONNECT pkg.opnsense.org:443 HTTP/1.1
20	0.041154	proxy.ipv6	kali.ipv6	HTTP	125	HTTP/1.1 200 Connection established
22	0.055019	kali.ipv6	proxy.ipv6	TLSv1.2	603	Client Hello
23	0.055144	proxy.ipv6	opnsense.ipv6	TLSv1.2	603	Client Hello
25	0.085650	opnsense.ipv6	proxy.ipv6	TLSv1.2	1414	Server Hello
27	0.085762	proxy.ipv6	kali.ipv6	TLSv1.2	1334	Server Hello
39	0.087734	opnsense.ipv6	proxy.ipv6	TLSv1.2	449	Certificate, Server Key Exchange, Server Hello Done
41	0.087819	proxy.ipv6	kali.ipv6	TLSv1.2	449	Certificate, Server Key Exchange, Server Hello Done
49	0.091556	kali.ipv6	proxy.ipv6	TLSv1.2	171	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
50	0.091670	proxy.ipv6	opnsense.ipv6	TLSv1.2	171	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
52	0.121026	opnsense.ipv6	proxy.ipv6	TLSv1.2	129	Change Cipher Spec, Encrypted Handshake Message
53	0.121109	proxy.ipv6	kali.ipv6	TLSv1.2	129	Change Cipher Spec, Encrypted Handshake Message
54	0.128164	kali.ipv6	proxy.ipv6	TLSv1.2	187	Application Data
55	0.128244	proxy.ipv6	opnsense.ipv6	TLSv1.2	187	Application Data
57	0.157335	opnsense.ipv6	proxy.ipv6	TLSv1.2	1414	Application Data
58	0.157441	proxy.ipv6	kali.ipv6	TLSv1.2	1334	Application Data

```
▸ Frame 23: 603 bytes on wire (4824 bits), 603 bytes captured (4824 bits)
▸ Ethernet II, Src: 86:10:9b:35:52:5c (86:10:9b:35:52:5c), Dst: 46:fd:65:1b:88:95 (46:fd:65:1b:88:95)
▾ Internet Protocol Version 6, Src: proxy.ipv6 (2001:470:b5b8:1b06:8410:9bff:fe35:525c), Dst: opnsense.ipv6 (2001:1af8:4f00:a005:5::)
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0101 1011 1100 0001 1100 = Flow Label: 0x5bc1c
    Payload Length: 549
    Next Header: TCP (6)
    Hop Limit: 64
    Source Address: proxy.ipv6 (2001:470:b5b8:1b06:8410:9bff:fe35:525c)
    Destination Address: opnsense.ipv6 (2001:1af8:4f00:a005:5::)
```

4.2 Reverse Proxy Test

We can test this by using `curl` on the `kali` machine:

```
curl -v -k https://webserver.acme27.com  --proxy-anyauth --proxy-user
pinta:pinta --proxy http://proxy.acme27.com:3128
```

The request will go through the forward proxy and then will reach the `webserver`, which will forward the request to `fantasticcoffee`.

29	1.394100	proxy.ipv6	webserver.ipv6	TLSv1.3	603	Client Hello
31	1.396871	webserver.ipv6	proxy.ipv6	TLSv1.3	1594	Server Hello, Change Cipher Spec, Application
33	1.399437	proxy.ipv6	webserver.ipv6	TLSv1.3	166	Change Cipher Spec, Application Data
35	1.399542	proxy.ipv6	webserver.ipv6	TLSv1.3	192	Application Data
37	1.399660	webserver.ipv6	proxy.ipv6	TLSv1.3	405	Application Data
38	1.399752	webserver.ipv6	proxy.ipv6	TLSv1.3	405	Application Data
43	1.402221	webserver.acme27.com	fantasticcoffee	HTTP	305	GET / HTTP/1.1
47	1.403579	fantasticcoffee	webserver.acme27.com	HTTP	650	HTTP/1.1 200 OK (text/html)
49	1.404908	webserver.ipv6	proxy.ipv6	TLSv1.3	2095	Application Data

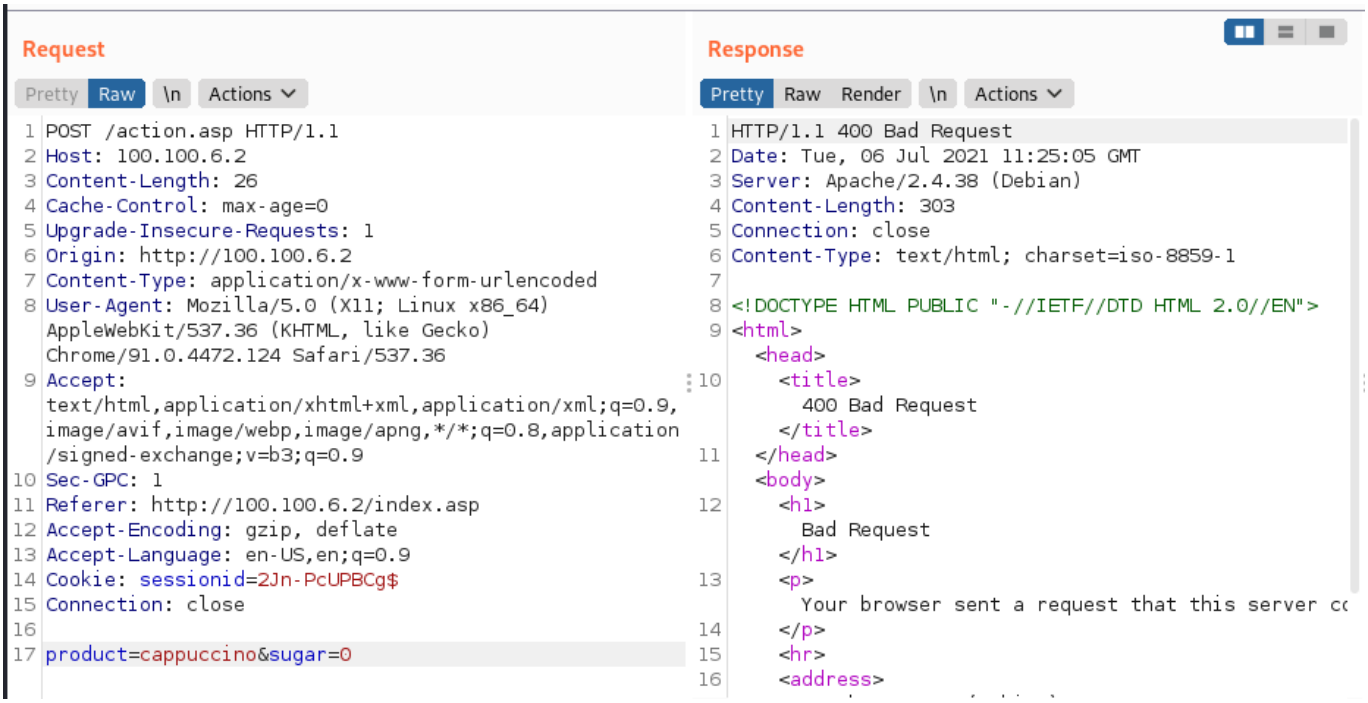
Here we can also see the `TLS` tunnel between the `webserver` and the `kali` machine (not present in capture)

4.3 Modsecurity test

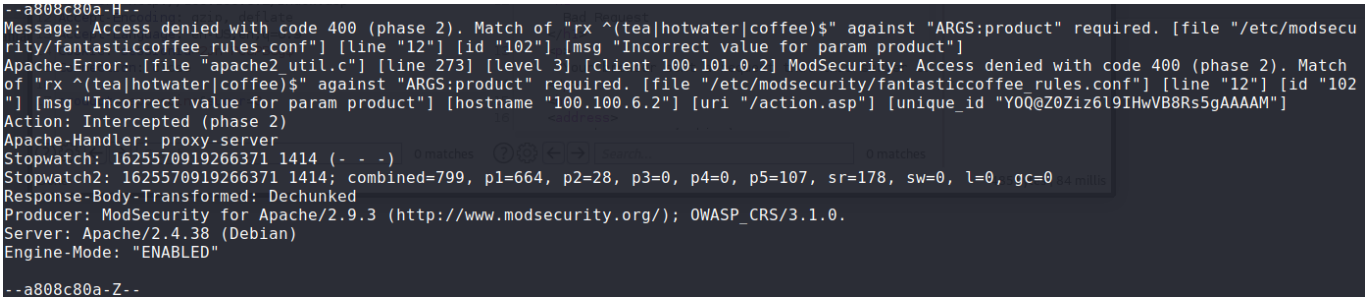
In order to test `modsecurity` we crafted some incorrect requests using `burp`.

In response to valid request `fantasticcoffee` sends back a `302 Found` and then redirects us to the correct page:

If we change some parameters, we trigger a modsecurity rule. For example, if we ask for a `cappuccino`: we get a `400 Bad Request`



and the violation has been logged:



5. Final Remarks

This assignment was overall simple to implement, except for `modsecurity` rules which have a syntax that isn't clear at first sight and required some documentation scraping.

We want to point out that the reverse proxy also supports IPv6 but we couldn't completely test it since `proxy` and clients in clients network support it while `fantasticcoffee` doesn't.