FIT5037 Network Security Assignment
Total Marks 100
Due on Sunday, 4 August 2024, 11:55 PM.

1 Overview

The learning objective of this assignment is for you to gain a first-hand experience on network attacks (i.e., TCP and DNS attacks) and get a deeper understanding on how to launch these attacks in practice. All tasks in this assignment can be done on the virtual machine used in the labs.

2 Submission Policy

You need to submit a lab report (one single PDF file) to describe what you have done and what you have observed with screen shots whenever necessary; you also need to provide explanation or codes to the observations that are related to the tasks. In your report, you are expected to answer all the questions listed in this manual. Typeset your report into .pdf format (make sure it can be opened with Adobe Reader) and name it as the format: **[Your Name]-[Student ID]-FIT5037-Assignment**, e.g., HarryPotter-12345678-FIT5037-Assignment.pdf.

All source code if required should be embedded in your report. In addition, if a demonstration video is required, you should record your screen demonstration with your voice explanation and upload the video to your Monash Google Drive. **For video demonstration, please keep it to maximum of 30 minutes in total duration; you are required to say your name and student ID at the start of recording, showing face is mandatory.** The shared URL of the video should be mentioned in your report wherever required. You can use any tool you would like to record videos, for example panopto (`https://monash-panopto.aarnet.edu.au/`) and Zoom.

Late submission penalty: 10-point deduction per day. If you require a special consideration, the application should be submitted and notified at least three days in advance. Zero tolerance on plagiarism: If you are found cheating, penalties will be applied, i.e., a zero grade for the unit. The demonstration video is also used to detect/avoid plagiarism. University polices can be found at `https://www.monash.edu/students/academic/policies/academic-integrity`.

3 Environment Setup

In this section, you need to double check whether you have configured GNS3 correctly. We will be using the Topic 6 - Week 5A lab configuration, i.e., your GNS3 configuration should look like below:

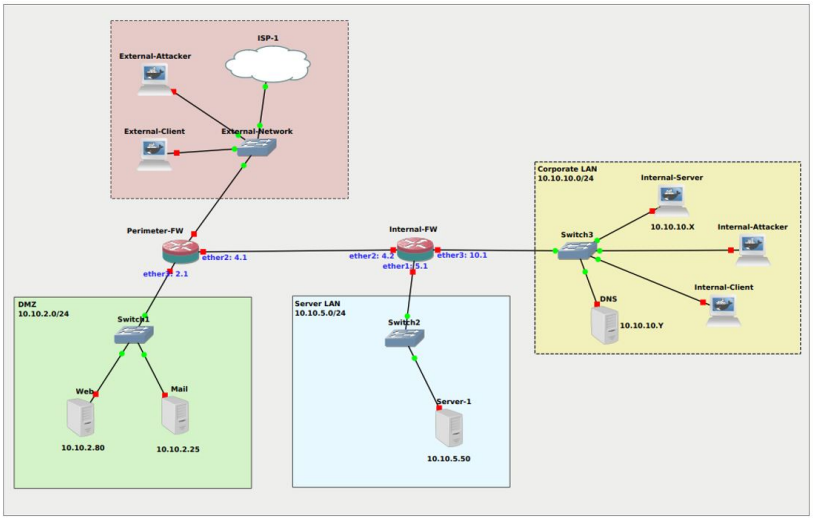

Figure 1: GNS3 Config

Otherwise, if you don't have the VM ready, we refer you to Environment Setup in Week 01. It is recommended to perform lab tasks of Topic 6 - Week 5A before proceeding.

4 TCP Attacks – Using Scapy [40 Marks]

The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It sits on top of the IP layer, and provides a reliable and ordered communication channel between applications running on networked computers. TCP is in a layer called Transport layer, which provides host-to-host communication services for applications. To achieve such reliable and order communication, TCP requires both ends of a communication to maintain a connection. Unfortunately, when TCP was developed, no security mechanism was built into this protocol, making it possible for attackers to eavesdrop on connections, break connections or hijack connections. In this section, you are required to perform these attacks using Scapy—a packet manipulation tool for computer networks written in Python.

4.1 Task 1: TCP Reset Attacks [15 Marks]

In the stream of packets of a TCP connection, each packet contains a TCP header. In the header, there is a bit known as the "reset" (RST) flag. In most packets, this bit is set to 0 and has no effect; however, if this bit is set to 1, it indicates that the receiver should immediately stop using the TCP connection. That means it should not send back any more packets using the connection's identifying numbers, called ports, and discard any further packets with headers belong to that connection. A TCP reset basically kills a TCP connection instantly.

It is possible for a third computer (aka attacker) to monitor the TCP packets on the connection and then send a "forged" packet containing a TCP reset to one or both endpoints. The headers in the forged packet must indicate, falsely, that it came from an endpoint, not the forger. This information includes the endpoint IP addresses and port numbers. Every field in the IP and TCP headers must be set to a convincing forged value for the fake reset to trick the endpoint into closing the TCP connection.

The idea is quite simple: to break up a TCP connection between A and B, the attacker just spoofs a TCP RST packet from A to B or from B to A.

Q1: Connect from `Internal-Client` to `Internal-Server` using SSH (use `apt install ssh` if SSH is not installed), the username and password are same: `msfadmin`. Perform TCP RST attack, from `Internal-Attacker` workstation, on SSH service using `Scapy` (python-based) packet generator. `Internal-Client` terminal should show the connection is terminated. Please submit your python code and the steps, along with video link showing that you have performed the attack. **(Python code: 5 marks, explanation during recording demonstration: 5 marks)**

Q2: Briefly explain the TCP RST attack and propose at least two theoretical countermeasures. You do not have to do any configuration/implementation for this task. **(Explanation: 2.5 marks, countermeasures: 2.5 marks)**

4.2 Task 2: TCP Session Hijacking Attacks [25 Marks]

Once a TCP client and server finish the three-way handshake protocol, a connection is established, and we call it a TCP session. From then on, both ends can send data to each other. Since a computer can have multiple concurrent TCP sessions with other computers, when it receives a packet, it needs to know which TCP session the packet belongs to. TCP uses four elements to make that decision, i.e., to uniquely identify a session: (1) source IP address, (2) destination IP address, (3) source port number, and (4) destination port number.

We call these four fields as the signature of a TCP session. As we have already learned, spoofing packets is not difficult. What if we spoof a TCP packet, whose signature matches that of an existing TCP session on the target machine? Will this packet be accepted by the target? Clearly, if the above four elements match with the signature of the session, the receiver cannot tell whether the packet comes from the real sender or an attacker, so it considers the packet as belonging to the session.

However, for the packet to be accepted, one more critical condition needs to be satisfied. It is the TCP sequence number. TCP is a connection-oriented protocol and treats data as a stream, so each octet in the TCP session has a unique sequence number, identifying its position in the stream. The TCP header

contains a 32-bit sequence number field, which contains the sequence number of the first octet in the payload. When the receiver gets a TCP packet, it places the TCP data (payload) in a buffer; where exactly the payload is placed inside the buffer depends on the sequence number. This way, even if TCP packets arrive out of order, TCP can always place their data in the buffer using the correct order.

The objective of this task is to hijack an existing TCP connection (session) between client and server by injecting malicious contents into their session.

Q3: Connect TELNET from `Internal-Client` to `Internal-Server`, the username and password are same: `msfadmin`. Write a python code, using Scapy, which can inject packets in the TELNET communication, the goal is to make a directory called "attacker" at the `Internal-Server` (as seen in the screenshot below). You can use `Internal-Attacker` workstation to run the python code. Submit python code and steps, along with video link that demonstrates you have performed the attack. **(Python code: 5 marks, explanation during recording demonstration: 5 marks)**

```
msfadmin@Internal-Server:~$ ls
attacker  vulnerable
```

Figure 2: Directories in `Internal-Server`

Q4: Connect TELNET from `Internal-Client` to `Internal-Server`. The objective is to get a reverse shell from `Internal-Server`. Reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine. We are omitting the details of reverse shell and encourage students to research about it, you can start from here: `https://hackernoon.com/reverse-shell-cf154dfee6bd`. Write a python code, using Scapy, which can inject packets in TELNET communication and create a reverse shell from `Internal-Server` to `Internal-Attacker` (as seen in the screenshot below, in this case the `Internal-Server`'s IP address is 10.10.10.197). Submit python code and steps, along with video link showing that you have performed the attack. **(Python code: 5 marks, explanation during recording demonstration: 5 marks)**

```
root@Internal-Attacker:~# nc -lvp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.10.197 49416
```

Figure 3: Receiving reverse shell

Q5: Connect SSH from `Internal-Client` to `Internal-Server`, the username and password are same: `msfadmin`. Perform same TCP hijacking attacks as you did for TELNET, i.e. make attacker directory in `Internal-Server` and create a reverse shell from `Internal-Server` to `Internal-Attacker` by hijacking SSH connection. If your attacks are successful, please submit python code and steps, along with video link showing that you have performed the attacks. If your attacks were unsuccessful, explain the reason in detail. **(Python Code and Explanation during recording demonstration: 5 marks)**

5 DNS Attacks – Using Scapy [60 Marks]

Domain Name System (DNS) is an essential component of the Internet infrastructure. It serves as the phone book for the Internet, so computers can look up for "telephone number" (i.e. IP addresses) from domain names. Without knowing the IP address, computers will not be able to communicate with one another. Due to its importance, the DNS infrastructure faces frequent attacks. In this section, you will explore the most primary attack on DNS. That is DNS cache poisoning by investigating both Local and Remote DNS cache poisoning attacks.

Due to the large number of computers and networks on the Internet, the domain namespace is organised in a hierarchical tree-like structure. Each node on the tree is called a domain or sub-domain when referencing to its parent node. The following figure depicts a part of the domain hierarchy.

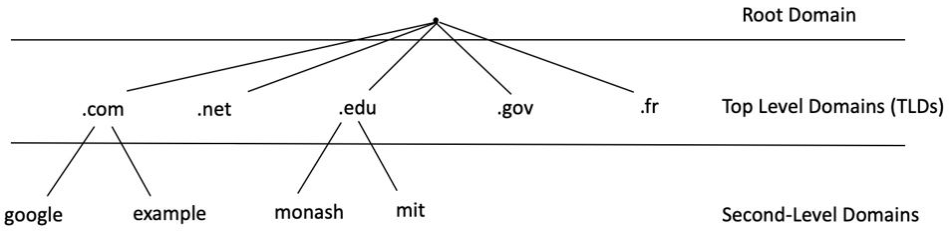

Figure 4: Domain hierarchy

The domain hierarchy tree structure describes how the domain namespace is organised, but that is not exactly how the domain name systems are organised. Domain name systems are organised according to zones. A DNS zone basically groups contiguous domains and sub-domains on the domain tree, and assign the management authority to an entity. Each zone is managed by an authority, while a domain does not indicate any authority information. The following figure depicts an example of the `example.com` domain.

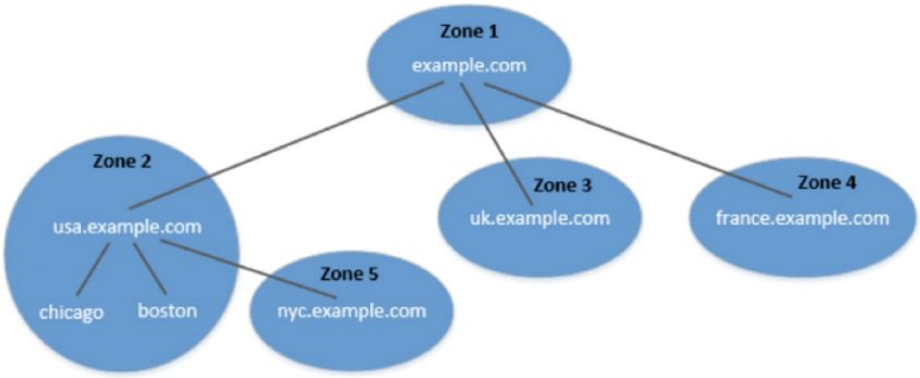

Figure 5: DNS Zones

Assume that `example.com` in the above figure is an international company, with branches all over the world, so the company's domain is further divided into multiple sub-domains, including `usa.example.com`, `uk.example.com`, and `france.example.com`. Inside US, the usa sub-domain is further divided into `chicago`, `boston`, and `nyc` subdomains.

Each DNS zone has at least one authoritative nameserver that publishes information about that zone. The goal of a DNS query is to eventually ask the authoritative DNS server for answers. That is why they are called authoritative because they provide the original and definitive answers to DNS queries, as opposed to obtaining the answers from other DNS servers.

With such arrangement, the root zone for example.com only needs to keep records of who the authority is for each of its subdomains. By doing this, it maintains the independence among the branches in different countries and enable the administrative right of those subdomains, so the branch in each country manages its own DNS information. For a given DNS query, if your local DNS server does not the answer, it will ask other DNS servers on the Internet for answer via hierarchical authority servers. The following example demonstrates a dig (DNS query) for the domain www.example.net when sending the query directly to one of the root server (i.e. `a.root-servers.net`).

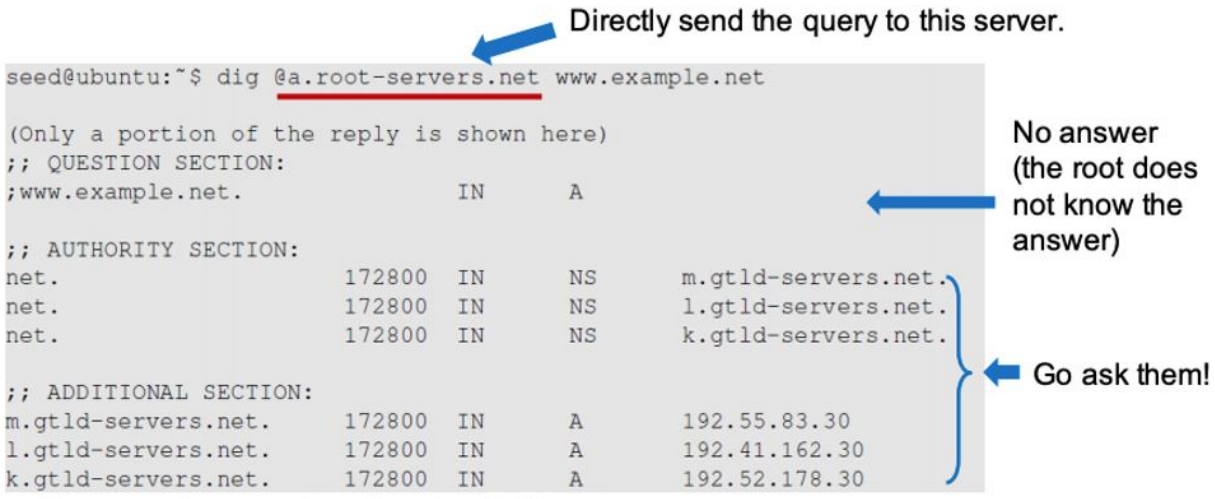

Figure 6: DIG to the root server

There are four types of sections in a DNS response: *question section*, *answer section*, *authority section*, and *additional section*. From the above result, we can see that the root server does not know the answer (because the reply does not include an answer section, but it tells several authoritative nameservers for the net zone (the NS records in the authority section), along with their IP address if possible in the *additional section*). If you continuously dig the domain `www.example.net` on one these authoritative nameservers, you will finally end up with the answer section showing the IP address of the machine hosting the website for `www.example.net`.

When your local DNS server gets information from other DNS servers, it caches the information, so if the same information is needed, it will not waste time to ask again.

5.1 Task 3: Local DNS Attack targeting Authority Nameserver [20 Marks]

We recalled that a DNS response contains question section, *answer section*, *authority section*, and *additional section*. If we only target the *answer section*, the attack only affects one hostname (as we did in our Topic 6 - Week 5A lab "DNS Spoofing Attacks"). Real DNS attacks usually target the authority section by providing a fake NS record for the target domain in the authority section. If the fake NS record is cached, when the victim local DNS server tries to find any IP address in the target domain, it will send a request to the malicious nameserver specified in the fake NS record. Such an attack can affect all the hostnames in the target domain. In this task, you will explore how to target the authority server of `example.net` and manage to replace it with `ns1.attacker.com` and `ns2.attacker.com`.

Q6: Submit your python code and write comments in the code step by step to perform the DNS spoofing attack that modifies the authority server of example.net to be `ns1.attacker.com` and `ns2.attacker.com`. Use `Internal-Client` as victim and `Internal-Attacker` as the attacker machine. **(Python code: 10 marks)**. If the attack works, you should see the result as in following figures for which the malicious authoritative servers are taken place.

```
root@Internal-Client:/# dig example.net

; <<>> DiG 9.16.1-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60077
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;example.net.                   IN      A

;; ANSWER SECTION:
example.net.            303030  IN      A       10.10.10.1

;; AUTHORITY SECTION:
example.net.            90000   IN      NS      ns1.attacker.com.
example.net.            90000   IN      NS      ns2.attacker.com.

;; ADDITIONAL SECTION:
ns1.attacker.com.       90000   IN      A       10.10.10.1
ns2.attacker.com.       90000   IN      A       10.10.10.2

;; Query time: 32 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sat Aug 21 22:20:55 AEST 2021
;; MSG SIZE  rcvd: 202
```

Figure 7: Q6

Q7: Provide your explanation in video demonstration to support your above DNS spoofing attack. **(Explanation and attack's results: 5 marks, highlight Wireshark packet monitor in the video: 5 marks)**

5.2 Task 4: Remote DNS Attack targeting Authority Server [40 Marks]

For this task, the attacker (`Internal-Attacker`) and `DNS` server need to be in different LAN. We will move `DNS` server to Server LAN and configure its IP statically. The GNS3 configuration for this task should look like below:

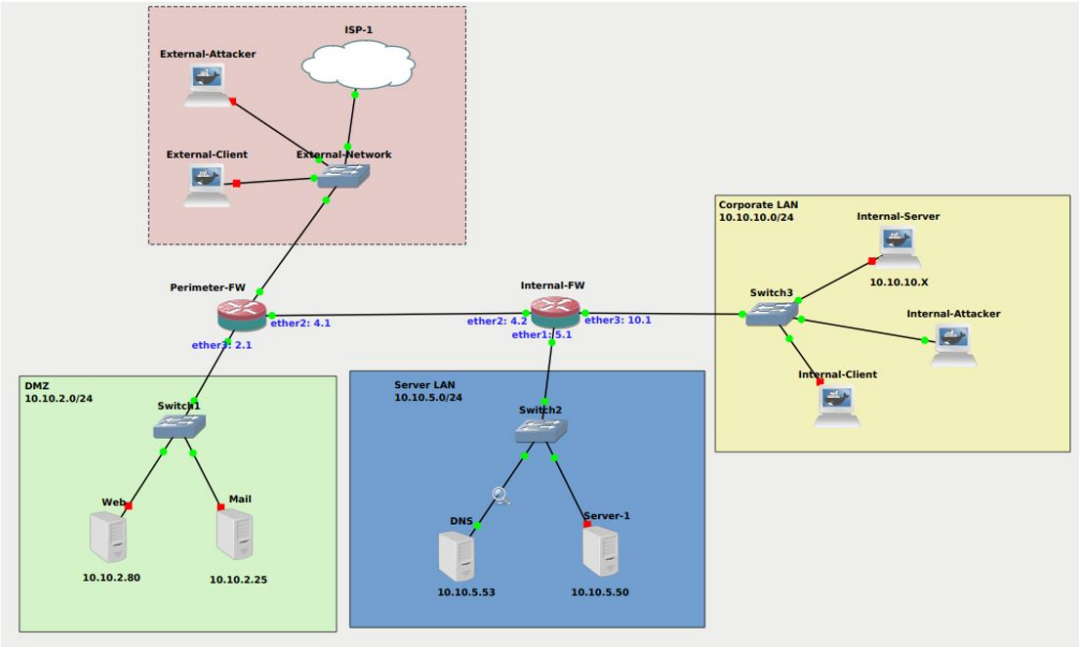

Figure 8: GNS3 for Remote DNS Attacks

Configure static IP for DNS:

```
#
# This is a sample network config uncomment lines to configure the network
#


# Static config for eth0
auto eth0
iface eth0 inet static
        address 10.10.5.53
        netmask 255.255.255.0
        gateway 10.10.5.1
        up echo nameserver 8.8.8.8 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

Figure 9: Static IP config for `DNS`

Now login to `Internal-FW` (username is 'admin', no password), and execute the following to disable NAT:

```
ip firewall nat remove 0
```

The previous local DNS attacks assume that the attacker and the DNS victim server are on the same LAN so that she can observe the DNS query message and reply with a forged DNS packet. When the attacker and the DNS server are not on the same LAN, the attack becomes harder since the attacker cannot perform ARP poisoning attack and see the DNS query. When the DNS victim server cannot resolve the DNS query, it will forward the DNS query packet to the forwarder DNS server (Google DNS server in our current setup). The DNS query is sent via a UDP packet where the UDP's source port is a 16-bit random number. In addition, the 16-bit transaction ID in the DNS header is also self-created by the DNS victim server. Hence, if the remote attacker wants to forge the DNS response, the forged packet must contain the correct values of these two numbers; otherwise, the reply will not be accepted.

Without being able to sniff the query packet, the remote attacker can only guess these two numbers. The chance is one out of 2^{32} for each guess. If an attacker can send out 1000 spoofed responses, it may take several days to try up 2^{32} time. In contrast, it only takes few seconds to receive the correct packet response from the forwarder Google DNS. Consequently, that real reply will be cached by the local DNS victim server. To make another try, the attacker has to wait for the server to send out another DNS query when its cache times out. Hence, this attacking chance makes the remote DNS attack unrealistic.

The remote DNS attack had become an open problem until Dan Kaminsky came up with a simple solution in 2008. The attack is depicted in the following figure.

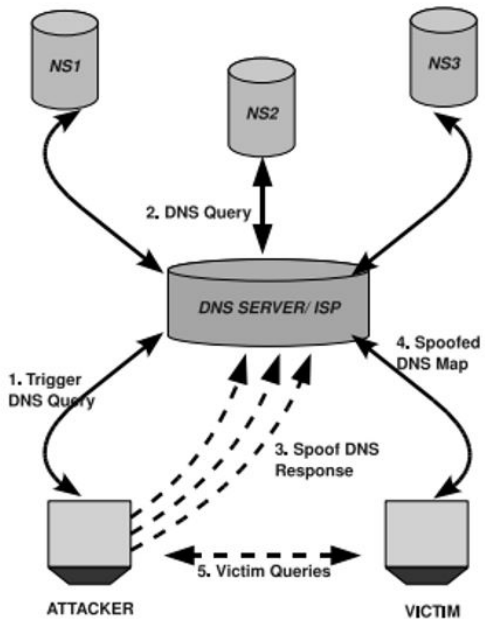

Figure 10: Kaminsky Attack

We choose a domain `test.com` as our targeted domain name in this task. When a client queries the `DNS` server for `www.test.com`, the attacker (`Internal-Attacker`) wants to cause the `DNS` server to use her DNS server (`ns.attacker.com`). The following steps with reference to above figure describe the outline of the attack.

1. The attacker queries the `DNS` server for a non-existing name in `test.com`, for example `xyz123.test.com`, where `xyz123` is a random name.
2. Since the mapping resolution cannot be resolved by the DNS server's cache, the server forwards the query to Google DNS (8.8.8.8) for that name resolution.
3. In the meantime, the attacker floods the `DNS` server with many spoofed DNS responses, each trying a different transaction ID and source port number (hoping one guess is correct). In that forged response, not only the attacker provides the IP resolution for the hostname `xyz123.test.com`, but also provides an authority name server for the domain `test.com`.

   Even if the response failed, the attacker would go back step one, and try another non-existing random name until the attacker succeeds.
4. Once the attack succeeds, when the client sends a DNS query to the poisoned DNS server for www.test.com, the nameserver returned by the `DNS` server will actually be set by the attacker.

To simplify and shorten the attack's simulation time in this task, we suggest you follow the below steps before doing the task.

1. Double check the IP addresses of the server, attacker, and client to ensure the server and the attacker are not in the same LAN (using ifconfig command)
   - `DNS`: 10.10.5.53
   - `Internal-Attacker`: 10.10.10.X
   - `Internal-Client`: 10.10.10.Y
2. In the DNS server's terminal, you can type the following command to configure DNS

   ```
   nano /etc/bind/named.conf.options
   ```

   Then, you should configure the forwarder 8.8.8.8, enable recursion and fix the query source port of the DNS server (i.e. 33333). With this constraint, the attacker now only needs to guess the transactionID of the DNS packet when performing remote DNS attacks. You can review the following figure for the correct configuration of DNS server.

```
  GNU nano 4.8                                  /etc/bind/named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.


        forwarders {
                8.8.8.8;
         };

        forward only;

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================

        query-source port 33333;
        dnssec-validation no;
        dump-file "/var/cache/bind/dumb.db";
        auth-nxdomain no;
        listen-on-v6 { any; };
};
```

Figure 11: DNS server config file

3. After making the changes in the above step, you should restart your DNS server by using the following command

```
/etc/init.d/named restart
```

4. Delete the DNS cache on the server using the below command. You need to run this command before starting the attack.

```
rndc flush
```

We provide you the remote_dns.py script template on Moodle that helps to perform the Kaminsky attack.

Q8: You need to complete Step 1 in the *remote_dns.py* to create 10000 dummy hostnames. **(The screenshot of your Python code for this step: 5 marks)**

Q9: You need to complete Step 2 in the *remote_dns.py* to generate a random DNS query for each dummy hostnames. **(The screenshot of your Python code for this step: 5 marks)**

Q10: You need to complete Step 3 in the *remote_dns.py* to flood about 100 random forged response packets. Each packet has:

- A randomly generated transaction ID for DNSpkt. **(5 marks for code and screenshot)**
- The malicious DNS server `ns.attacker.com` is included in the nameserver authority for the domain test.com when you construct DNSpkt. **(5 marks for code and screenshot)**
- *Additional section* showing `ns.attacker.com` has the IP of the attacker 10.10.10.X. **(5 marks for code and screenshot)**

Q11: Provide your video demonstration evidence to support and verify that you have performed the attack. You need to upload your demo video to your Monash Google Drive and embed its shared link to your report so that the teaching team can view and verify your works. In the video, you need to demonstrate following key points:

- Wireshark traffic captured on `DNS` server on `eth1` shows the `transactionID` in DNS packet sent by `DNS` server to Google, and the correctly matched transaction ID in the forged packet sent by `Internal-Attacker` to the `DNS` server. **(5 marks for step by step explanation of the attack using Wireshark in the demonstration video)**
- Once the poisoning is completed, from `Internal-Client`'s terminal use `dig` command to send a DNS query for the specific subdomain for which the attack was successful (e.g. in below screenshot it's snuy6.test.com). Do you get the attacker's IP?
- Now from the same terminal send a DNS query for `test.com`. If the attack was successful, the response should show `ns.attacker.com` in the *authority section* for the domain `test.com`. Was your attack successful? Explain in detail why or why not. **(10 marks for your explanation during the demonstration video)**

Figure 12: Attacker's screen shows poisoning was successful

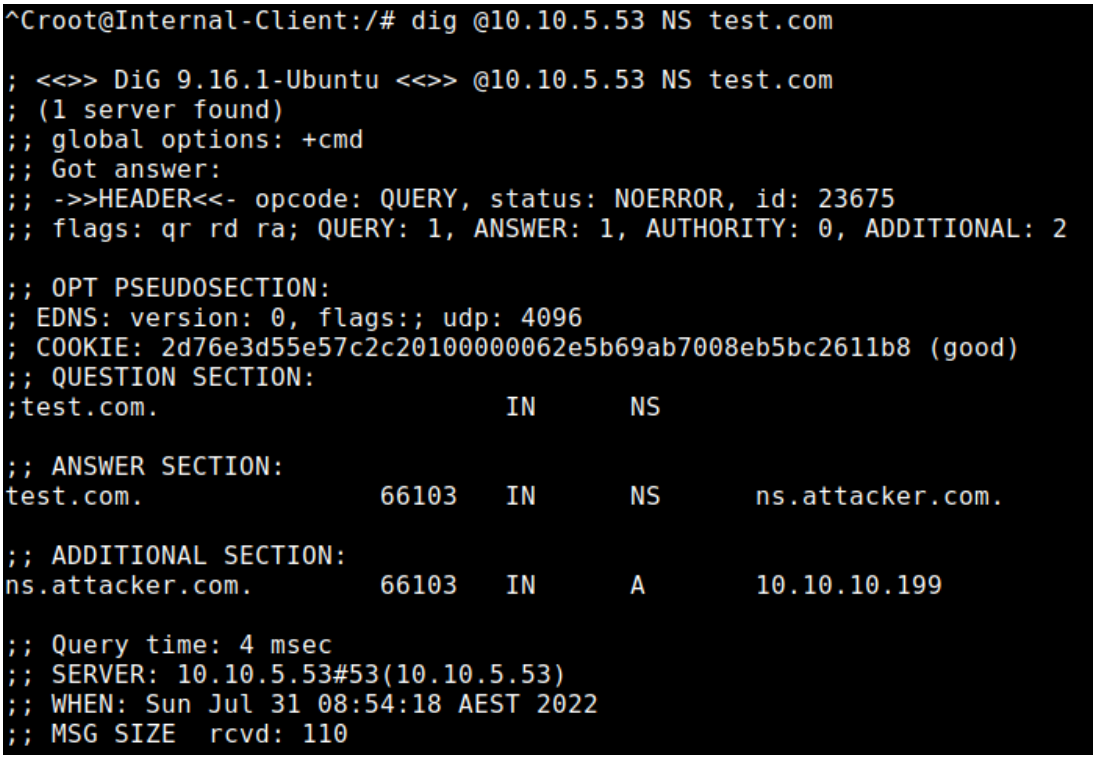

Figure 13: Internal-Client is answered from the poisoned cache.

6 Acknowledgement

Parts of this assignment and instructions are based on the SEED project (Developing Instructional Laboratory for Computer Security Education) `https://seedsecuritylabs.org`.