**Name: Omkar Keluskar NetID: ork216**

**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:

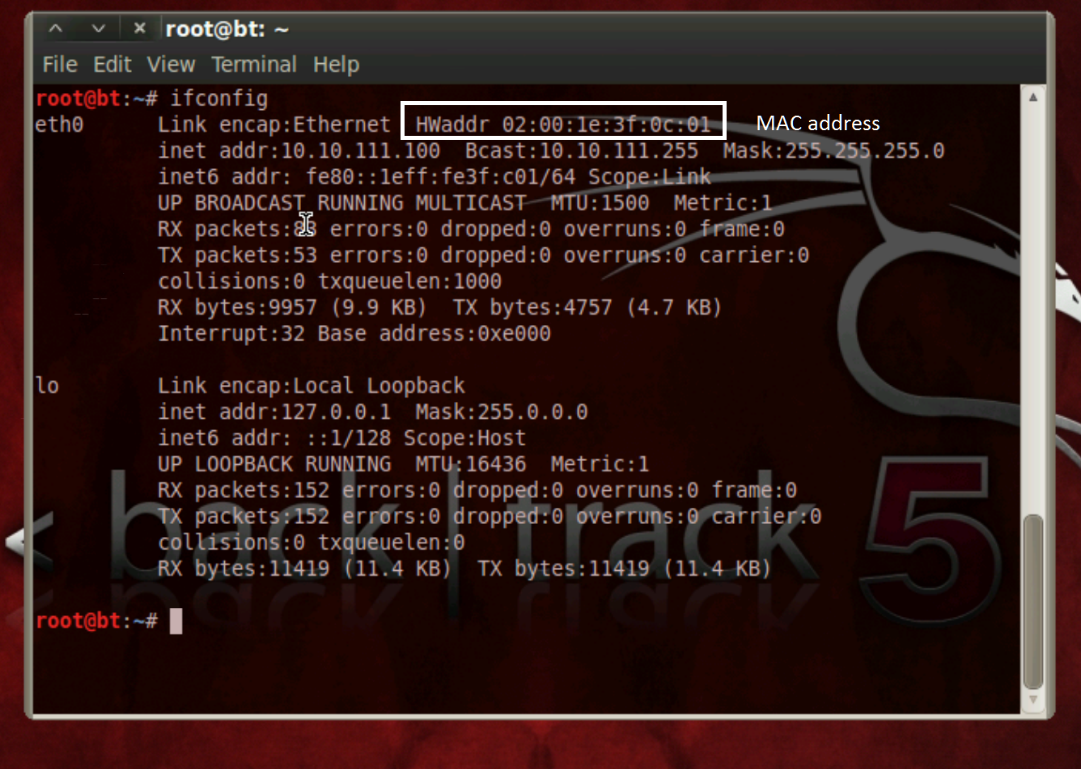
echo "1" > /proc/sys/net/ipv4/ip\_forward

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

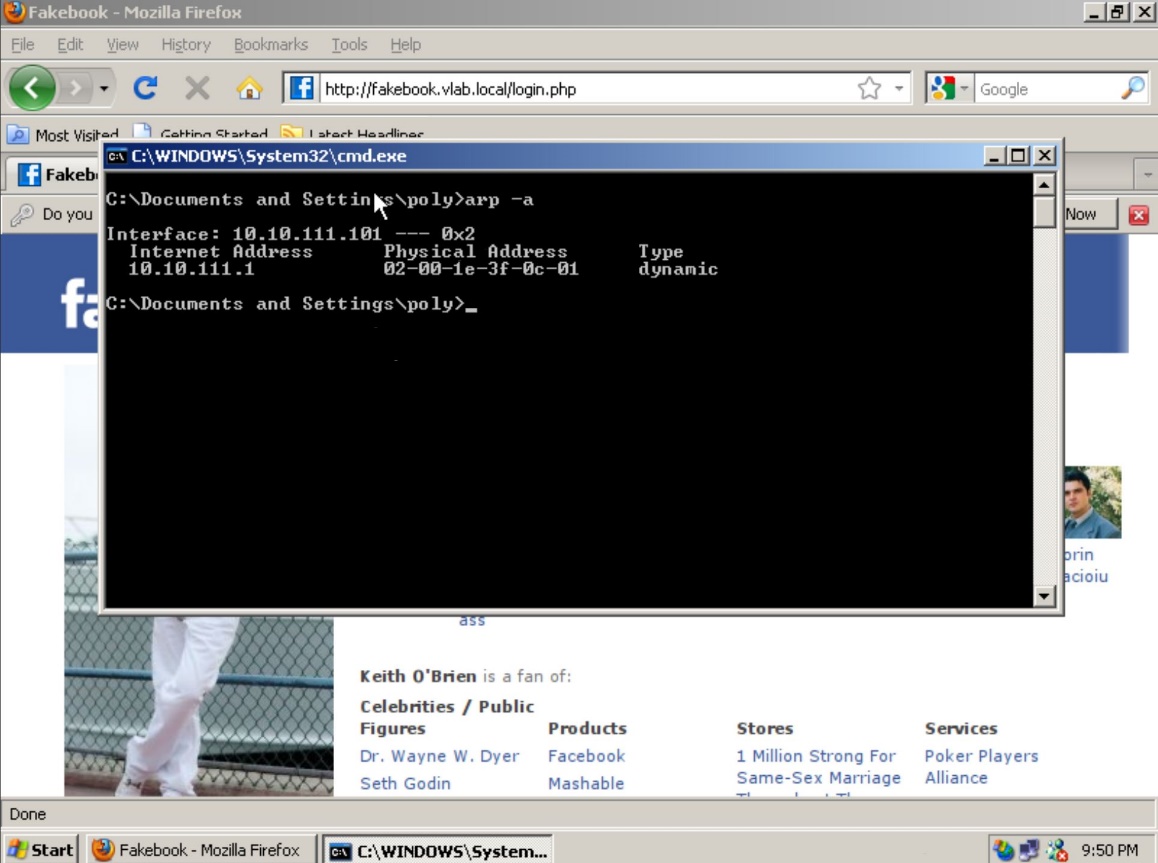
After that, we run the python/scapy file at the terminal to send the gratuitous ARP messages as mentioned above.

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

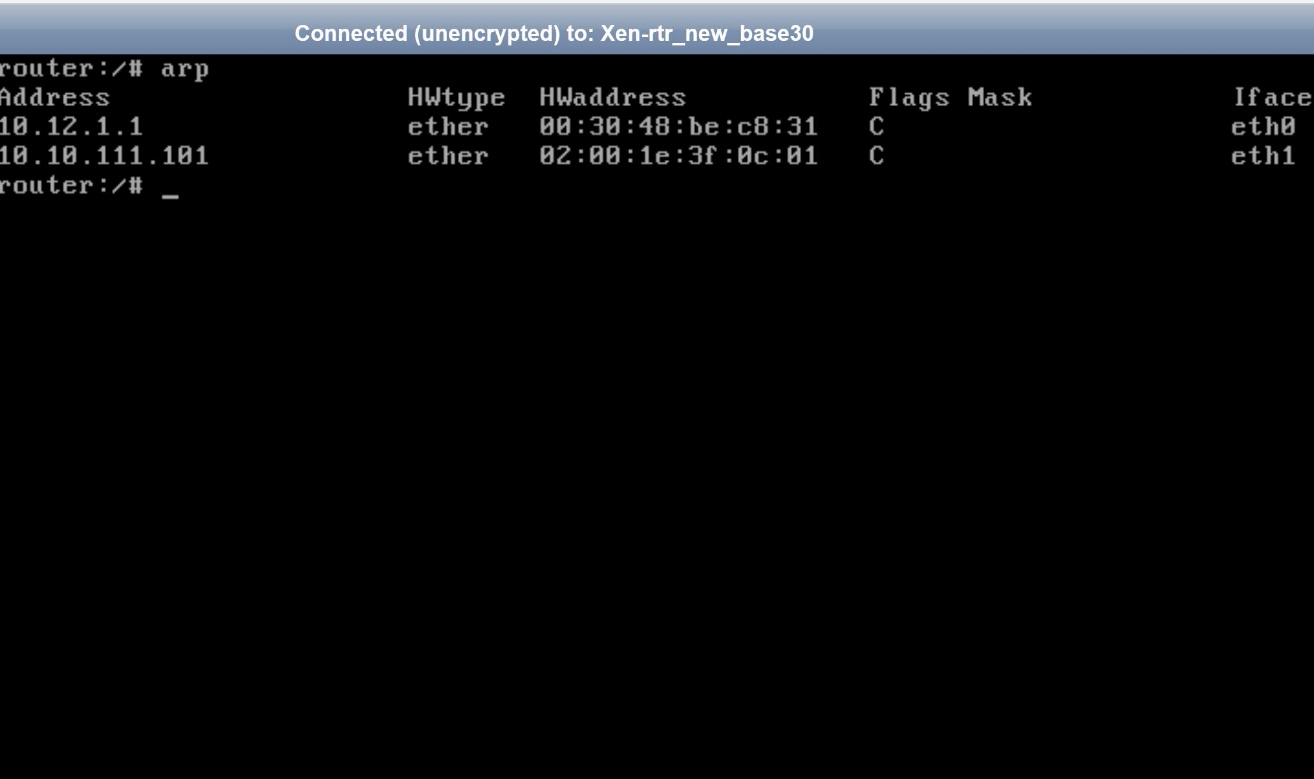




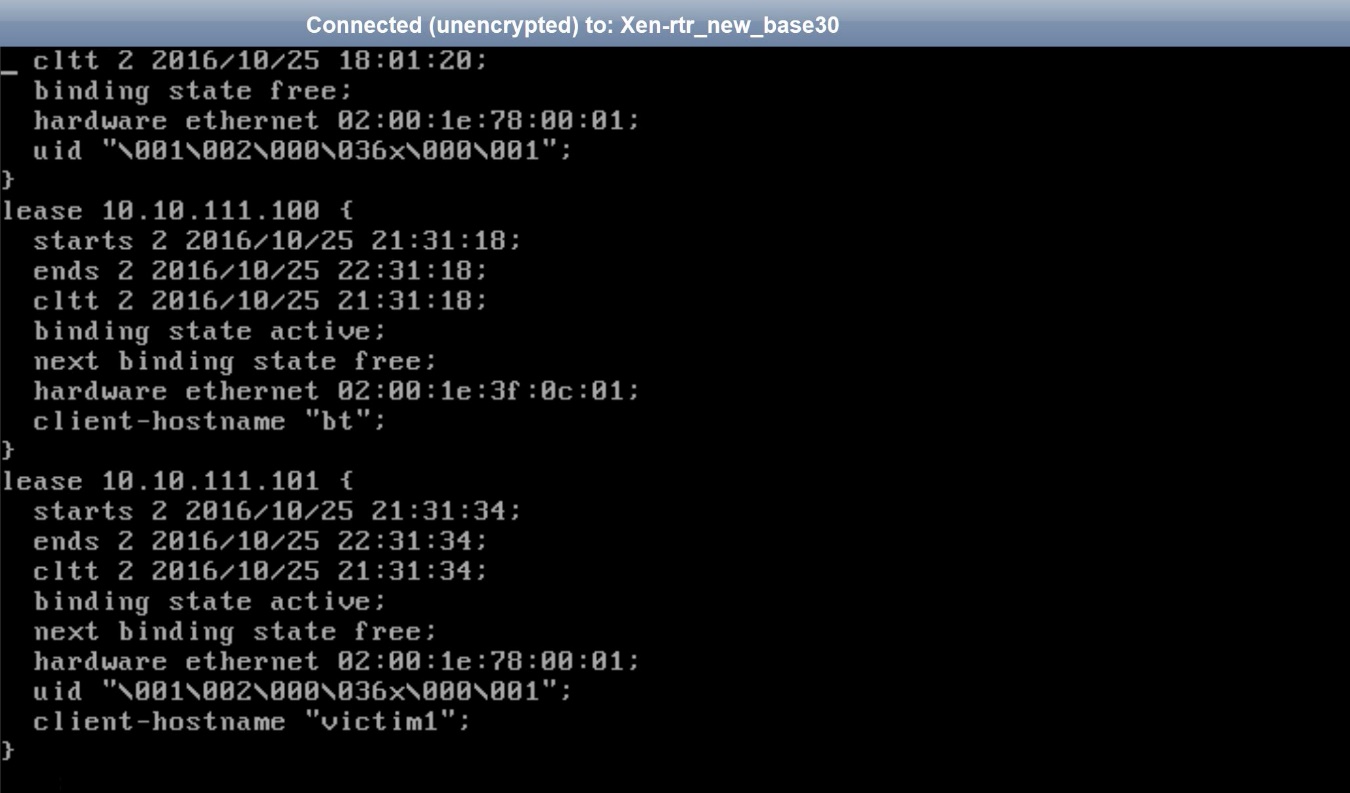
Below is the screenshot from XP machine, where the rtr’s logical address (IP address) is poisoned with bt5’s physical address (MAC address)



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



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.

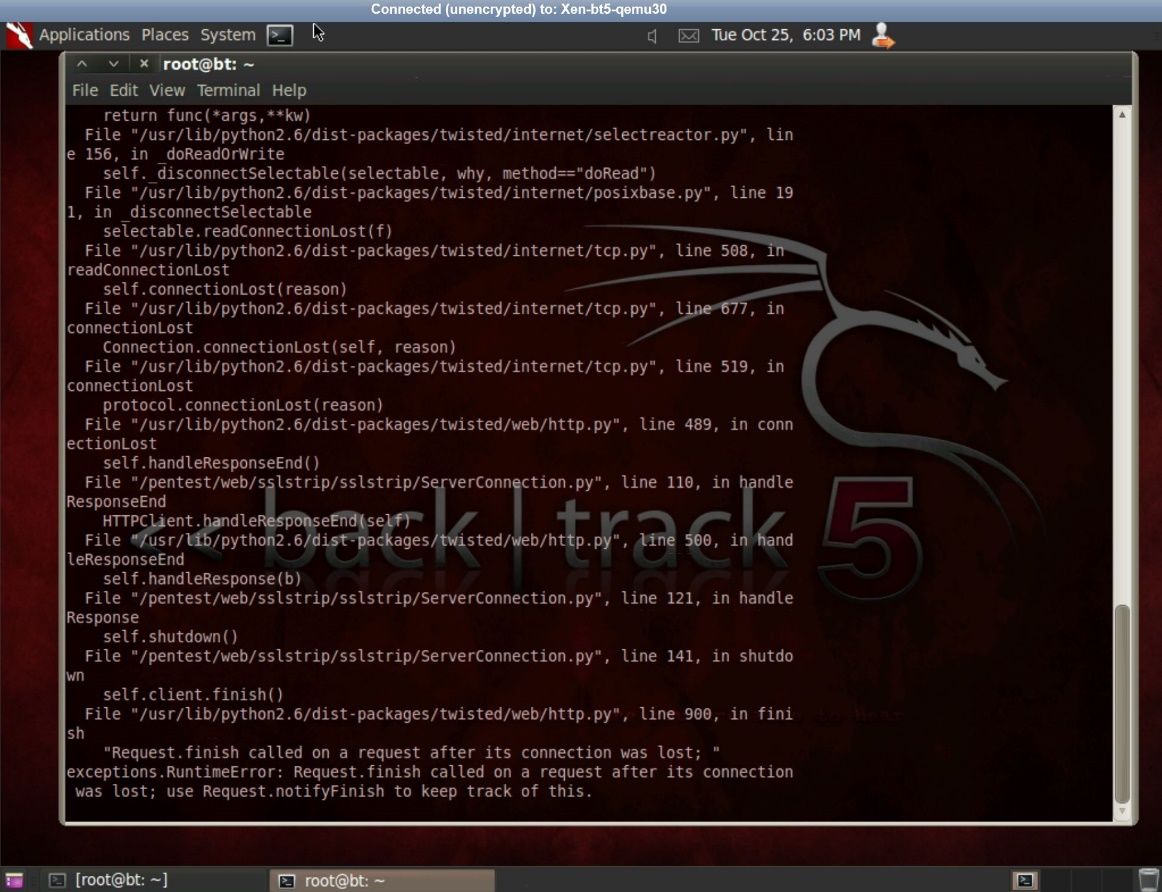


**Step 3:** sslstrip was launched listening to port 8080, where it just took the *https* request for the *login* page from the webserver and provided the *http* version to the client, as shown in the screenshots below. Below is the command line prints for sslstrip’s live listening.

sslstrip was enabled using the following command:

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

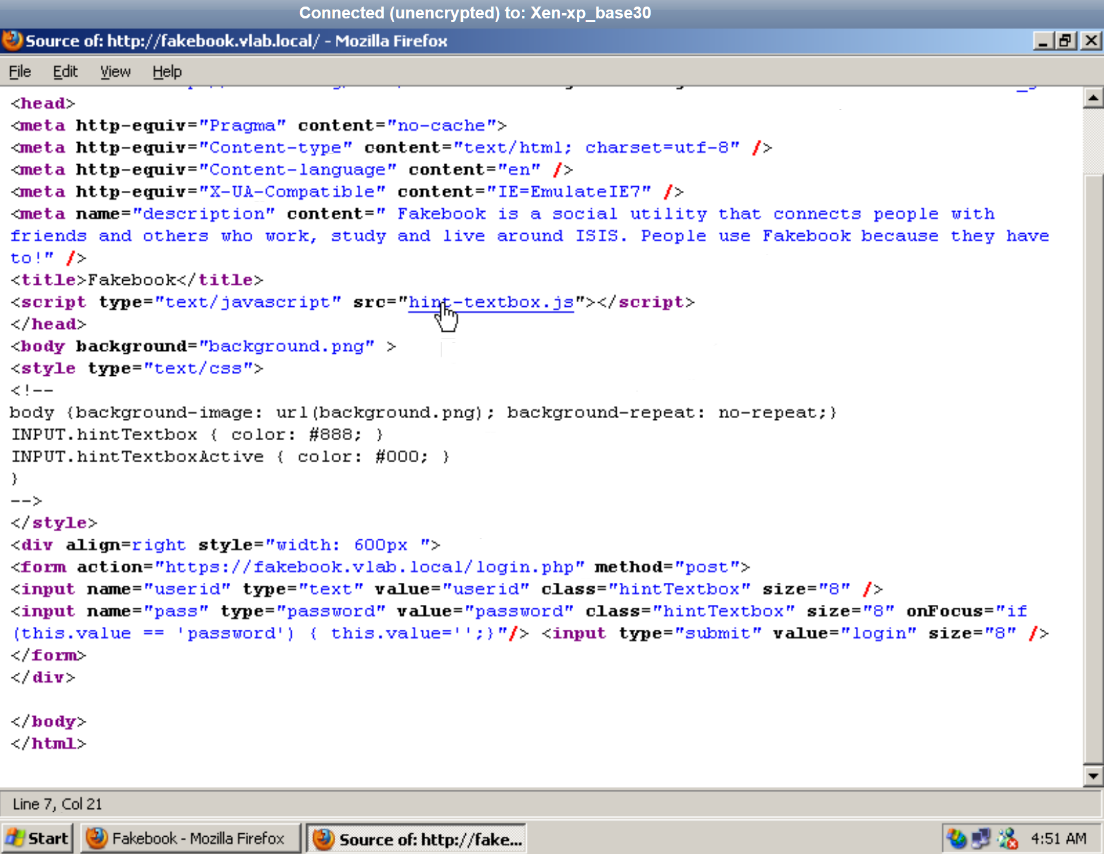
After this, magic happens (kidding), the sslstrip does the job of taking the https traffic from server and providing the victim machine unsecured(http) version of traffic.



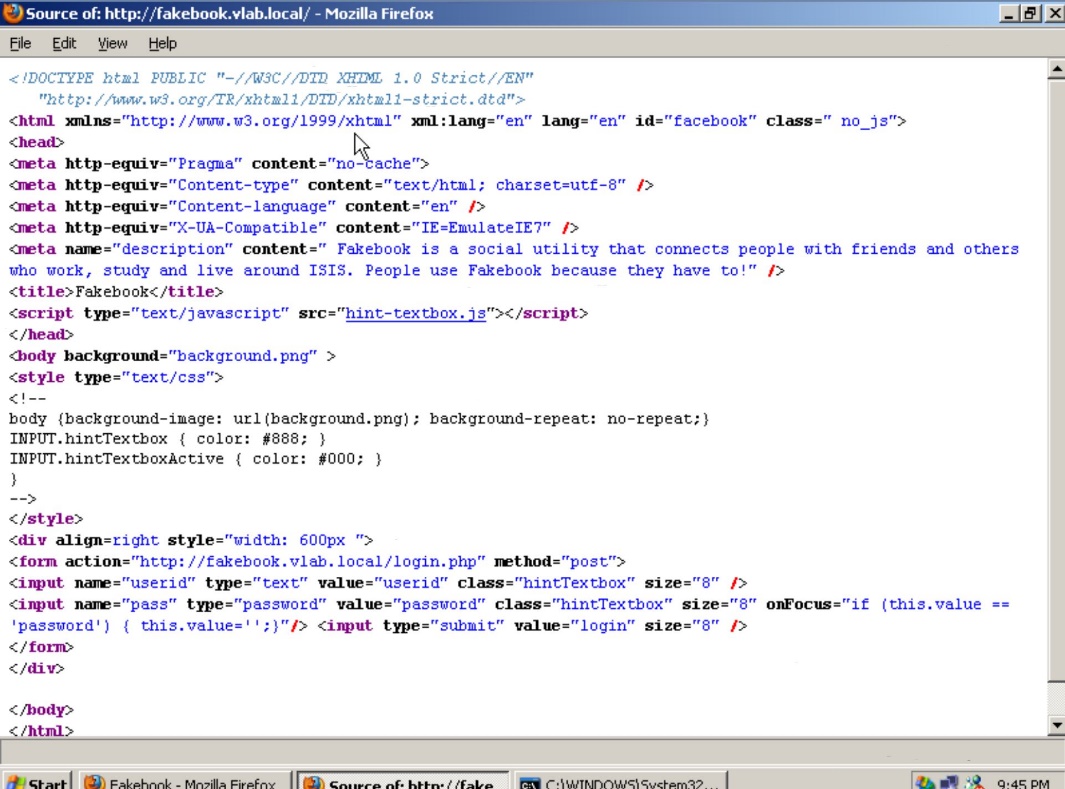
Interestingly, the victim has no idea whatsoever of this. In the meantime, victim accesses the fakebook webserver as usual, logging in to the webserver using the credentials over the http connection. But in the background the attacker can access information from CSS, JavaScript, credentials in form method.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

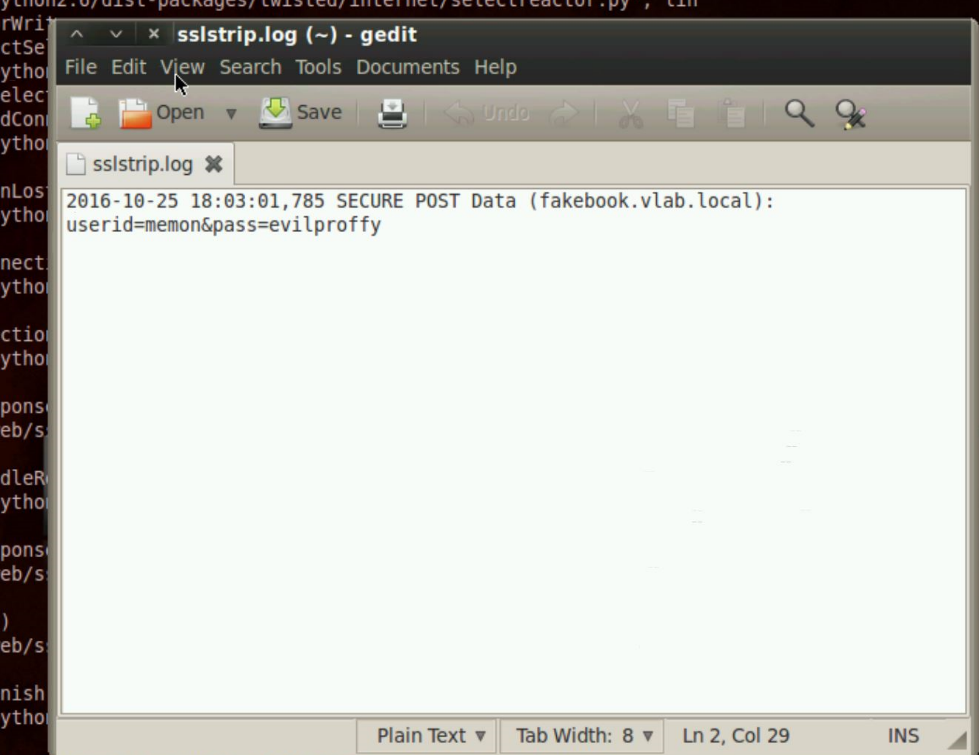


**Before**



**After**

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



**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.