CS4404

Mission Debriefing Report of Mission 2

Matthew Malone, Nour Elmaliki, Irakli Grigolia

Table of Contents

Introduction

This report will explore the security goals of DNS and describe how we accomplished the following:

1. Launched DNSredirect attack from perspective of an on path adversary(a scummy ISP)
2. Implemented DNSsec to prevent redirect attacks
3. Conducted a reflection/amplification dos attack on a website
4. Setup rate limiting to reduce the effectiveness of a reflection attack

Reconnaissance

Confidentiality, integrity, availability, and source authentication are security goals that are common to any electronic system. However, DNS is expected to provide name resolution information for any publicly available Internet resource. Hence in general DNS is not deemed confidential. Hence confidentiality is not one of the security goals of DNS. Ensuring the authenticity of information and maintaining the integrity of information in transit is critical for efficient functioning of the Internet, for which the DNS provides the name resolution service. Hence, integrity and source authentication are the primary DNS security goals.

1. Integrity

The purpose of DNS is to reliably provide address translation, reliability requires integrity. Customers must be able to trust that DNS replies have not been tampered with. DNS poisoning attacks alters DNS records, redirecting users to unintended destinations. DNSsec will prevent cache poisoning from taking place as communications between DNS nodes are encrypted, making it difficult for an attacker to impersonate a DNS response and poison the cache.

2. Availability

Another integral requirement for reliability is availability. DNS is the backbone of the internet, when a company's DNS server goes down, that company can no longer access the internet. A common attack is a random subdomain attack, in which a botnet requests random subdomains for an entity that does not exist. This ties up more DNS resources as they have to scan to ensure that the record does not exist. Availability attacks are difficult to combat in that it just requires more infrastructure to keep up with the attacks but rate limiting can be used to try to slow down the flood of packets.

3. Authenticity

In addition to message integrity, users must be confident in the authenticity of the DNS server. DNS hijacking is a common threat to authenticity. An attacker tries to redirect the target's DNS traffic to a hostile entity that will serve false records. This kind of hijacking forces the victim to visit arbitrary websites. DNSsec is a defense against DNS hijacking. If the potential victim knows the public key of the correct DNS server, then it can detect when a third party is serving a record, as they would be unable to encrypt the record with the correct private key.

Hence, the security objectives—and consequently the security services—that are required for securing the DNS query/response transaction are data origin authentication and data integrity verification.These services could be provided by establishing trust in the source and verifying the signature of the data sent by that source. The specification for a digital signature mechanism in the context of the DNS infrastructure is in IETF’s DNSSEC standard.

Section 1: Infrastructure Building

How to Replicate the Setup

To setup the attack, the VMs were configured as VM1 being the router the ISP (Bombast) uses for traffic from clients, VM2 being the DNS server (Separate from bombast for the sake of our goals), and VMs 3 and 4 being used for server hosting as well as VM3 being used as a client. Bind9 as a simple DNS server was installed on VM2 by following the documentation here: https://bind9.readthedocs.io/en/v9_16_7/. Python3 was installed on all the VMs to allow for ease of use. The necessary python dependencies we used such as NetfilterQueue and Scapy were downloaded from pypi.org and manually transferred to the VMs and installed as required. The following commands were run on each VM to set up the infrastructure after copying the necessary program files to each VM.
On VM 1 (10.21.8.1)
all with sudo privileges):
sysctl net.ipv4.ip_forward=1
Modify /etc/resolv.conf so the name server is 10.21.8.2
route add default gw 10.21.8.1
route add 10.21.8.2 gw 10.21.8.1
route add 10.21.8.3 gw 10.21.8.1
route add 10.21.8.4 gw 10.21.8.1
Sudo python3 attack.py (or normal.py for regular ISP behavior)
//Note that the scripts above run the interception rules and set up some of the necessary routes, if you see duplicate messages, do not worry about it.

On VM 2 (10.21.8.2)
sudo service bind9 restart

On VM3 and 4 (10.21.8.3 and 10.21.8.4)
Modify /etc/resolv.conf so the name server is 10.21.8.2
route add default gw 10.21.8.1
route add 10.21.8.2 gw 10.21.8.1

navigate to the bombastServer directory on VM3 and type sudo python3 server.py &   navigate to the shueServer directory on VM4 and type sudo python3 server.py &
run dig or nslookup on bombast.com or shueISP.com on VM3

To set up bind9, use the included configuration files and zone files found in the VM2.zip folder attached to the project. You will need the named.conf.* files and the zones.

Section 2: Attacking DNS

We explored the possibilities of several attacks and chose to execute DNS hijacking attack as part of MITM attack and also DNS Amplification Attack as one of the most interesting and realistic ones.

DNS Hijacking / Redirection Attack

Domain Name Server (DNS) hijacking, also known as DNS redirection or DNS Poisoning, is a type of DNS attack in which DNS queries are subverted in order to redirect users to malicious sites or in our case to Bombast web servers. To execute the attack, attackers can install malware on user computers, take over routers, or intercept or hack DNS communication. The last option seemed the most feasible to us to model.

DNS hijacking can be used for pharming (to display unwanted ads to generate revenue) or for phishing (to display fake websites and steal data or credentials).

Many Internet Service Providers (ISPs) such as Bombast make use of DNS hijacking, to take over a user’s DNS requests, collect statistics and return ads when users access an unknown domain or in our case, completely redirecting all traffic from competitor ISP websites to bombast.com. This was accomplished by writing a python script that uses NetfilterQueue to redirect all traffic on port 53 (the port reserved for DNS queries) to NFQUEUE which in turn let us access the packets in the python script. From there, the packets were audited, and if they contained URLs on the blacklist, the packets were modified to redirect the client to bombast.com, otherwise the traffic was allowed to continue as normal. See appendix 1 for proof of this attack working, and take a look at the attached code for attack.py to get an idea of how this works. Also read the How We Setup the DNS Redirect Attack segment below.

There are also different types of hijacking attacks.

DNS Hijacking Attack Types

- **Local DNS hijack** — Trojan malware is installed on a user’s computer, and DNS settings are changed to redirect the user to different web sites.

- **Router DNS hijack** — Attackers make use of Routers' week default passwords or firmware vulnerabilities. Attackers overwrite DNS settings, affecting all users connected to that router.
- **Man in the middle DNS attacks** — Attackers intercept communication between a client and a DNS server, and provide different destination IP addresses pointing to different web sites. (The one we are actually doing. In this case the ISP itself is acting as a router for traffic and using itself as an on-path adversary).
- **Rogue DNS Server** — Attackers actually hack a DNS server, change DNS records and redirect DNS requests to different web sites.

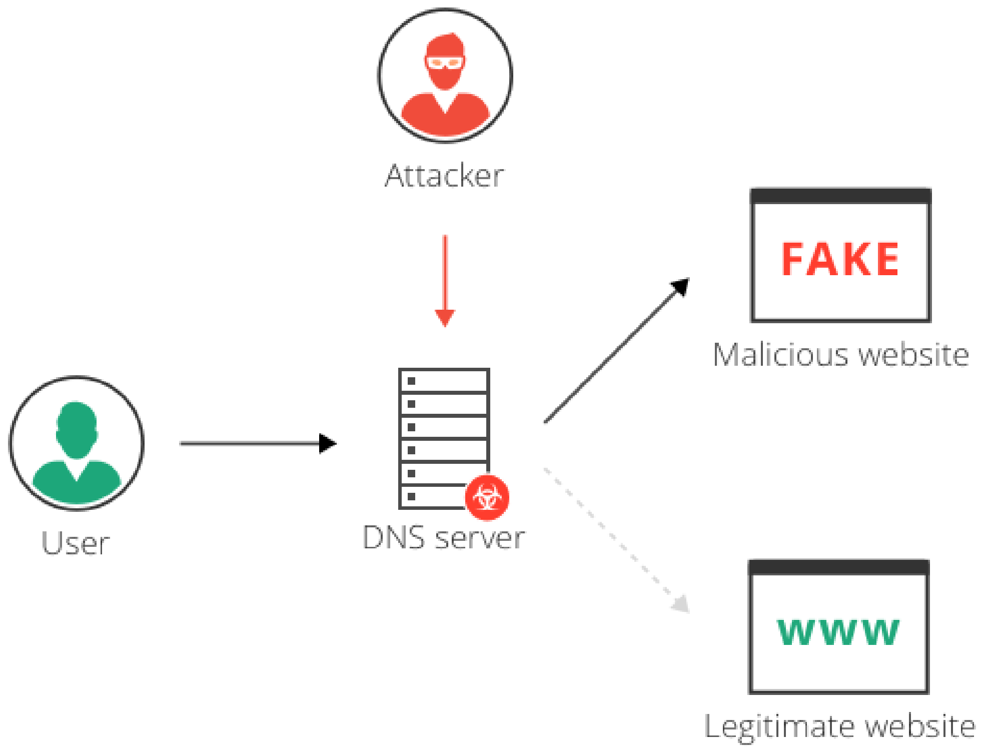


Figure 1 (DNS hijacking attack)

How We Setup the DNS Redirect Attack

In order to successfully run this attack, two python tools were used. NetfilterQueue, and Scapy. NetfilterQueue allows redirection of traffic to programs by using iptables rules to redirect traffic from a network device to NFQUEUE. Scapy was used to analyze these packets and redirect and modify them as necessary. Both were copied from the pip repository online alongside with their dependencies and installed on VM1 manually. Next it was a matter of writing the interception script attack.py as found in the attached files. It begins by modifying the system to enable ipv4 forwarding and setting up iptable rules for NetfilterQueue to run effectively. For our cases, we specifically set it to only intercept DNS traffic to trim out the fluff and reduce delays. Then it was a matter of writing the code to detect whether or not the DNS requests contained a site on the blacklist (in this case the competitor shueISP) and modify the packets to redirect you to bombast as necessary. Originally we tried using NetfilterQueue's accept and deny functionality to reduce load times, however we quickly realized it was incompatible with the Linux kernel for Ubuntu 20+. Instead we found a solution by using scapy to copy the packets verbatim then send them to the necessary DNS server. This was made possible due to the fact that NFQUEUE was only taking in DNS traffic on port 53, so it was easy to manually redirect the packets rather than rely on NetfilterQueue to accept or deny them.

How to Replicate the DNS Redirect Attack

Start by configuring each of the VMs as described in the infrastructure segment. After configuring bind9 as per the documentation and setting up the zone files (see attached zone files in zip and configuration files), restart the service on VM2 with 'sudo service bind9 restart'. Next on VM1 navigate to the ispData directory and execute the following command: 'sudo python3 attack.py'. If you configured the VMs correctly, this should launch without issue. See screenshot 1 from the appendix for a visual representation of what the script looks like while running. Next, on VM3 run the command 'curl bombast.com' and see that it returns the webpage as specified to bombast. Next try running 'curl shueISP.com' and watch as the page for bombast.com appears. See screenshot 2 from the appendix for proof that the attack worked successfully.

This was accomplished by using VM1 as a router for packets and then intercepting all DNS traffic and spoofing it as necessary using the attack.py script. All traffic to any domain related to shueISP.com is redirected to bombast.com. In our implementation, we assumed that the ISP did not have control over the DNS server. Otherwise the ISP could simply overwrite the zone files to redirect traffic automatically, and that was deemed too boring for an attack.

DNS Amplification/Reflection Attacks

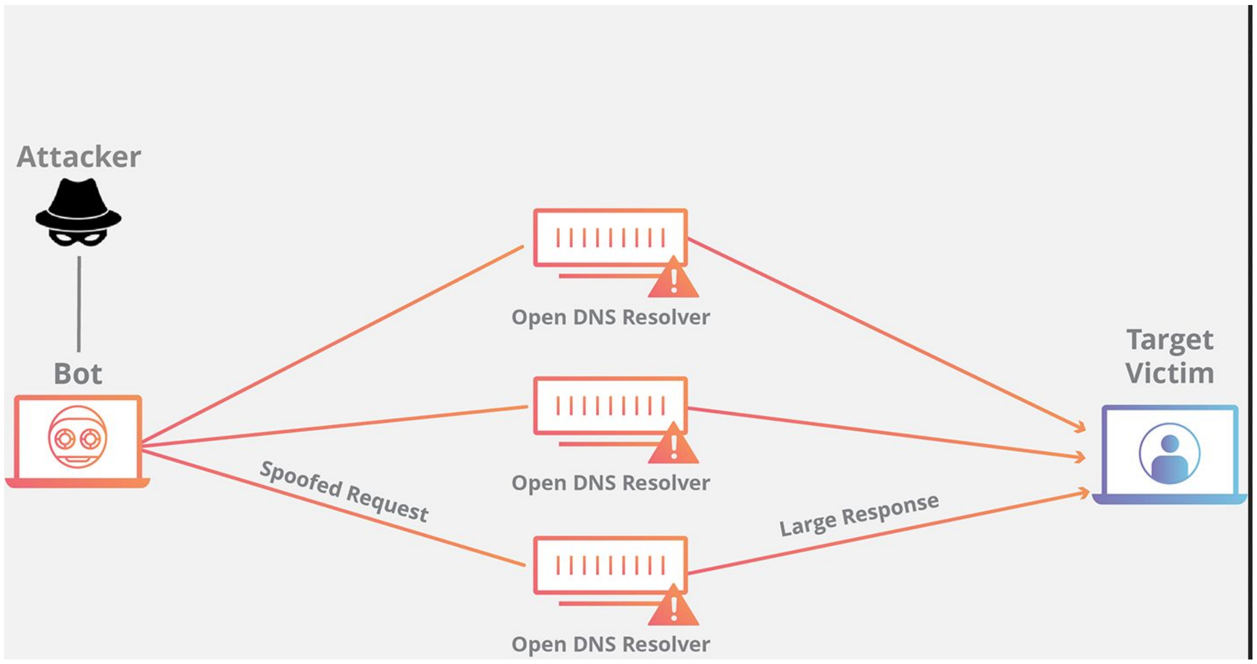


Figure 2: DNS Amplification Attack

A Domain Name Server (DNS) Amplification attack is a popular form of Distributed Denial of Service (DDoS), in which attackers use publicly accessible open DNS servers to flood a target system with DNS response traffic. In our case, due to the system being run on WPI owned hardware, we decided to go for a simple DOS attack, and only send a few packets at a time to simulate the concept.

DNSSEC is useful for mitigating the risk of DNS spoofing, because it helps to verify DNS requests. However, it does not prevent Distributed Denial of Service (DDoS) attacks leveraging DNS servers. In contrast, DNSSEC can potentially amplify the effects of DDoS. Responses sent for DNS queries are larger in DNSSEC due to additional fields and cryptographic information used to verify records. The larger responses can allow an attacker to attain as much as 70 times the attack volume with the same bandwidth, compared to regular DNS.

All amplification attacks exploit a disparity in bandwidth consumption between an attacker and the targeted web resource. When the disparity in cost is magnified across many requests, the resulting volume of traffic can disrupt network infrastructure. By sending small queries that result in large responses, the malicious user is able to get more from less. By multiplying this magnification by having each bot in a botnet make similar requests, the attacker is both safe from detection and benefits of greatly increased attack traffic.

While the most common form of this attack has been observed involves DNS servers configured to allow unrestricted recursive resolution for any client on the Internet, attacks can also involve authoritative name servers that do not provide recursive resolution. The attack method is similar to open recursive resolvers, but is more difficult to mitigate since even a server configured with best practices can still be used in an attack. In the case of authoritative servers, mitigation should focus on using Response Rate Limiting to restrict the amount of traffic.

How to Run the Reflection Attack

See the attacked reflection.py file and import it to VM3 after setting up the VMs as described in the infrastructure. Note that you will need python and scapy installed in order to run this script. Simply run the python script with root privileges and watch the TCPdump on VM1. See attached screenshots 9-11.

Setup VMs as described above. Use scapy to manually create packets. Specify the IP source as the target, the destination port as 53, and create a dns request using the DNSQR function to structure the request. The qtype argument decides the type of requests, usually ‘A’ or “txt” for text records. Send the packets to the dns and use tcpdump on the target machine to observe DNS unprovoked packets from the DNS server. Send as many packets as you can get away with

How the Reflection Attack Works

The amplification part of the attack is decided by the actual forged request. The larger the request, the more effective the attack. For our attack the packet is set to query for a slightly larger response from the DNS server (**seen in screenshot 12**) . In our case we used a TXT record stored on the DNS zone file due to DNSSEC not being finalized at the time. In actuality, a real reflection attack would take advantage of the large signature strings in the protected zone file to launch a reflection attack. We spoofed this before DNSSEC was finished by simply launching a reflection at a large TXT record which effectively accomplished the same thing. This recommendation to use a large TXT record was made by professor Shue. This response is only slightly larger than a normal record to demonstrate that amplification is possible. The script works by simply querying the DNS server for these records 10 times in quick succession as a proof of concept since we were disappointingly not actually allowed to perform a real denial of service attack on the WPI network.

Section 3: Defense

DNSSEC

DNS is an unencrypted protocol, making it easy to intercept traffic with spoofing. What’s more, DNS servers do not validate the IP addresses to which they are redirecting traffic.

DNSSEC is a protocol designed to secure your DNS by adding additional methods of verification. The protocol creates a unique cryptographic signature stored alongside your other DNS records. This signature is then used by your DNS resolver to authenticate a DNS response, ensuring that the record wasn't tampered with.

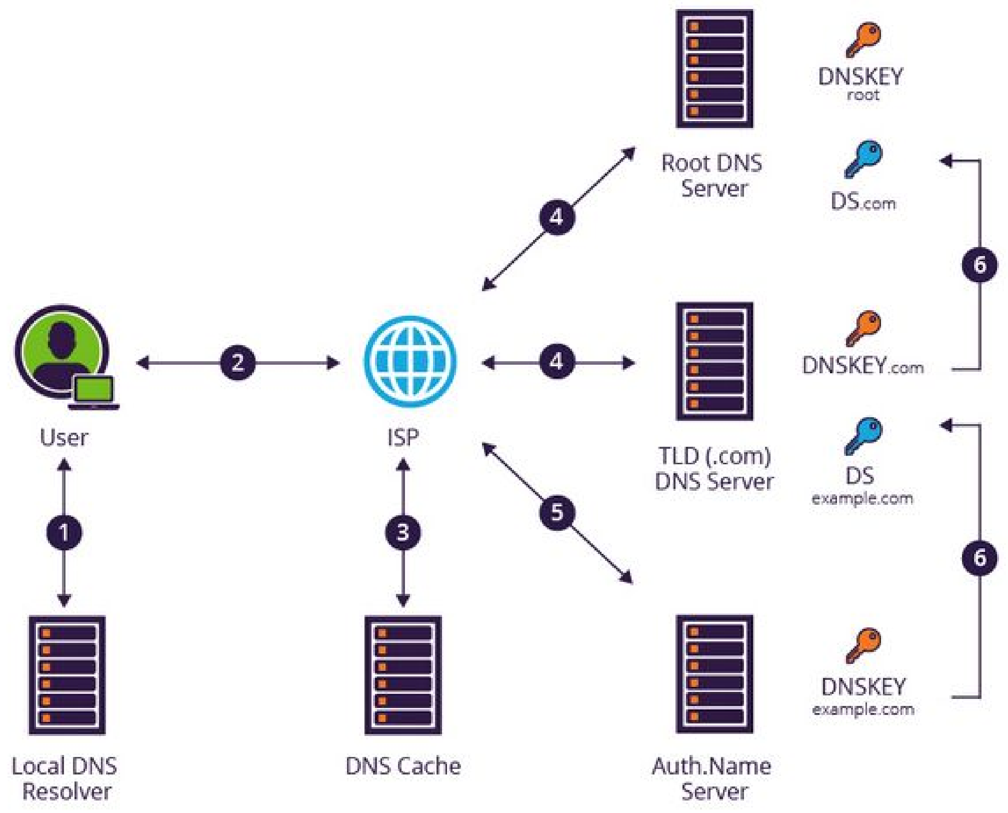


Figure 3: DNSSEC Validation

DNSSEC ,using digital signature technology, verifies the authenticity of responses sent by name servers to clients. DNSSEC adds cryptographic signatures to DNS records, which protects data published in the DNS. With DNSSEC, the DNS resolver verifies its authenticity by checking the signature associated with a record, before serving responses to clients. All records must match those stored on an authoritative DNS server.

Setting Up the Defense

To set up the defense, we configured VM2 with our bind9 server to have DNSSEC for connections made to shueISP.com. Because our server was implied to be separate from Bombast, we didn't feel the necessity in the time frame to set up DNSSEC for them. ShueISP is a better company anyways. We decided to use the resources found here to set this up: https://blog.apnic.net/2019/05/23/how-to-deploying-dnssec-with-bind-and-ubuntu-server/. Firstly we modified the named.conf files to enable DNSSEC, and also configure the zone for ShueISP to use it. See appendix 3 and 4. After this, a folder called keys was made in the folder with the configuration files to store the necessary keys. DNSSEC relies on sets of public and private keys to verify signatures and traffic. Next, we ran the following commands to generate keys in the keys folder:

```
dnssec-keygen -a RSASHA256 -b 1024 -n ZONE shueISP.com
dnssec-keygen -a RSASHA256 -b 2048 -fKSK -n ZONE shueISP.com
```

Next we signed the keys manually by navigating to /etc/bind/keys/ and running:

```
dnssec-signzone -o <key for zone files> -N increment -S ../zones/db.shueISP.com
rndc reload
rndc sign shueISP.com
```

For our intents and purposes, everything was stored in the /etc/bind/ folder to make things easier and cleaner. After the keys were generated, and the zones were signed, we modified the named.conf.local file to serve the signed domain instead of the unsigned one for shueISP.com.

To set up a chain of trust we relied on bind9's automatic DNS recursion as documented here: https://dnsinstitute.com/documentation/dnssec-guide/ch03s04.html and https://dnsinstitute.com/documentation/dnssec-guide/ch03.html#easy-start-guide-for-recursive-servers. The key for the root dot server can be found in the bind.keys file automatically while the rest of the validation was done manually by setting up records as shown above. This ensures that the DNS server verifies each DNS response with a chain of trust through the manually configured zone records all the way through the dot record.

Verifying DNSSEC

To test whether or not the key was enabled, we installed ldnsutils and ran:

```
drill -D shueISP.com OR drill -k <public zone-signing key> -D shueISP.com
```

The "-D" flag tells the drill command to use DNSSEC.

The query in appendix 5 was the result. Note how the RRSIG appears alongside the A record for shueISP.com. This signifies that DNSSEC is enabled and working for the domain. See screenshot 6 for a dig of DNSKEY records. These records are the public keys so clients can verify responses from the DNS server. Screenshot 7 shows us using the public keys to verify the DNS server. Screenshot 8 demonstrates what happens when an attacker has performed a dns hijack and can't verify the true DNS's public key. Unfortunately due to time constraints we were not able to completely protect against this defense, as a proper defense would involve DNS over HTTPS or IPSEC. However just the implementation alone of DNSSEC serves as a red flag to a user and any software they are using that their traffic has been tampered with.

Rate Limiting

This defense was aimed at reducing the impact of the reflection/amplification attack. Newer versions of bind9 come with a rate limiting option. However, for our defense we decided to implement rate limiting ourselves. We pretended that our ISP had decided to offer rate limiting per source services to their DNS. We used NetfilterQueue to intercept packets and scapy to dissect the packets. We mimicked bind9's fetches-per-server limiting so we counted the number requests from each server in a given period of time and dropped excessive packets. The code for the rate limiter can be seen in ispData/rateLimiting.py on VM1.

To run our defense, on the ISP machine(VM 1) w/ sudo run:
Python3 rateLimiting.py. -c max_packets -t time_period

NOTE: max_packets is the max number of packets allowed from a source within a time_period in seconds.

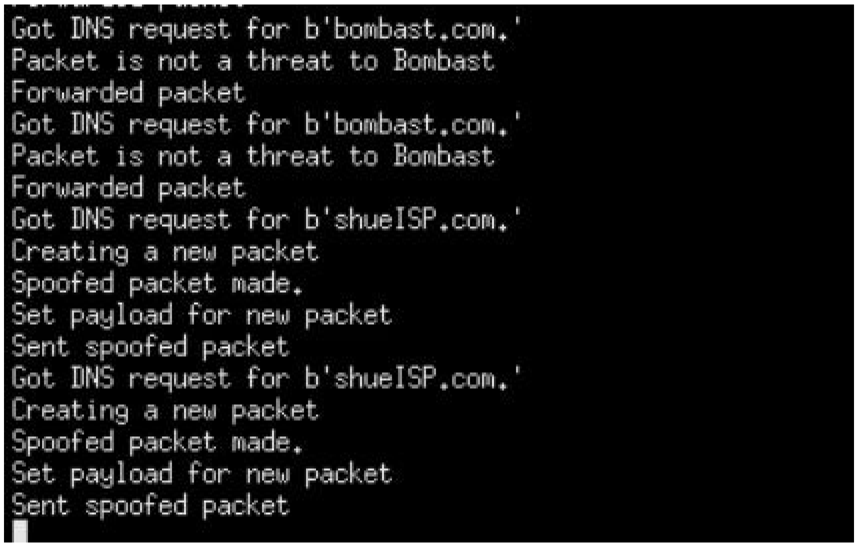
Screenshot 13 demonstrates rate limiting in action, only allowing *c* number of packets in a *t* time period and dropping the rest. Screenshot 14 shows the would-be victim's perspective, Despite the attacker sending 7 spoofed packets, the would-be victim only 4. Obviously in the real world the volume of packets would be significantly larger and the DNS server would likely expect more than 4 requests every 20 seconds, but this is a miniature replica of how rate limiting could work.

Conclusion

DNS has been demonstrated to be vulnerable to a host of potential attacks, specifically DNS redirection. We have seen how an on-path adversary can redirect DNS traffic and send users to an attacker controlled webpage. While there are several ways to reduce this threat, DNSsec verifies the DNS host and can alert users to when their traffic has been redirected. Unfortunately, DNSsec is not widely used due to its performance limitations. Currently, similar methods are being explored such as DNS over HTTPS and DNS over TLS, ideally one of these will become standard in the future.

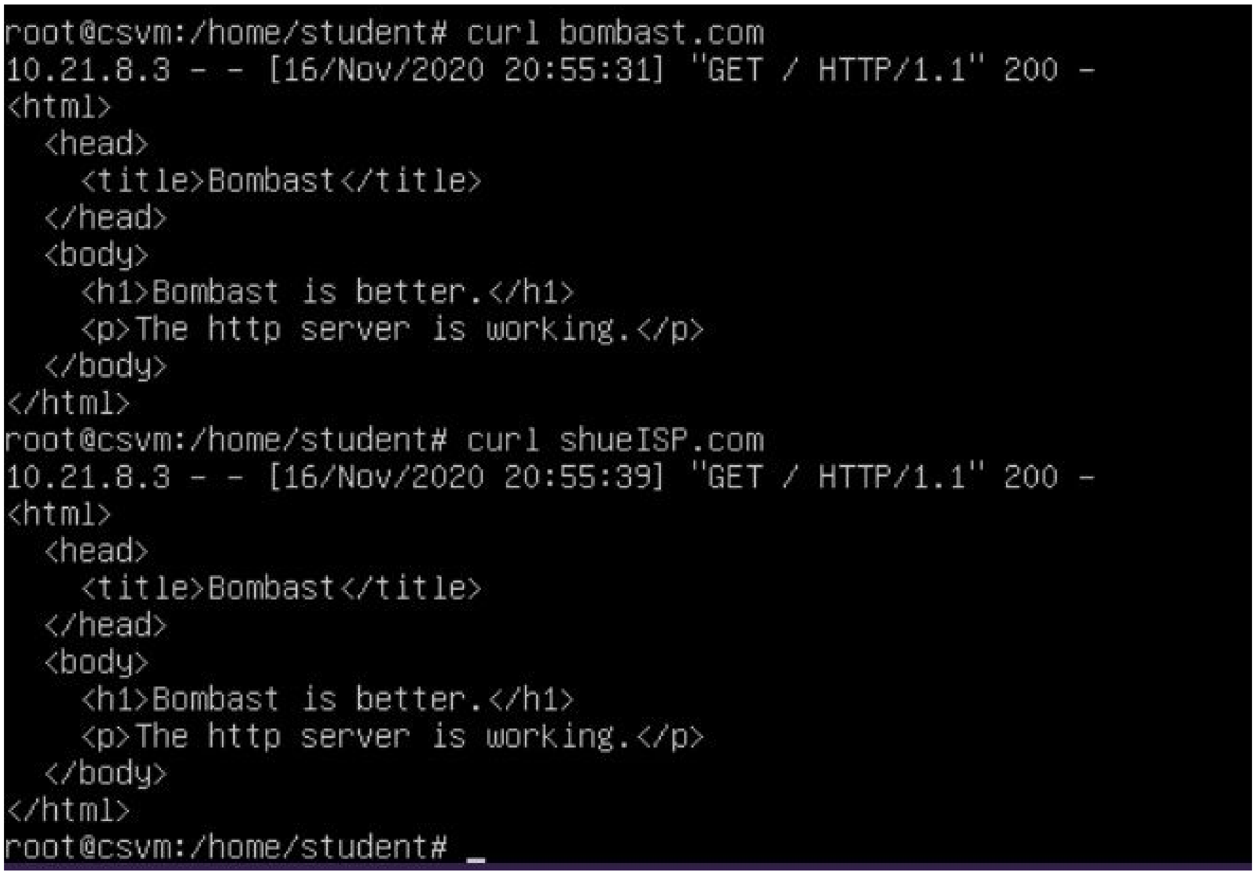
Additionally, DNS amplification can be used in denial of service attacks. The specific DNS request decides the level of amplification and attackers have to find a balance between maximum output(like request all records) and remaining innocuous. Larger requests are more suspicious. Defending against reflection attacks are difficult, as there's a tradeoff between security and blocking legitimate traffic. We explored rate limiting and found success in our somewhat limited example. Researchers have found other ways of responding to reflection attacks but the best defense to DDOS attacks in general is to have enough resources to deal with malicious and benign traffic simultaneously.

Appendix



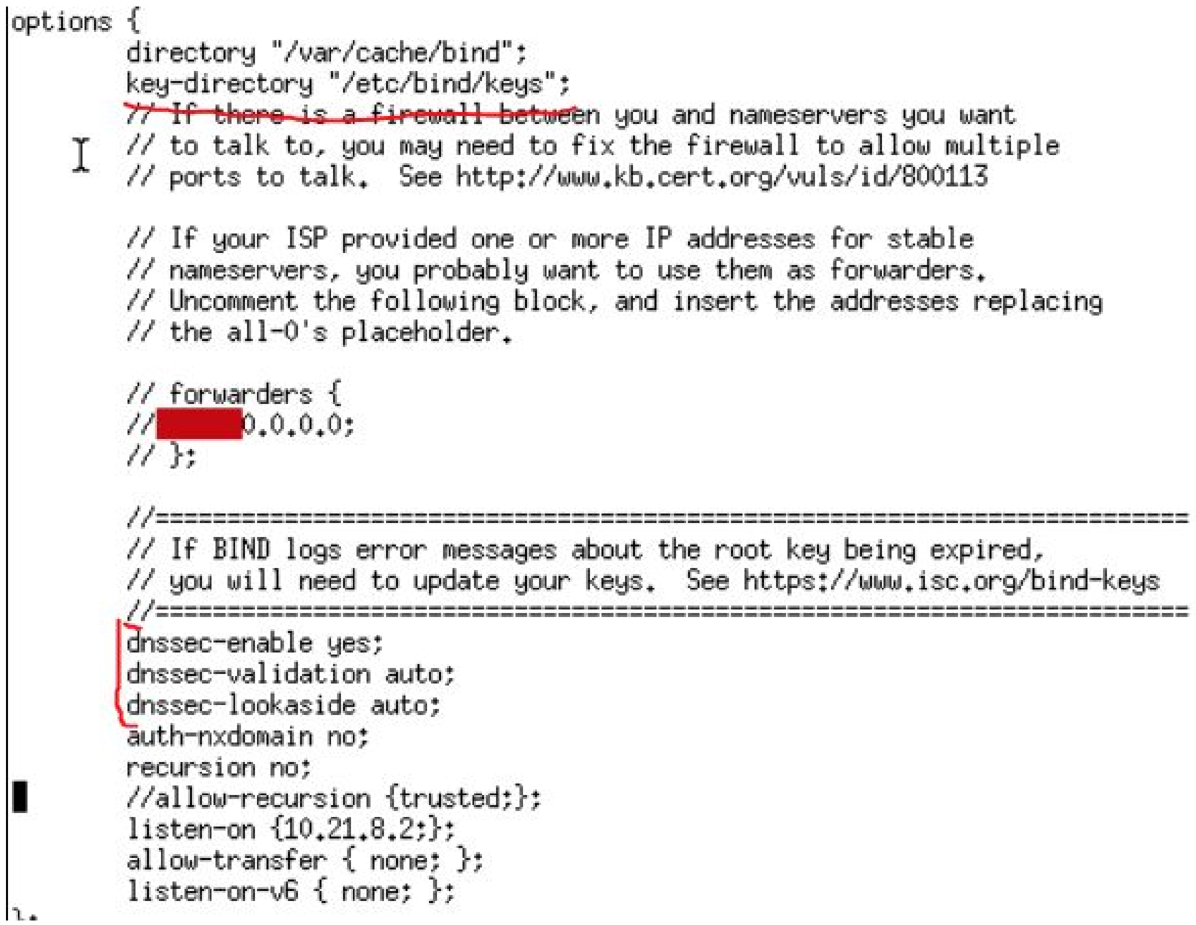
```
Got DNS request for b'bombast.com.'
Packet is not a threat to Bombast
Forwarded packet
Got DNS request for b'bombast.com.'
Packet is not a threat to Bombast
Forwarded packet
Got DNS request for b'shueISP.com.'
Creating a new packet
Spoofed packet made.
Set payload for new packet
Sent spoofed packet
Got DNS request for b'shueISP.com.'
Creating a new packet
Spoofed packet made.
Set payload for new packet
Sent spoofed packet
```

Screenshot 1: Python attack.py script being executed on VM1



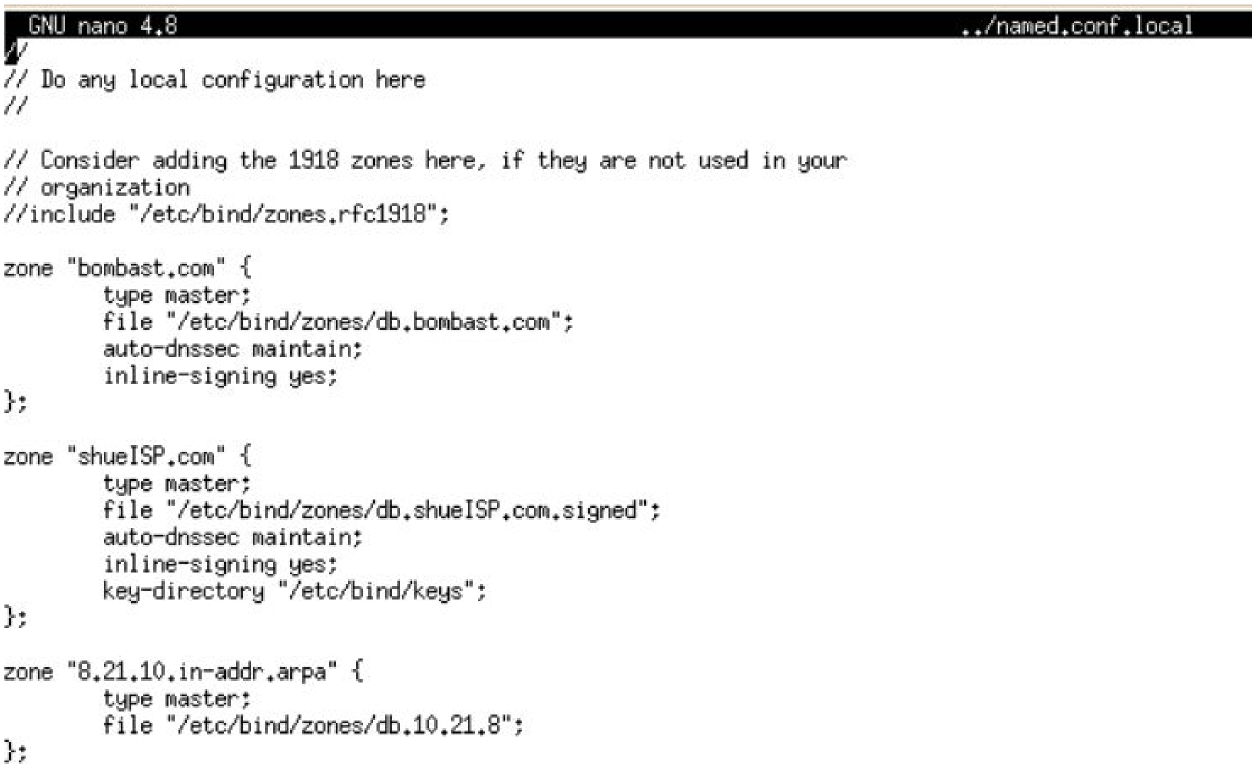
```
root@csvm:/home/student# curl bombast.com
10.21.8.3 - - [16/Nov/2020 20:55:31] "GET / HTTP/1.1" 200 -
<html>
  <head>
    <title>Bombast</title>
  </head>
  <body>
    <h1>Bombast is better.</h1>
    <p>The http server is working.</p>
  </body>
</html>
root@csvm:/home/student# curl shueISP.com
10.21.8.3 - - [16/Nov/2020 20:55:39] "GET / HTTP/1.1" 200 -
<html>
  <head>
    <title>Bombast</title>
  </head>
  <body>
    <h1>Bombast is better.</h1>
    <p>The http server is working.</p>
  </body>
</html>
root@csvm:/home/student# _
```

Screenshot 2: Attack working successfully as queried from VM3



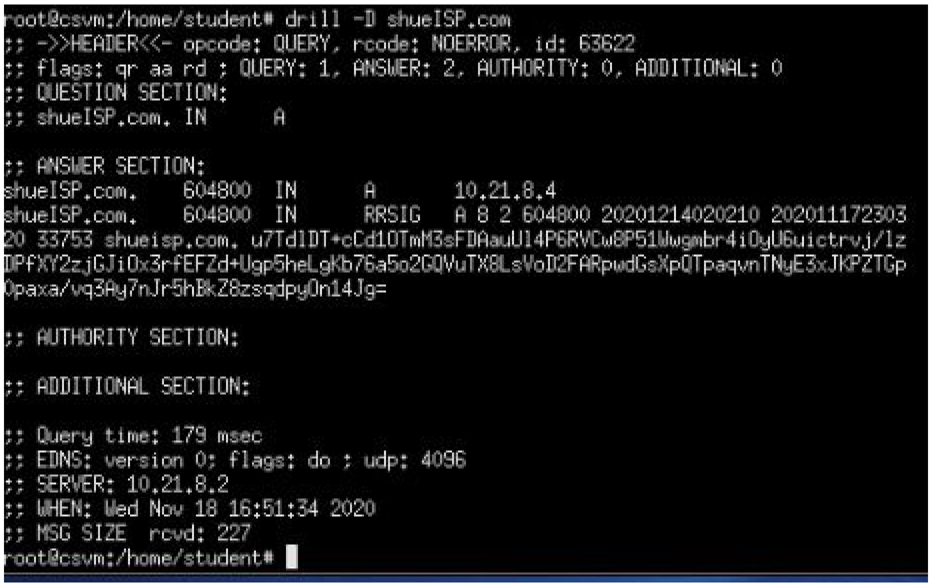
```
options {
        directory "/var/cache/bind";
        key-directory "/etc/bind/keys";
        ~~// If there is a firewall between~~ you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //[redacted]0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================
        dnssec-enable yes;
        dnssec-validation auto;
        dnssec-lookaside auto;
        auth-nxdomain no;
        recursion no;
        //allow-recursion {trusted;};
        listen-on {10.21.8.2;};
        allow-transfer { none; };
        listen-on-v6 { none; };
```

Screenshot 3: Enabling DNSSEC in the configuration files.



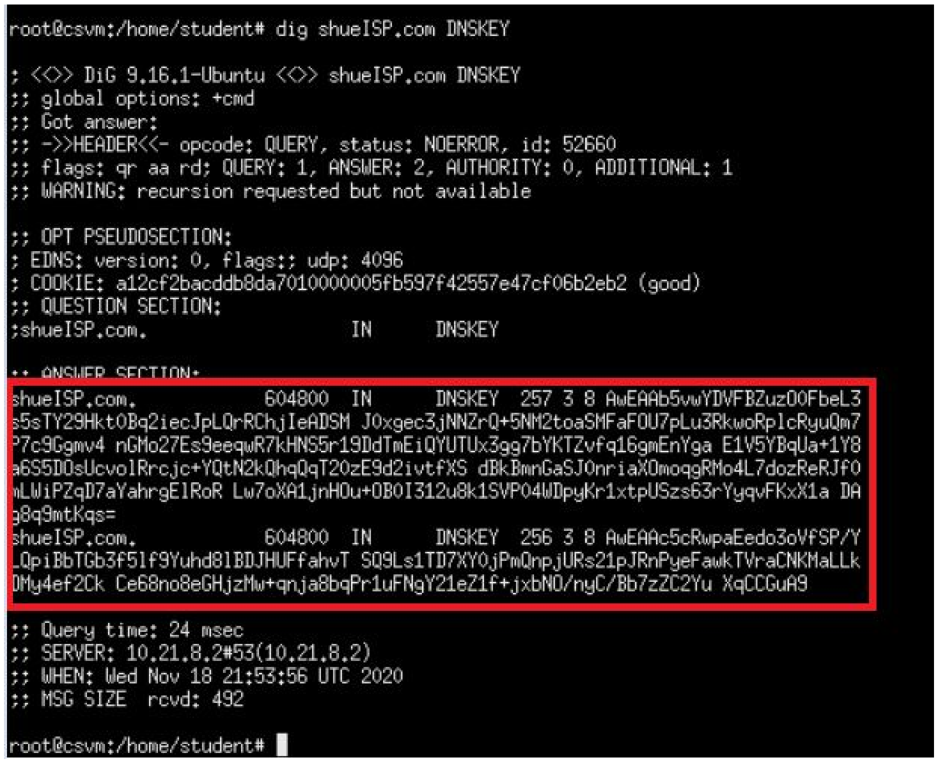
```
  GNU nano 4.8                                                     ../named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "bombast.com" {
        type master;
        file "/etc/bind/zones/db.bombast.com";
        auto-dnssec maintain;
        inline-signing yes;
};

zone "shueISP.com" {
        type master;
        file "/etc/bind/zones/db.shueISP.com.signed";
        auto-dnssec maintain;
        inline-signing yes;
        key-directory "/etc/bind/keys";
};

zone "8.21.10.in-addr.arpa" {
        type master;
        file "/etc/bind/zones/db.10.21.8";
};
```

Screenshot 4: Enabling DNSSEC in zone files.



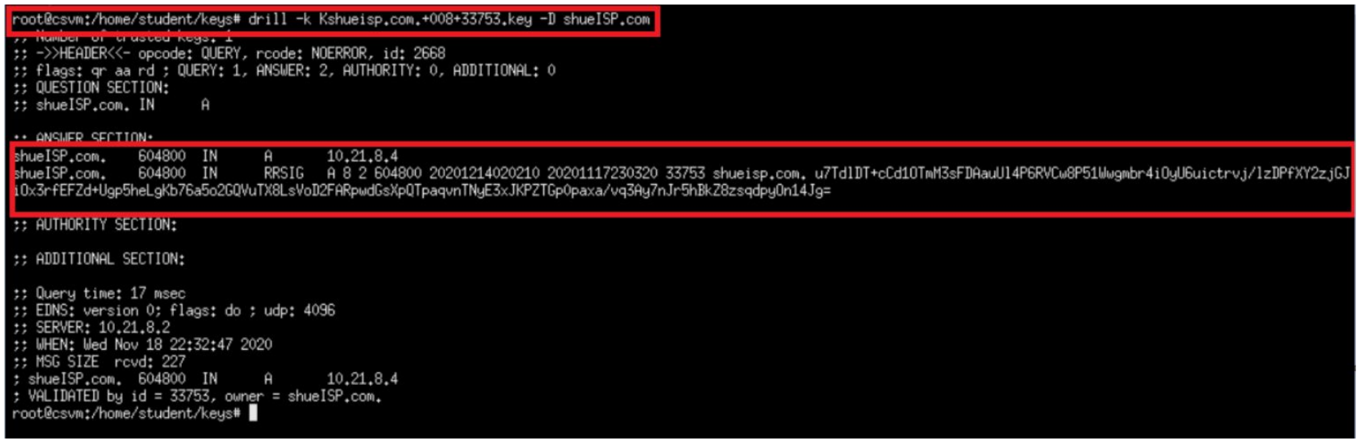
```
root@csvm:/home/student# drill -D shueISP.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 63622
;; flags: qr aa rd ; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; shueISP.com. IN      A

;; ANSWER SECTION:
shueISP.com.    604800  IN      A       10.21.8.4
shueISP.com.    604800  IN      RRSIG   A 8 2 604800 20201214020210 202011172303
20 33753 shueisp.com. u7Td1DT+cCd1OTmM3sFDAauU14P6RVCw8P51Wwgmbr4iOyU6uictrvj/lz
IPfXY2zjGJiOx3rfEFZd+Ugp5heLgKb76a5o2GQVuTX8LsVoD2FARpwdGsXpQTpaqvnTNyE3xJKPZTGp
0paxa/vq3Ay7nJr5hBkZ8zsqdpyOn14Jg=

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 179 msec
;; EDNS: version 0; flags: do ; udp: 4096
;; SERVER: 10.21.8.2
;; WHEN: Wed Nov 18 16:51:34 2020
;; MSG SIZE  rcvd: 227
root@csvm:/home/student# 
```

Screenshot 5: RRSIG in drill response signifying that DNSSEC is enabled for the domain.



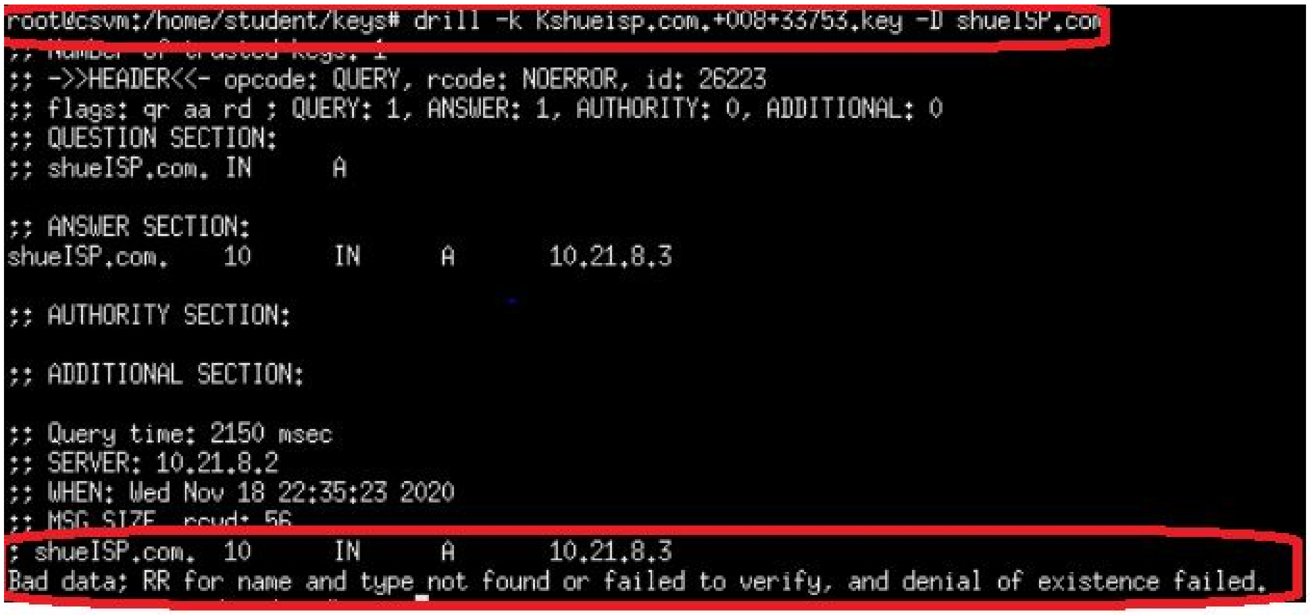
```
root@csvm:/home/student# dig shueISP.com DNSKEY

; <<>> DiG 9.16.1-Ubuntu <<>> shueISP.com DNSKEY
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52660
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: a12cf2bacddb8da7010000005fb597f42557e47cf06b2eb2 (good)
;; QUESTION SECTION:
;shueISP.com.                   IN      DNSKEY

;; ANSWER SECTION:
shueISP.com.            604800  IN      DNSKEY  257 3 8 AwEAAb5vwYDVFBZuzOOFbeL3
s5sTY29HktOBq2iecJpLQrRChjIeADSM JOxgec3jNNZrQ+5NM2toaSMFaFOU7pLu3RkwoRplcRyuQm7
P7c9Ggmv4 nGMo27Es9eeqwR7kHNS5r19DdTmEiQYUTUx3gg7bYKTZvfq16gmEnYga E1V5YBqUa+1Y8
a6S5DOsUcvolRrcjc+YQtN2kQhqQqT20zE9d2ivtfXS dBkBmnGaSJOnriaXOmoqgRMo4L7dozReRJfO
mLWiPZqD7aYahrgElRoR Lw7oXA1jnHOu+OBOI312u8k1SVPO4WDpyKr1xtpUSzs63rYyqvFKxX1a DA
g8q9mtKqs=
shueISP.com.            604800  IN      DNSKEY  256 3 8 AwEAAc5cRwpaEedo3oVfSP/Y
LQpiBbTGb3f51f9Yuhd81BDJHUFfahvT SQ9Ls1TD7XYOjPmQnpjURs21pJRnPyeFawkTVraCNKMaLLk
DMy4ef2Ck Ce68no8eGHjzMw+qnja8bqPr1uFNgY21eZ1f+jxbNO/nyC/Bb7zZC2Yu XqCCGuA9

;; Query time: 24 msec
;; SERVER: 10.21.8.2#53(10.21.8.2)
;; WHEN: Wed Nov 18 21:53:56 UTC 2020
;; MSG SIZE  rcvd: 492

root@csvm:/home/student# 
```

Screenshot 6: Dig of shueISP.com showing DNSKEY records.



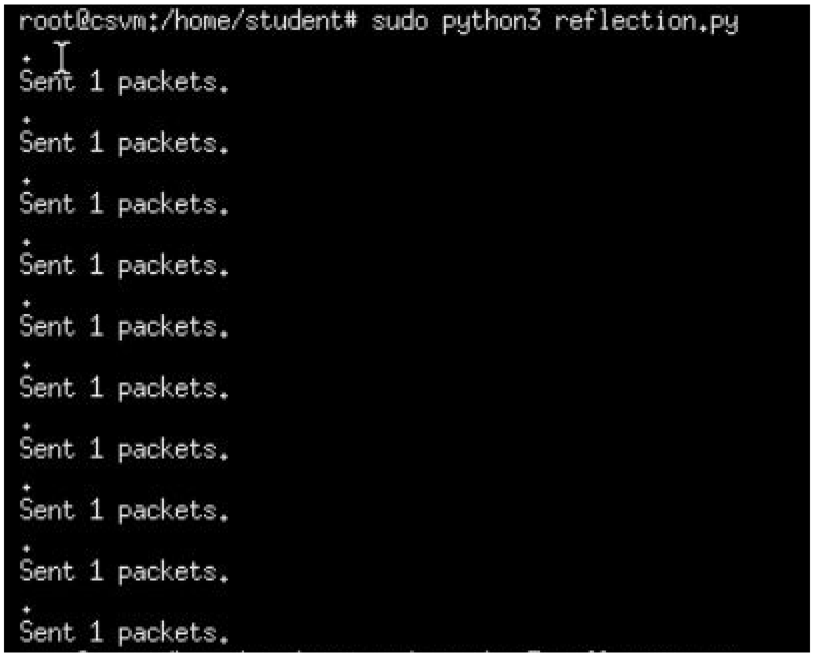
```
root@csvm:/home/student/keys# drill -k Kshueisp.com.+008+33753.key -D shueISP.com
;; Number of trusted keys: 1
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 2668
;; flags: qr aa rd ; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; shueISP.com. IN      A

;; ANSWER SECTION:
shueISP.com.    604800  IN      A       10.21.8.4
shueISP.com.    604800  IN      RRSIG   A 8 2 604800 20201214020210 20201117230320 33753 shueisp.com. u7TdlDT+cCd1OTmM3sFDAauUl4P6RVCw8P51Wwgmbr4iOyU6uictrvj/lzDPfXY2zjGJ
iOx3rfEFZd+Ugp5heLgKb76a5o2GQVuTX8LsVoD2FARpwdGsXpQTpaqvnTNyE3xJKPZTGpOpaxa/vq3Ay7nJr5hBkZ8zsqdpyOn14Jg=

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 17 msec
;; EDNS: version 0; flags: do ; udp: 4096
;; SERVER: 10.21.8.2
;; WHEN: Wed Nov 18 22:32:47 2020
;; MSG SIZE  rcvd: 227
; shueISP.com.  604800  IN      A       10.21.8.4
; VALIDATED by id = 33753, owner = shueISP.com.
root@csvm:/home/student/keys# 
```

Screenshot 7: Verified response from a dnssec service



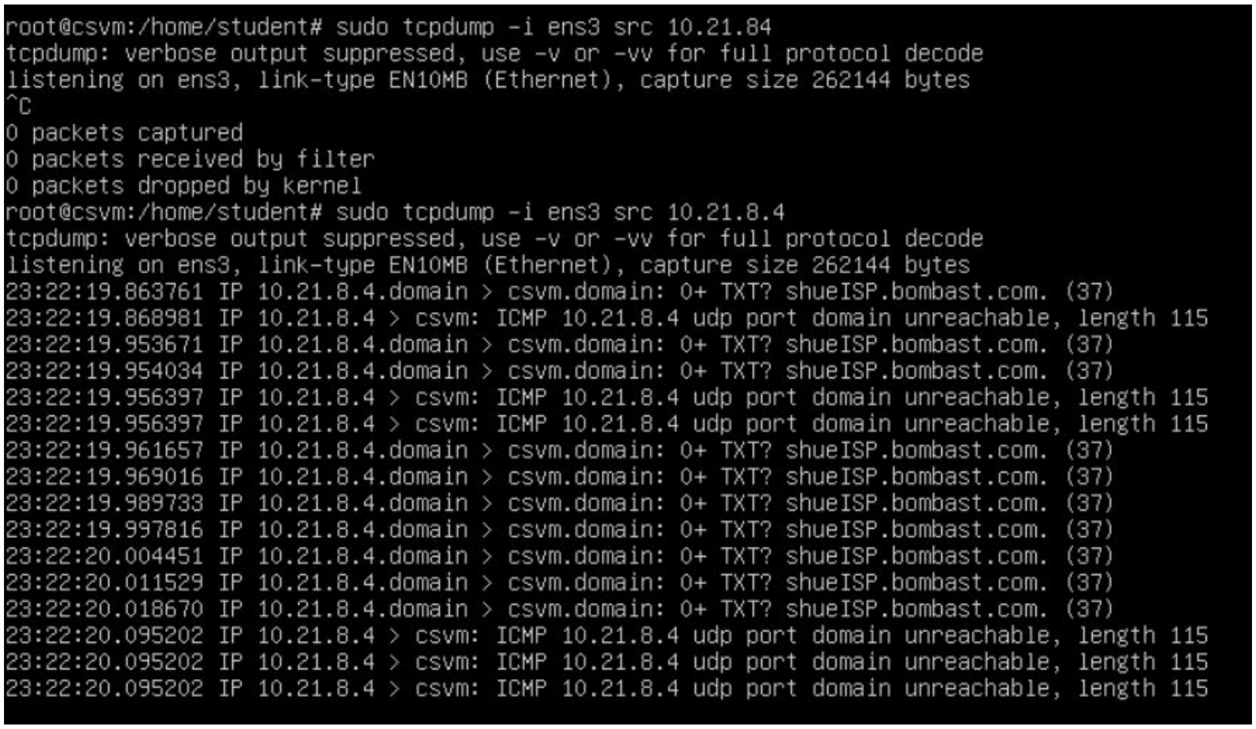
```
root@csvm:/home/student/keys# drill -k Kshueisp.com.+008+33753.key -D shueISP.com
;; Number of trusted keys: 1
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 26223
;; flags: qr aa rd ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; shueISP.com.	IN	A

;; ANSWER SECTION:
shueISP.com.	10	IN	A	10.21.8.3

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 2150 msec
;; SERVER: 10.21.8.2
;; WHEN: Wed Nov 18 22:35:23 2020
;; MSG SIZE  rcvd: 56
; shueISP.com.	10	IN	A	10.21.8.3
Bad data; RR for name and type not found or failed to verify, and denial of existence failed.
```

Screenshot 8: Using DNSSec to detect that your communication was not secure



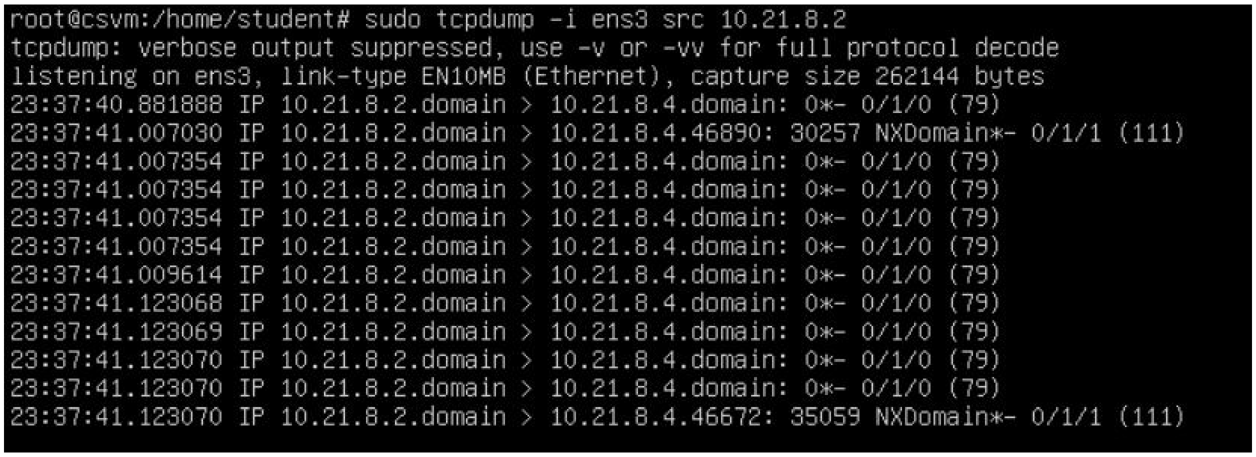
```
root@csvm:/home/student# sudo python3 reflection.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Screenshot 9. VM 3 sending requests impersonating VM 4



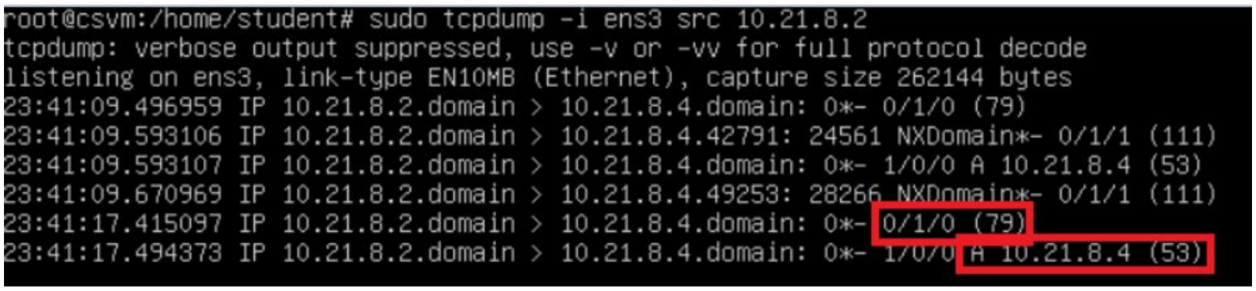
```
root@csvm:/home/student# sudo tcpdump -i ens3 src 10.21.84
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@csvm:/home/student# sudo tcpdump -i ens3 src 10.21.8.4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
23:22:19.863761 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:19.868981 IP 10.21.8.4 > csvm: ICMP 10.21.8.4 udp port domain unreachable, length 115
23:22:19.953671 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:19.954034 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:19.956397 IP 10.21.8.4 > csvm: ICMP 10.21.8.4 udp port domain unreachable, length 115
23:22:19.956397 IP 10.21.8.4 > csvm: ICMP 10.21.8.4 udp port domain unreachable, length 115
23:22:19.961657 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:19.969016 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:19.989733 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:19.997816 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:20.004451 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:20.011529 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:20.018670 IP 10.21.8.4.domain > csvm.domain: 0+ TXT? shueISP.bombast.com. (37)
23:22:20.095202 IP 10.21.8.4 > csvm: ICMP 10.21.8.4 udp port domain unreachable, length 115
23:22:20.095202 IP 10.21.8.4 > csvm: ICMP 10.21.8.4 udp port domain unreachable, length 115
23:22:20.095202 IP 10.21.8.4 > csvm: ICMP 10.21.8.4 udp port domain unreachable, length 115
```

Screenshot 10: DNS Server Packets show request coming from VM 4



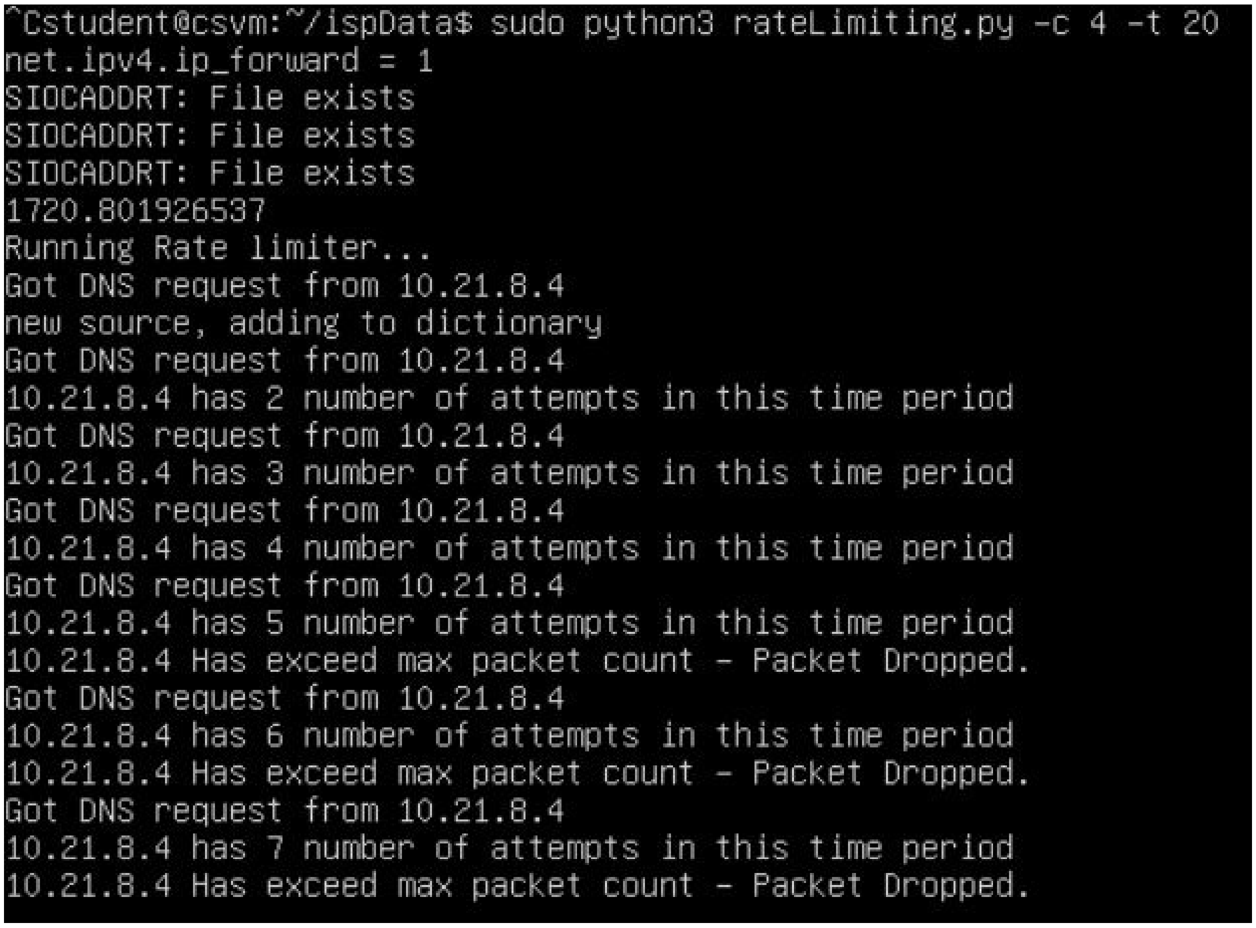
```
root@csvm:/home/student# sudo tcpdump -i ens3 src 10.21.8.2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
23:37:40.881888 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.007030 IP 10.21.8.2.domain > 10.21.8.4.46890: 30257 NXDomain*- 0/1/1 (111)
23:37:41.007354 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.007354 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.007354 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.007354 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.009614 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.123068 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.123069 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.123070 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.123070 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:37:41.123070 IP 10.21.8.2.domain > 10.21.8.4.46672: 35059 NXDomain*- 0/1/1 (111)
```

Screenshot 11: VM 4 receiving a bunch of DNS responses that they didn't request



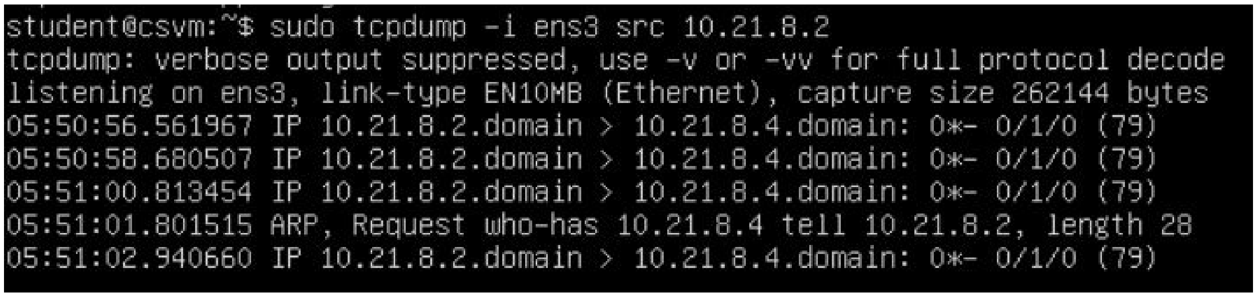
```
root@csvm:/home/student# sudo tcpdump -i ens3 src 10.21.8.2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
23:41:09.496959 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:41:09.593106 IP 10.21.8.2.domain > 10.21.8.4.42791: 24561 NXDomain*- 0/1/1 (111)
23:41:09.593107 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 1/0/0 A 10.21.8.4 (53)
23:41:09.670969 IP 10.21.8.2.domain > 10.21.8.4.49253: 28266 NXDomain*- 0/1/1 (111)
23:41:17.415097 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
23:41:17.494373 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 1/0/0 A 10.21.8.4 (53)
```

Screenshot 12: Size of a slightly amplified request versus a normal record request



```
^Cstudent@csvm:~/ispData$ sudo python3 rateLimiting.py -c 4 -t 20
net.ipv4.ip_forward = 1
SIOCADDRT: File exists
SIOCADDRT: File exists
SIOCADDRT: File exists
1720.801926537
Running Rate limiter...
Got DNS request from 10.21.8.4
new source, adding to dictionary
Got DNS request from 10.21.8.4
10.21.8.4 has 2 number of attempts in this time period
Got DNS request from 10.21.8.4
10.21.8.4 has 3 number of attempts in this time period
Got DNS request from 10.21.8.4
10.21.8.4 has 4 number of attempts in this time period
Got DNS request from 10.21.8.4
10.21.8.4 has 5 number of attempts in this time period
10.21.8.4 Has exceed max packet count - Packet Dropped.
Got DNS request from 10.21.8.4
10.21.8.4 has 6 number of attempts in this time period
10.21.8.4 Has exceed max packet count - Packet Dropped.
Got DNS request from 10.21.8.4
10.21.8.4 has 7 number of attempts in this time period
10.21.8.4 Has exceed max packet count - Packet Dropped.
```

Screenshot 13: Rate Limiting usage, each server only gets 4 requests for every 20 seconds



```
student@csvm:~$ sudo tcpdump -i ens3 src 10.21.8.2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
05:50:56.561967 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
05:50:58.680507 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
05:51:00.813454 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
05:51:01.801515 ARP, Request who-has 10.21.8.4 tell 10.21.8.2, length 28
05:51:02.940660 IP 10.21.8.2.domain > 10.21.8.4.domain: 0*- 0/1/0 (79)
```

Screenshot 14: Victim’s perspective, when attack is reduced by rate limiting