COMP8505 Assignment 3

DNS Spoofer

Harjinder Daniel Khatkar

A00746060

October 30, 2012

Introduction

The purpose of this assignment was to create a DNS spoofer in Ruby. To do this I had to combine our ARP spoofing, packet sniffing and raw sockets techniques I have already learned. The ARP spoofing tools used were similar to the code we used in class, with some modifications. In my DNS spoofer, I generated two threads, one for the ARP spoofing of the network and one for the DNS spoofer. The ARP spoofer allows us to capture all of the packets from the target machine whereas the DNS spoofer will capture DNS packets, handle queries and feed a response back to our victim machine. The response would then redirect the victims DNS requests, in a browser for example, to the main page of Facebook.

Application

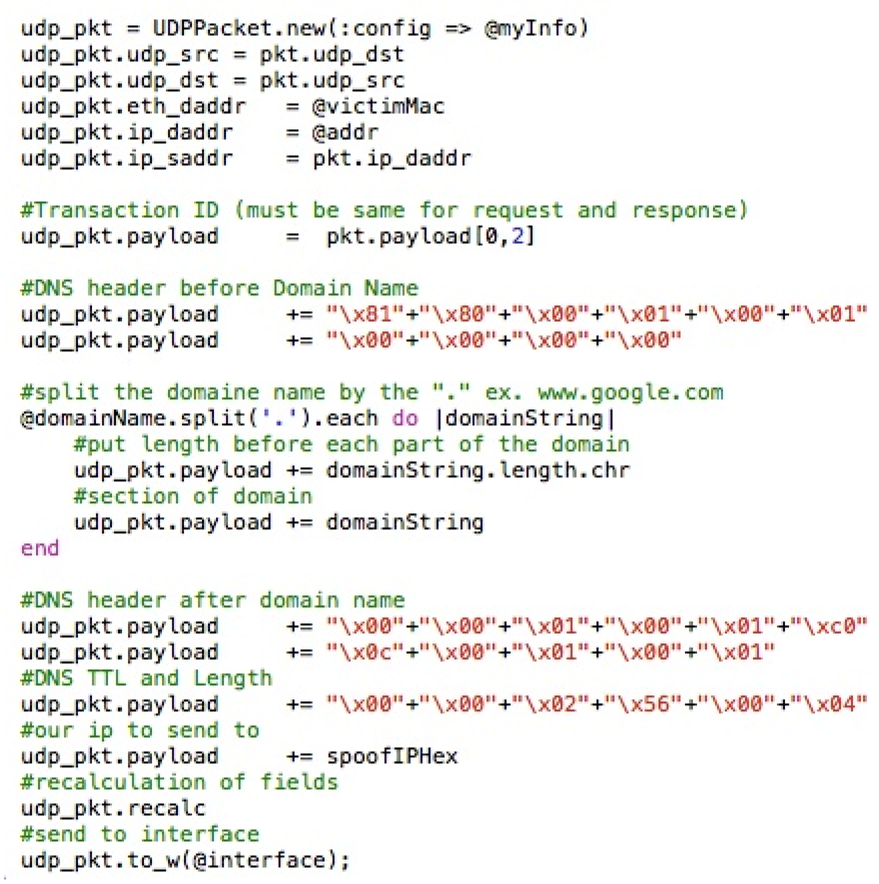
For my ARP spoofer I used the code provided made one major change. Using the PacketFu library you are able to get the MAC address of an IP address using the Utils.Arp function, which is what I did instead of typing them manually. The ARP spoofing method then sends the generated target arp packet and router arp packet with my MAC address, allowing my computer to become a hop between the target machine and the router. This allows me to see all the traffic from the target machine.

For the DNS spoofer I had to capture DNS request packets from the target machine then send spoof a response. To do this, I used the PacketFu library which allows us to capture packets at the card level. I set a filter that only accepts packets on port 53, are UDP and only coming from the target machine. If a packet gets through that filter, then there will be a check to see if it is a DNS Query Request. I did this by checking the 3rd and 4th octet of the DNS header, which contains this info. If these two octets concatenated together equal “01 then we can continue on and generate our spoofed response packet.

To create the spoofed response I again use the PacketFu library which allows us to create raw sockets and send packets back to the client machine. I created a new UDP packet where we have to fill the headers with information as well as the DNS header. The source IP stays as the routers IP so nothing suspicious will be seen. The DNS header is in the UDP packets payload so we can add the DNS response information there. The transaction ID of the query must be the same as the response, so we can re-use this. I add the transaction ID to the payload first, because this is what is at the start of a DNS packet always. I had to open Wireshark and analyze a DNS response packet to see how it is crafted. By doing so, I was able to figure out how the DNS packet is padded.

We then add the DNS name of the website the target machine requested. After, we pad more hex values into the payload until we need to insert the DNS time to live value and our packet length. At the very end of the payload I give the spoofed IP address which I want the target machine to redirect to, which in this case is Facebook.com’s IP address. To finish up the creation I had to recalculate all the fields of the UDP packet and send it to my machines ethernet card interface. This will then send the spoofed response back to the target machine, still thinking it came from the router. The full payload of the dns packet can be seen below. I will go over this step by step with screenshots in the testing section of this document.

UDP (DNS) Packet Payload



```
udp_pkt = UDPPacket.new(:config => @myInfo)
udp_pkt.udp_src = pkt.udp_dst
udp_pkt.udp_dst = pkt.udp_src
udp_pkt.eth_daddr   = @victimMac
udp_pkt.ip_daddr    = @addr
udp_pkt.ip_saddr    = pkt.ip_daddr

#Transaction ID (must be same for request and response)
udp_pkt.payload     =  pkt.payload[0,2]

#DNS header before Domain Name
udp_pkt.payload     += "\x81"+"\x80"+"\x00"+"\x01"+"\x00"+"\x01"
udp_pkt.payload     += "\x00"+"\x00"+"\x00"+"\x00"

#split the domaine name by the "." ex. www.google.com
@domainName.split('.').each do |domainString|
    #put length before each part of the domain
    udp_pkt.payload += domainString.length.chr
    #section of domain
    udp_pkt.payload += domainString
end

#DNS header after domain name
udp_pkt.payload     += "\x00"+"\x00"+"\x01"+"\x00"+"\x01"+"\xc0"
udp_pkt.payload     += "\x0c"+"\x00"+"\x01"+"\x00"+"\x01"
#DNS TTL and Length
udp_pkt.payload     += "\x00"+"\x00"+"\x02"+"\x56"+"\x00"+"\x04"
#our ip to send to
udp_pkt.payload     += spoofIPHex
#recalculation of fields
udp_pkt.recalc
#send to interface
udp_pkt.to_w(@interface);
```

Testing

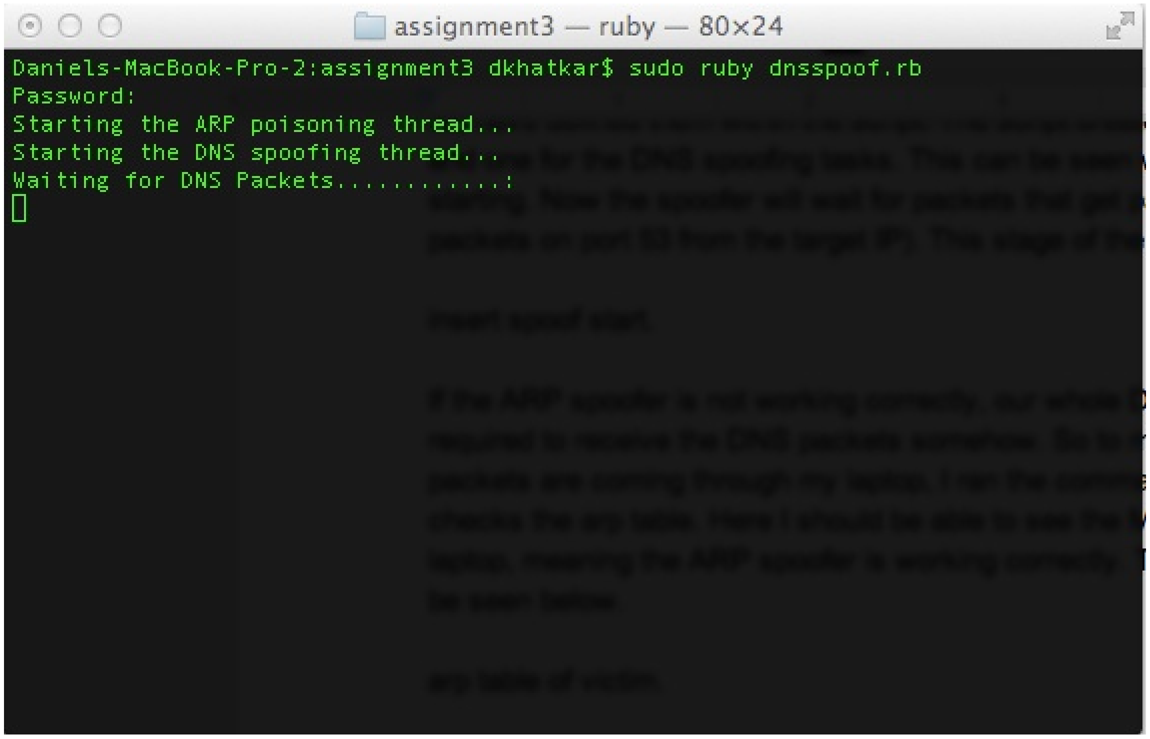
For this test, I used my laptop with another machine on the same network. My laptop was the “middle” hop, while the other machine was the target machine (victim). Before I began my test, I need to change a few parameters in our ruby script. In this script the mac addresses of the target and router are generated automatically using the Utils.Arp() method in the PacketFu library. All I had to do was supply the IP addresses of each. I must define the IP addresses and interface of my laptop before the script can run correctly. The MAC addresses are then defined for the router and target packets used in the ARP spoofing. This can be seen in the screenshots below.

IP address, MAC address and Interface Definitions

```
@interface = "en1"
@myInfo = Utils.whoami?(:iface => @interface)
@addr = "192.168.0.188"
@routerIP = "192.168.0.1"
@spoofIP = "69.171.234.21"
@routerMac = Utils.arp(@routerIP, :iface => @interface)
@victimMac = Utils.arp(@addr, :iface => @interface)
```

Next, I can start the DNS spoofer script. As you can see here it takes no arguments, because we have defined them within the script. The script creates two threads, one for the ARP spoofer and one for the DNS spoofing tasks. This can be seen when the spoofer states both are starting. Now the spoofer will wait for packets that get past the filter, as I described earlier (udp packets on port 53 from the target IP). This stage of the spoofer can be seen in the image below.

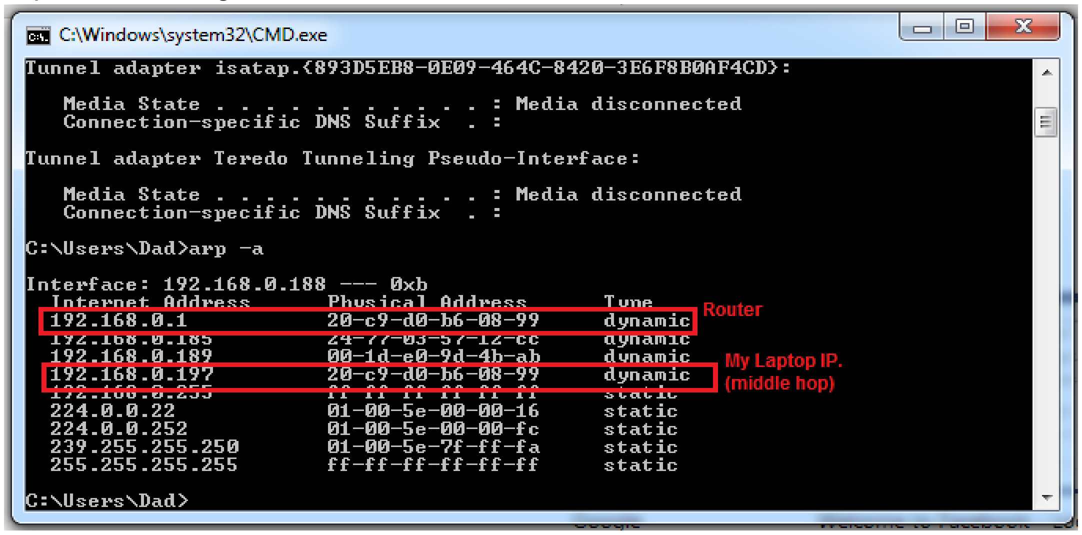
Start DNS Spoofer



```
Daniels-MacBook-Pro-2:assignment3 dkhatkar$ sudo ruby dnsspoof.rb
Password:
Starting the ARP poisoning thread...
Starting the DNS spoofing thread...
Waiting for DNS Packets............:
```

If the ARP spoofer is not working correctly, our whole DNS spoofer will not work, since we are required to receive the DNS packets somehow. So to make sure that the target machines packets are coming through my laptop, I ran the command “arp -a” on the target machine. This checks the arp table. Here I should be able to see the MAC address and IP address of the my laptop, meaning the ARP spoofer is working correctly. The arp table of the target machine can be seen below.

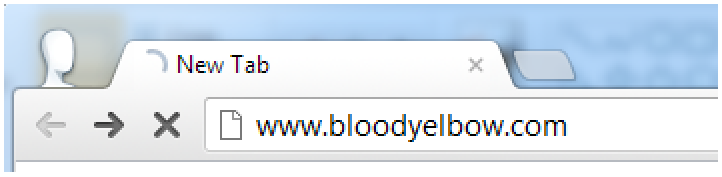
Arp Table of Target Machine



```
Tunnel adapter isatap.{893D5EB8-0E09-464C-8420-3E6F8B0AF4CD}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\Dad>arp -a

Interface: 192.168.0.188 --- 0xb
  Internet Address      Physical Address      Type
  192.168.0.1           20-c9-d0-b6-08-99     dynamic
  192.168.0.185         24-77-03-57-12-cc     dynamic
  192.168.0.189         00-1d-e0-9d-4b-ab     dynamic
  192.168.0.197         20-c9-d0-b6-08-99     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\Dad>
```

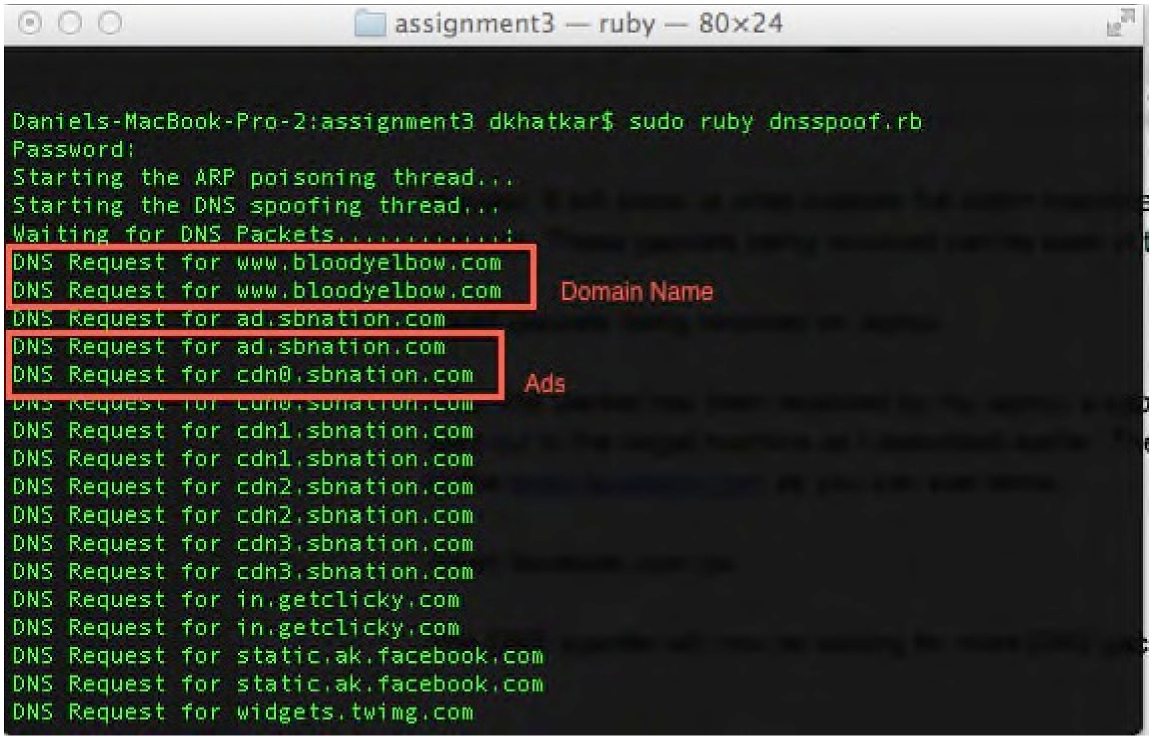
Next we can open a browser on the target machine and begin to type a website address (in this case www.bloodyelbow.com see image below). At this point everything is working as planned for the victim on the target machine. As soon as the victim hits enter, there will be packets being received on my laptop (the middle machine).

Entering the BloodyElbow website (before DNS Request sent)



To identify if a packet is being received on the laptop, I have added in a print statement that grabs the domain name from the DNS query request. It will show us what domain the victim machine is requesting the IP for from DNS servers. In the below screenshot you can see the packets being received. You can notice that there are also extra dns packets, these are ads on the website that are also sending requests to the DNS server.

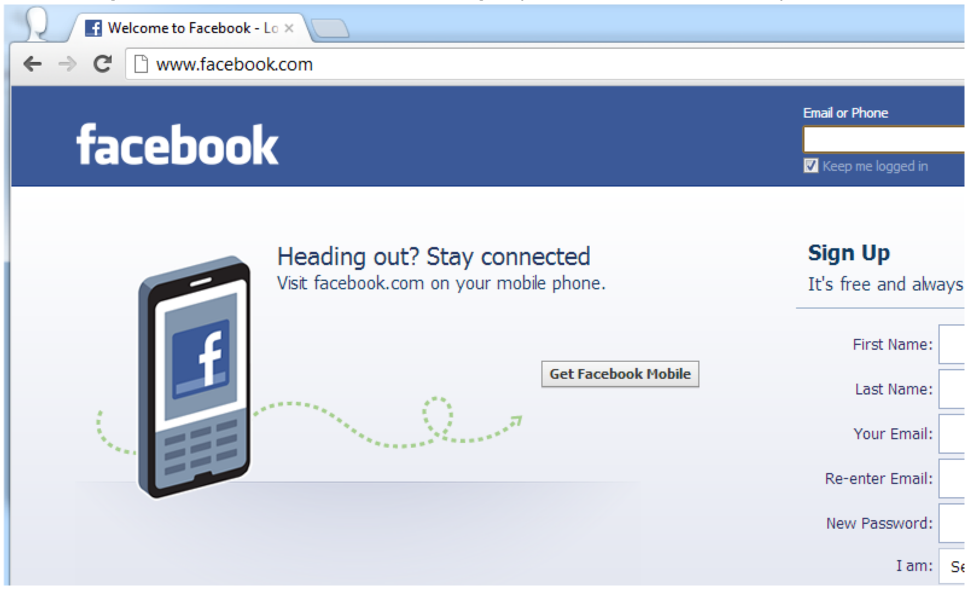
DNS Request packets being
received



```
Daniels-MacBook-Pro-2:assignment3 dkhatkar$ sudo ruby dnsspoof.rb
Password:
Starting the ARP poisoning thread...
Starting the DNS spoofing thread...
Waiting for DNS Packets............:
DNS Request for www.bloodyelbow.com
DNS Request for www.bloodyelbow.com
DNS Request for ad.sbnation.com
DNS Request for ad.sbnation.com
DNS Request for cdn0.sbnation.com
DNS Request for cdn0.sbnation.com
DNS Request for cdn1.sbnation.com
DNS Request for cdn1.sbnation.com
DNS Request for cdn2.sbnation.com
DNS Request for cdn2.sbnation.com
DNS Request for cdn3.sbnation.com
DNS Request for cdn3.sbnation.com
DNS Request for in.getclicky.com
DNS Request for in.getclicky.com
DNS Request for static.ak.facebook.com
DNS Request for static.ak.facebook.com
DNS Request for widgets.twimg.com
```

After the request packet has been received by my laptop, a spoofed response packet is generated and sent out to the target machine as I described earlier. The IP to facebook.com is put in the payload causing the victims browser to be redirected to the www.facebook.com as you can see below.

The Target Machines Redirected Page (www.facebook.com)



The DNS spoofer will now be waiting for more DNS packets and can be stopped whenever.

Conclusion

The DNS spoofer I have created takes advantages of a vulnerability within networks that can be used for many dangerous types of attacks on an individual. This DNS spoofer I created is working on one IP address right now, but there is definitely an opportunity to be able to run it on a whole network. Being able to do that could be very dangerous. You would be able to directly victims to perhaps a web server you have set up and gain information about the victim. There are multiple different avenues an attacker could take while using DNS spoofer.

This tool is included in many programs like Metasploit and Ettercap, showing just how easy it is to run and that someone with barely any experience could run it and perform this type of attack. By using a script that has already been created for you limits the attacker. This is what separates the more knowledgeable network analysts and the script kiddies. That is why I found this assignment useful. Not only did I create a successful DNS spoofer that redirects to facebook.com, but I can also modify it at anytime. By modifying it I could possibly make my filter tighter, only respond to certain domain names, or create more conditions based on the DNS request packet. These are only a few examples of things you can do.

Having the ability and knowledge to perform this DNS spoofing attack is very powerful within a network. Anyone can take advantage of the main problem with people within networks today, themselves. There are many ways to protect DNS spoofing, which may include setting up security features that will be check all ARPs and compare authenticity on the router and switch. The reality is that you may only see that within big businesses and not at a place such as Starbucks, which is popular for such attacks. People need to become more educated about their Internet use, but until then DNS spoofing with ARP spoofing will continue to take advantage of less informed users.