**Report CSE 508 Network security – DNS packet injector and Detector**

**SBU-ID -111446700, Saahil Khurana**

The program has 2 sections dnsinject.py and dnsdetect.py. I have used 2 VM’s to test the IP spoofing. On both the VM’s I have used Ubuntu 16.04.3 LTS ISO images on VMware workstation. The name of the first VM is attacker Ubuntu and the second is victim Ubuntu.

The attacker Ubuntu runs dnsinject.py file.

Steps to run the python file on both the VM’s:

1. Install pip for python – sudo apt-get install python-pip
2. Now install the dependencies of the program like scapy, netifaces. Using commands like

pip install scapy

pip install netifaces

Before this the compiler may ask you to upgrade the version of the pip. Do so using –

pip install –upgrade pip

We need to run these commands on both the VM’s to ensure that the python files runs.

After the setup the dnsinject file will be run using-

dnsinject [-i interface] [-f hostnames] expression

-i specifies the interface

-f specifies the host file

<expression> = BPF filter. This is optional and can be used to sniff and spoof only a particular kind of traffic like udp port 53.

1. sudo python dnsinject

This is the default command and will spoof for all the packets and on the default interface. It will select the first interface which has a legitimate IP as the default interface. A case has been added in code to prevent it from selecting the ‘lo’ as the default interface. The chooseInterface() method chooses the default interface.

sudo python dnsinject –i eps33

Like before, this command will spoof all packets on the given interface. And will use the attackets IP in the spoofed response.

Sudo python dnsinject –h host –i eps33

In this case the command will spoof only the packets for the requests made for urls in the host file. Suppose the host file does not have amazon.com and victim requests for amazon.com at its end. Then the program will not spoof the packet for such a request because it’s not a part of the host file. –h option can be given with or without the –i option. The spoofed packet will have the ip as mentioned in the host file in the response. For example –

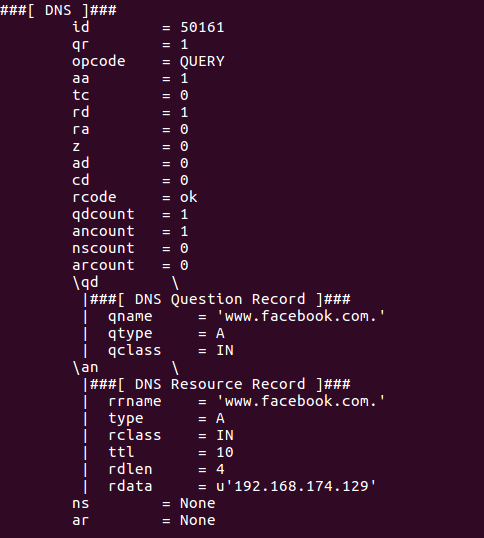
10.6.6.6 foo.example.com

10.6.6.6 bar.example.com

192.168.66.6 [www.cs.stonybrook.edu](http://www.cs.stonybrook.edu)

A typical host file is a list of pairs of IP and url addresses.

Snapshot of dnsinject –



**spoofed packet injected with local IP**

The dnsdetect program runs on the other VM named Victim Ubuntu. Here the program will detect if multiple answers are coming for a DNS request. To run the program install pip, scapy and netifaces like we did on Attacker Ubuntu VM.

Format:

dnsdetect [-i interface] [-r tracefile] expression

-i specifies the interface

-r specifies the trace file, it should be a pcap file.

<expression> = BPF filter. This is optional and can be used to sniff and spoof only a particular kind of traffic like udp port 53.

The dnstect command can be invoked by typing –

1. sudo python dnsdetect
2. sudo python dnsdetect –i eps33
3. sudo python dnsdetect –r packets.pcap

The first command will listen on the default interface for all duplicate or forged responses. The default interface will be selected using similar code we used in dnsinject for selecting the default interface. The second command will listen on the interface mentioned for all the spoofed responses. The first 2 commands can be tested in real time by opening a terminal on Victim Ubuntu and using dig command like dig @8.8.8.8 paypal.com. This dig command is a dns request to the google dns server which will cause 2 responses – one from dns and one from dnsinject code. The conflicting packets will be printed in the dnsdetect terminal window. The format of the output is –

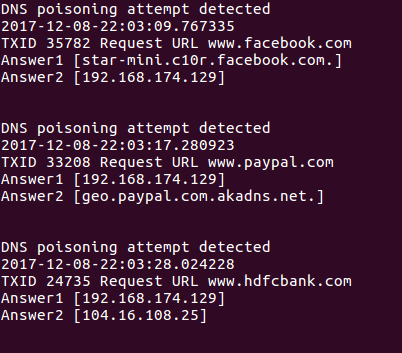
Time stamp

Transaction id

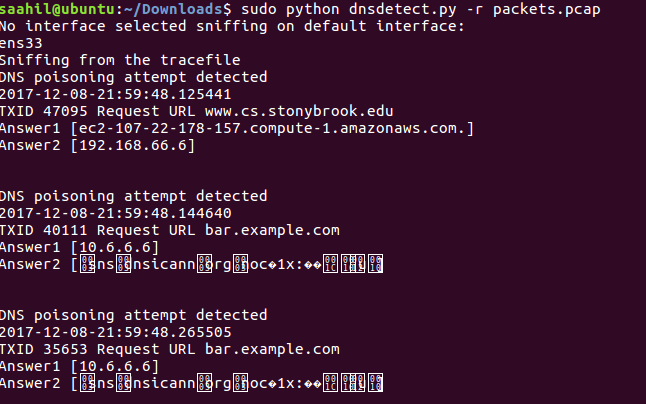
Response 1 with IP

Response 2 with IP

The third command will read a pcap file and check for multiple dns response packets having the same id but differing rdata. It will print such packets to the terminal where we are running the dnstect.py.



**Snapshot of dnsdetect.py on live packets –**



**Snapshot of dnsdetect.py from packets.pcap file-**

**General design**

The Programs have been written in Python using Scapy library APIs

Part1:

All the packets are sniffed on the selected interface (or default interface). A call back function is called using sniff() which sniffs for packets having a DNS Question record. A spoofed ip packet is constructed based on whether the sniffed packet is TCP or UDP. After the packet is constructed it is sent to the victim as a legitimate response from the DNS.

Part2:

All the packets are sniffed at the victim in promiscous mode for online mode. For offline mode the pcap file is scanned. The callback function is called. A deque data-structure has been used which holds last 20 received packets.

A comparison of all fields is made in the callback function to detect if the packet is a forged response. Once found appropriate message with required details are printed on the stdout.

Whenever a packet is received it is compared with all the packets in the queue. When a packet is received with same destination IP, source port, Destination port, transaction ID, Request URL, But different response IP and payload as compared to a packet already present in the queue, It is declared to be forged and the appropriate fields are printed to the stdout.

How to avoid False positives-

There might me a case when 2 similar but legitimate packets come from the DNS server but the order of the Ip’s sent might me different, in such a case when we compare 2 packets their IP, source, destination and their ports will match. So we call a method def matchRdata() which dissects the two packets and puts all their IP in the answer section to a list. If we compare the sorted lists and they are equal it means that both the packets are legitimate and we will return False. That’s how my code avoids false positives.

 Included files:

------------------

dnsinject.py

dnsdetect.py

report.txt

host

packets.pcap (captured on the detector side using tcpdump command, can be used with dnsdetector code with –r option)

5. If you have used any standard code available over the internet, even if for a part of your hw, make sure you provide the citation

One important point to be noted is that - The sniff (offline=tracefile, filter=expression) seems to be working on the current scapy version, it is has not been pushed in the stable pip version. We can clone the latest repo and try it out - <https://github.com/secdev/scapy>. (Piazza discussions)

References -

<http://www.cs.dartmouth.edu/~sergey/netreads/local/reliable-dns-spoofing-with-python-scapy-nfqueue.html#selection-1213.0-1213.1>

<http://bt3gl.github.io/black-hat-python-infinite-possibilities-with-the-scapy-module.html>

https://stackoverflow.com/questions/12501780/dnsrr-iteration