CS528 Lab 2 Report

Mohammad Haseeb
mhaseeb@purdue.edu

In this report I describe the process of creating the attacker for the Kaminsky attack on a remote DNS server. I then describe the verification process to verify that the attack was successful.

Following are the IP addresses and names of each of the three machines:

- DNS Server Machine (Apollo): 192.168.15.7
- User Machine (dns_usr): 192.168.15.8
- Attacker Machine (dns_attacker): 192.168.15.9

Following are the IP addresses of the two name-servers for example.edu

- Name-server 1: 199.43.135.53
- Name-server 2: 199.43.133.53

Note: Because of the nondeterministic nature of the attack, sometimes it may take longer for the DNS cache to get poisoned and the attack to be successful. On average, it takes 5-6 minutes for the attack to be successful.

Task 1: Remote Cache Poisoning

Attack Configuration

In order to make the attack possible, I had to make some changes to each of the three machines i.e. `Apollo`, `dns_usr` and `dns_attacker`. For the `dns_usr` and `dns_attacker` machines, I simply set the default DNS server to be `Apollo`. This was necessary so that both machines by default send their DNS requests to `Apollo`. I had to make some more configuration changes for the verification section, which will be explained later. For the DNS server `Apollo`, I changed the source port number such that all DNS query requests would be sent via this port. If this were not done, the attack would have become significantly harder due to the DNS server choosing random ports. Moreover, we disabled the `dnssec` functionalities by toggling the `dnssec-enable` field in the DNS server. Finally, I flushed the cache and restarted the `bind9` server.

Attacker Program

Much of the attack's code came from the `udp.c` file provided. The file provides starter code for continuously fabricating DNS request packets. To fabricate a correct DNS response packet, I first captured the DNS response from the correct `example.edu` name server (IP provided above) and observed what fields were required and how they changed.

Next, I declared a long buffer that would serve as the data to be passed to the raw socket. To fabricate the whole packet, I started from the IP header and moved up to the DNS header and eventually the DNS response payload. For each header, I casted the corresponding buffer pointer value to the one I required. For example, to make an IP header, I casted the memory address from the start of the buffer to the length of the IP header struct to be of type `struct ipheader`. I did the same for `udpheader` and `dnsheader`, changing the buffer offset for each header.

In order to construct the various sections of the DNS response, I used the sample packet and the corresponding byte `HEX` values provided in the lab handout. After filling the buffer with the DNS header values like flags, number of queries, the actual query etc., I entered sections and values for the authoritative name servers and the additional records. In order for the attack to be successful, all sections including the authoritative section are necessary. The DNS server ignores the values mentioned in the additional records section due to the DNS zones issue. I provide a brief discussion about this in the question at the end of this task.

The next part was to generate the transaction IDs for the spoofed DNS response packets. This is what ensures that the DNS server accepts our spoofed responses. An interesting thing to note here was that in all genuine DNS responses from the actual name servers, the transaction IDs were quite large. Consequently, after trial and error, I decided to use a higher transaction ID value to begin with.

After generating a new random query and sending out a DNS request for it, the attacker program increments the value of the transaction ID, assigns it to the DNS header part of the buffer, recalculates the checksums and then sends out the spoofed DNS response to `Apollo`.

Attack Limitations

The probability of the attack being successful depends on several factors that work in tandem. These factors are *(i)* port numbers *(ii)* transaction IDs *(iii)* attack window. I will briefly talk about each of them.

1. Port Numbers

As mentioned in the lab handout, we fix the source port number of the DNS to `33333`. I noticed that the response from the actual name server sent all responses to this port. This makes sense as the response should only go to the port number on which the DNS server's process is running. The default port numbers mentioned in the `udp.c` program were random. Therefore, I changed the source port number of my DNS responses to `53` and the destination port number to `33333`. This took care of the port numbers factor.

2. Transaction IDs

Kaminsky Attack relies on the fact that the transaction ID of our spoofed response matches the transaction ID of the initial request. The transaction ID field is 16 bits long and so, there are $2^{16} = 65536$ different transaction IDs we must try. I initially started generating transaction IDs from zero. However, I noticed that that the transaction ID value of the responses from the actual name server were quite large. I, therefore, set the initial transaction ID value to be much higher and this proved to be a good decision as the probability of the transaction IDs matching increased.

3. Attack Window

This is a subtle factor not mentioned in the lab handout that I discovered while studying the packet exchanges on Wireshark. I noticed that even if the transaction IDs matched, the attack was only successful if my correct spoofed DNS response reached `Apollo` *after* it had sent out the DNS query to the real name server. This also makes sense because the DNS server would simply ignore a response for which it had not sent out a query earlier. In order to account for this subtlety, I put my attacker program to sleep for a small period of time after it sent out a random DNS query and before sending out the spoofed responses. This was done so that I would send out the spoofed responses only within the valid attack window, which is often small. I could not find a solid way of determining the time period for which to put my attack program to sleep. It was based mainly on trial-and-error.

The above three factors played an important role in the attack being successful. Of the three, the latter two are nondeterministic and the attacker can only optimize these factors so much. The rest is all probabilistic.

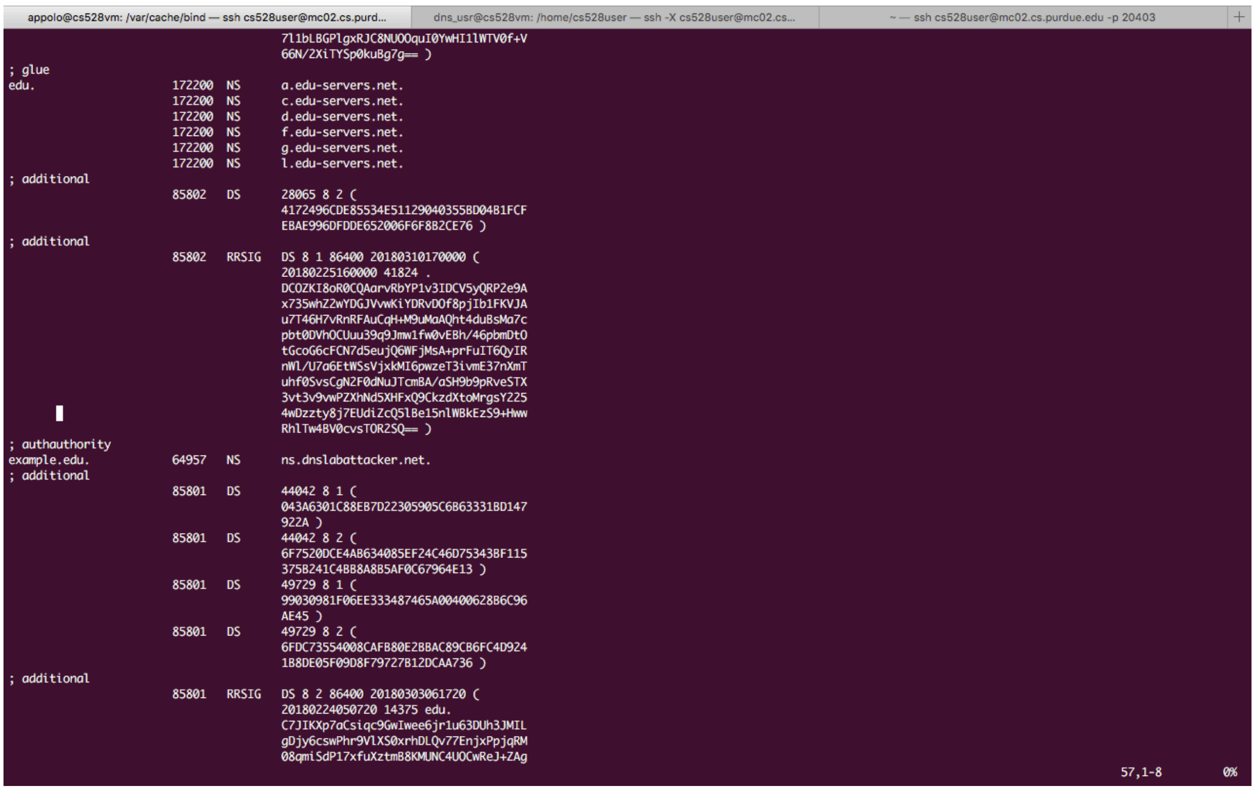
Question: Why is the IP address for `ns.dnslabattacker.net` mentioned in the additional records section of the spoofed DNS response not accepted by Apollo?

DNS is a distributed database organized in a hierarchical tree structure. Each layer of the tree represents a different domain or *zone*. As per DNS's specifications, only a special entity(s) may assume administrative role for different zones. In the case of this lab, we are dealing with two zones: *(i)* .edu zone and *(ii)* .net zone. When we send spoofed response packets to `Apollo`, we mention in our response that the authoritative name server for the `example.edu` domain is `ns.dnslabattacker.net`. `Apollo` accepts that response and updates its entry because we can spoof the IP of the actual name server of `example.edu` and that can be considered to have administrative rights to specify values of the `example.edu` domain. However, because we are not spoofing to the `ns.dnslabattacker.net` and we cannot, the information mentioned in the additional section that tells the IP of the `ns.dnslabattacker.net` is ignored. We are not given the authority to specify the IP for that zone/domain.

Task 2: Result Verification

In this section, I verified that the attack was successful as mentioned in the lab handout. The general idea of the verification process was to manually set the IP address of the `ns.dnslabattacker.net` server at `Apollo` to be that of the `dns_attacker` machine so that whenever a `dns_usr` machine queried for a `*.example.edu` URL, `Apollo` would look up its cache and find that the name server corresponding to `*.example.edu` is `ns.dnslabattacker.net` and return the corresponding IP as that of `dns_attacker`. Once the `usr_machine` has the IP of the (attacker) DNS server, it queries that and the `dns_attacker` responds with the spoofed IP of the URL. In this way, the attack is shown to be successful.

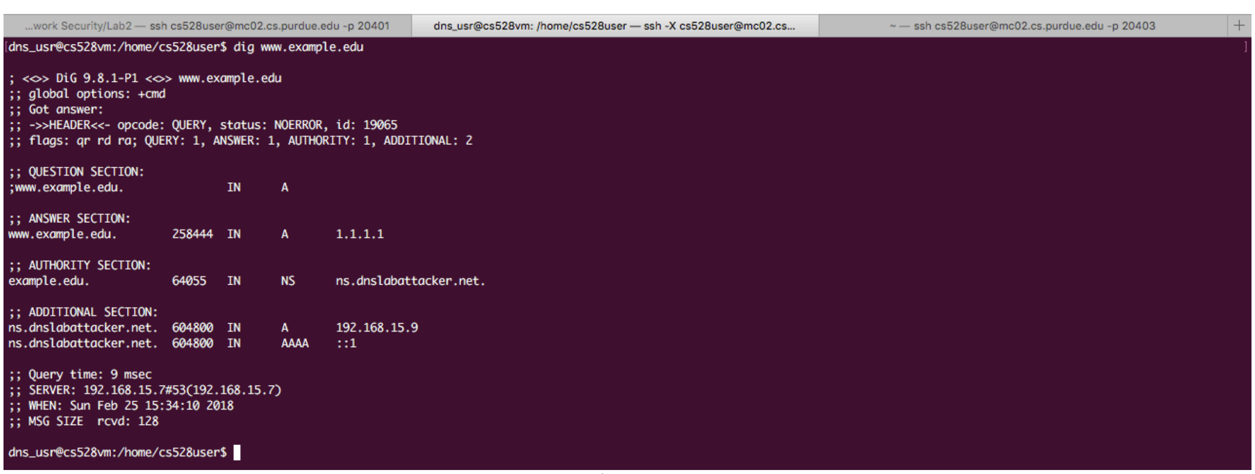
After executing the attack on `Apollo` and configuring the three machines as mentioned in the lab handout, I ran the dig command for various host names of the example.edu domain and verified that the results were all spoofed. The following screenshots demonstrate this.



```
appolo@cs528vm: /var/cache/bind — ssh cs528user@mc02.cs.purd...   dns_usr@cs528vm: /home/cs528user — ssh -X cs528user@mc02.cs...   ~ — ssh cs528user@mc02.cs.purdue.edu -p 20403
                                        7l1bLBGPlgxRJC8NUOOquI0YwHI1lWTV0f+V
                                        66N/2XiTYSp0kuBg7g== )
; glue
edu.                    172200  NS      a.edu-servers.net.
                        172200  NS      c.edu-servers.net.
                        172200  NS      d.edu-servers.net.
                        172200  NS      f.edu-servers.net.
                        172200  NS      g.edu-servers.net.
                        172200  NS      l.edu-servers.net.
; additional
                        85802   DS      28065 8 2 (
                                        4172496CDE85534E51129040355BD04B1FCF
                                        EBAE996DFDDE652006F6F8B2CE76 )
; additional
                        85802   RRSIG   DS 8 1 86400 20180310170000 (
                                        20180225160000 41824 .
                                        DCOZKI8oR0CQAarvRbYP1v3IDCV5yQRP2e9A
                                        x735whZ2wYDGJVvwKiYDRvDOf8pjIb1FKVJA
                                        u7T46H7vRnRFAuCqH+M9uMaAQht4duBsMa7c
                                        pbt0DVhOCUuu39q9Jmw1fw0vEBh/46pbmDtO
                                        tGcoG6cFCN7d5eujQ6WFjMsA+prFuIT6QyIR
                                        nWl/U7a6EtWSsVjxkMI6pwzeT3ivmE37nXmT
                                        uhf0SvsCgN2F0dNuJTcmBA/aSH9b9pRveSTX
                                        3vt3v9vwPZXhNd5XHFxQ9CkzdXtoMrgsY225
                                        4wDzzty8j7EUdiZcQ5lBe15nlWBkEzS9+Hww
                                        RhlTw4BV0cvsTOR2SQ== )
; authauthority
example.edu.            64957   NS      ns.dnslabattacker.net.
; additional
                        85801   DS      44042 8 1 (
                                        043A6301C88EB7D22305905C6B63331BD147
                                        922A )
                        85801   DS      44042 8 2 (
                                        6F7520DCE4AB634085EF24C46D75343BF115
                                        375B241C4BB8A8B5AF0C67964E13 )
                        85801   DS      49729 8 1 (
                                        99030981F06EE333487465A00400628B6C96
                                        AE45 )
                        85801   DS      49729 8 2 (
                                        6FDC73554008CAFB80E2BBAC89CB6FC4D924
                                        1B8DE05F09D8F79727B12DCAA736 )
; additional
                        85801   RRSIG   DS 8 2 86400 20180303061720 (
                                        20180224050720 14375 edu.
                                        C7JIKXp7aCsiqc9GwIwee6jr1u63DUh3JMIL
                                        gDjy6cswPhr9VlXS0xrhDLQv77EnjxPpjqRM
                                        08qmiSdP17xfuXztmB8KMUNC4UOCwReJ+ZAg
                                                                                57,1-8        0%
```

Figure 1

Figure 1 shows the dump of `Apollo's` cache. It can be seen that the name server corresponding to `example.edu` has been changed to `ns.dnslabattacker.net`



```
...work Security/Lab2 — ssh cs528user@mc02.cs.purdue.edu -p 20401   dns_usr@cs528vm: /home/cs528user — ssh -X cs528user@mc02.cs...   ~ — ssh cs528user@mc02.cs.purdue.edu -p 20403
dns_usr@cs528vm:/home/cs528user$ dig www.example.edu

; <<>> DiG 9.8.1-P1 <<>> www.example.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19065
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.edu.        258444  IN      A       1.1.1.1

;; AUTHORITY SECTION:
example.edu.            64055   IN      NS      ns.dnslabattacker.net.

;; ADDITIONAL SECTION:
ns.dnslabattacker.net.  604800  IN      A       192.168.15.9
ns.dnslabattacker.net.  604800  IN      AAAA    ::1

;; Query time: 9 msec
;; SERVER: 192.168.15.7#53(192.168.15.7)
;; WHEN: Sun Feb 25 15:34:10 2018
;; MSG SIZE  rcvd: 128

dns_usr@cs528vm:/home/cs528user$
```

Figure 2

Figure 2 shows the output of running the `dig` command at `dns_usr`. It can be seen that the IP returned for [`www.example.edu`](http://www.example.edu) is `1.1.1.1` (the spoofed IP). Moreover, note that the IP of the name server is shown as `102.168.15.9`, which is `dns_attacker's` IP. Finally, note that the IP of the DNS server queried is `192.168.15.7` (`Apollo's`) IP. This verifies that the DNS server has been compromised and the attacker now has control over what IP to respond with.

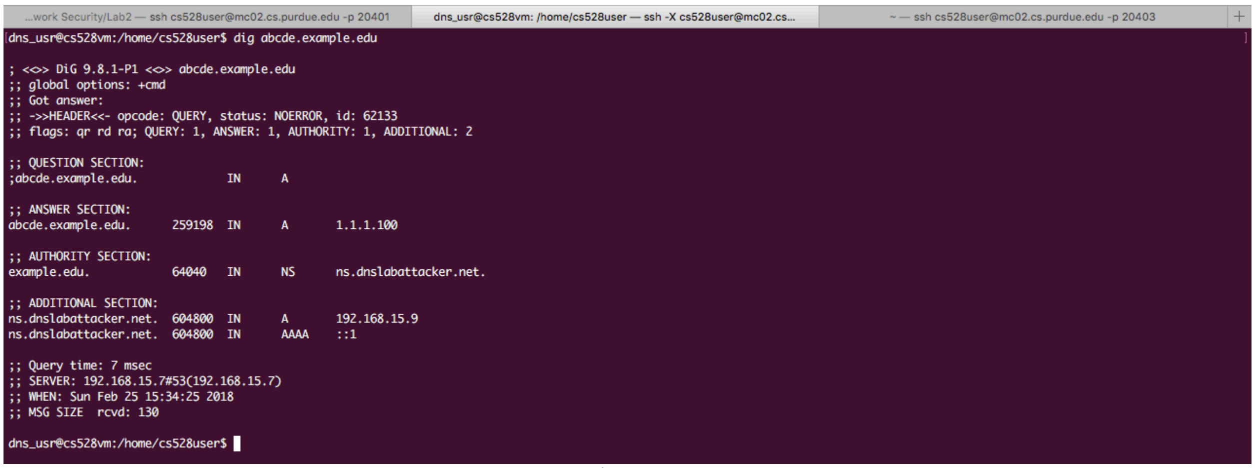


Figure 3

Figure 3 shows the output of the `dig` command for some other host but the same domain i.e. `example.edu`. In this case, we get a different IP address: `1.1.1.100`

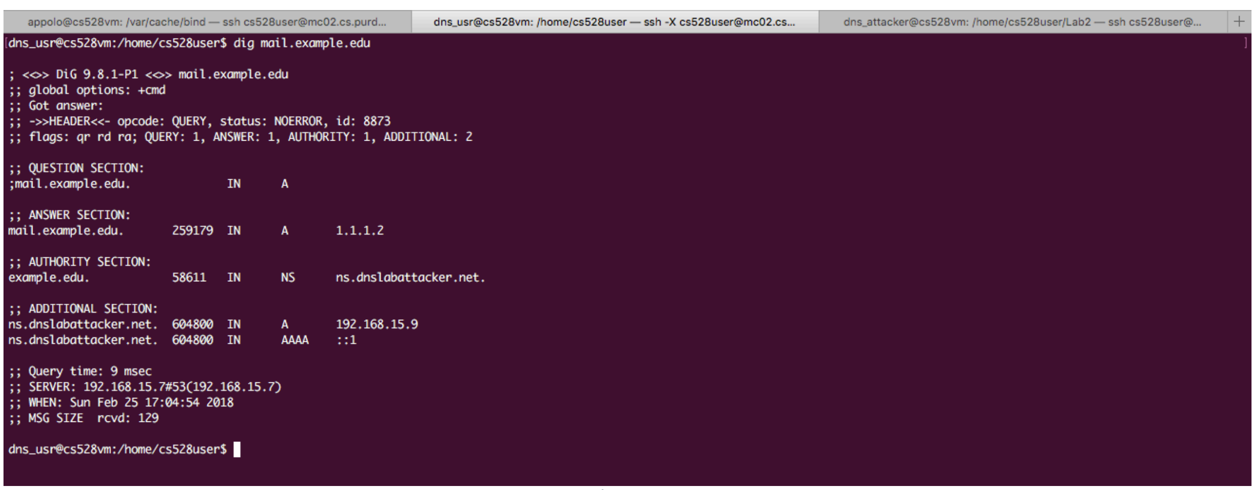


Figure 4

Figure 4 shows the output of the `dig` command after running `dig` for `mail.example.edu` at `dns_usr`. Again, a different but spoofed IP address is returned according to the attacker database we configured earlier.

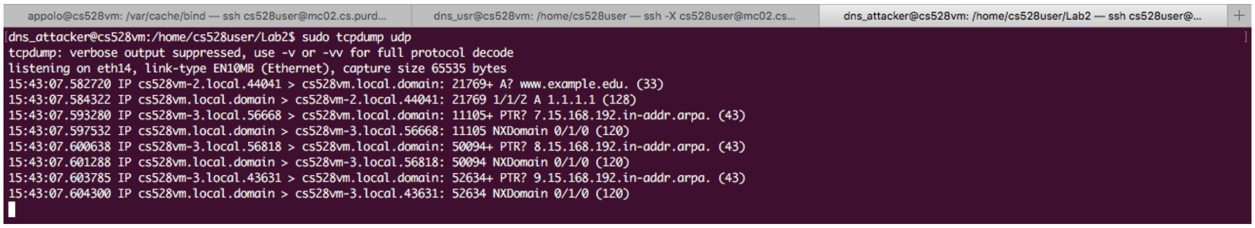


Figure 5

Figure 5 shows the output of running `tcpdump` on the `dns_attacker` machine. The first entry shows that a DNS query is sent to the attacker machine. The second entry shows that a DNS response is sent from the attacker machine to the machine `cs528vm-2` (which is the `dns_usr` machine).