Introduction

The situation put forth in this assignment was an unethical CEO deciding that, while he has been blocked from tracking customers internet movements, they can “enhance” their marketing campaigns by diverting IP addresses to their own servers instead of the ones asked for by the client. This is a very real thing that can happen. If a malicious ISP decided that it wanted to increase traffic to it’s own websites, it can do one of a number of different attacks that accomplishes this goal. Routing attacks such as blackholing DNS responses is something, that when unprotected or unpredicted, could cause specific websites to not get the traffic they are expecting. This could lead to other issues as well. If the client does not realize what is happening, and only knows that they are getting rerouted to a different website than what is desired, the business model of their trusted Certificate Authority (CA) might be in jeopardy. They might no longer trust their CA to authorize certificates due to the fact that they are wrong. This project was broken down into four separate phases: the reconnaissance, the Infrastructure building phase, the attack phase and the defense phase. During the reconnaissance phase our group did preliminary research into the different attacks and defenses for common DNS exploits. Cache poisoning and DNS hijacking are two major problems that can be solved by DNSSEC as we read about in class. However, when looked at the stakeholders in the DNS process, we came up with our idea of blackhole attacks. Cache poisoning and DNS hijacking can be caused by another end user, blackhole attacks must come from the ISP. Blackhole attacks directly involved all of the stakeholders in DNS including a malicious ISP that might not necessarily be involved in other DNS attacks. In the infrastructure building phase we set up a small scale, yet realistic representation of DNS infrastructure that the client VM could make DNS requests from, through a router VM, to obtain the IP address of a web server. We chose this setup because it allowed us to accurately display a man in the middle style attack, by intercepting the response packet from the DNS and changing the destination IP of the web server to Bombasts’ IP. To implement the defense against this attack, we set up an encryption and decryption scheme to make sure the ‘man in the middle’ could not replace the destination IP in real time. The goal of this paper and the associated project was to explore the security goals and setup of DNS, through the successful implementation of an infrastructure, attack and defense.

Reconnaissance Phase

Domain Name System (DNS) provides three different security goals, integrity, authentication, and availability. While it is still possible to hack into DNS server, doing so would mean being able to hack a server owned by the Department of Defense of the United States. As you can imagine, that would be pretty hard. Therefore, the two more vulnerable goals are integrity and authentication. Within DNS we see a number of different stakeholders, entities that are affected when the security goals of DNS are not upheld. These stakeholders include end

users, web servers, root servers, top level servers, authoritative namespace servers, and internal Internet Service Provider (ISP) servers.

If we take a look at how each of these stakeholders would be affected if each of the security goals were not maintained, the easiest thing to identify would be the fact that none of them would be able to use the internet. For example, if a root server went down, all of the traffic would have to be routed through a different root server, which would slow everything down. This could create a chain reaction and bring that root server down until each one goes down. This, although very unlikely, would not be impossible. The more serious concerns come from the breach of integrity and authentication. If the goal of integrity is not met, then concerns such as DNS caching attacks could come into play. End users would have poisoned caches, so they would ask different servers which would give them fake phishing websites they might fall for, the web servers might not get the traffic they need, which would mess with their business model. However, if authentication is compromised, everything falls apart.

In DNS, we have to unconditionally trust the root servers. If someone successfully acts as the root servers, they then compromise the top level and authoritative namespace servers due to the fact that they trust root. If the root server is compromised, then no one will be able to trust any of the servers under that root that send the request to another server below it. The entire DNS model falls apart. Although there is a fix, such as resetting the server and certificates, it still takes a long time to get the chain of trust to revert back to the way it was before the compromise.

The countermeasures for the compromises in goals come in the form of DNSSEC. DNSSEC takes principals from the older Kerberos system. They use a predetermined key to talk to servers below it. The servers then can verify the person that referred them is who they say they are (the root server). DNSSEC uses asymmetric operations to ensure authenticity. For integrity, the DNS messages send an identification number along with the message which comes in the form of the ID number on the UDP packet. This reduces the chance of the message being compromised, an attacker would have to flood the computer with a large number of packets that brute force the UDP DNS request.

ISPs are in a unique vantage point to misuse DNS infrastructure. As all traffic for DNS queries go through ISPs they are able to track and change the integrity of those DNS queries. These violate the goals of the internet and directly affect the end users and web servers on the internet. The ISPs can take the DNS queries can if the DNS responds with an error the ISP can send the user to a web server, hosted by the ISP, that is filled with advertisements. The ISP can also commit domain hijacking. When a user has a DNS query for a competitive ISP the ISP can change the query to be for their own web server. This will cause the DNS to respond with the IP of the ISP. This causes the browser to cache the incorrect IP and forcing the user to go to the incorrect IP.

ISPs are not the only people able to attack users through DNS hijacking or domain parking. ISP do have a unique vantage point to easily commit domain hijacking attacks. To commit an attack similar to DNS hijacking without the vantage point, an adversary could easily flood the victim's computer with responses. If the attacker floods the user's computer with responses that match the results of a DNS query the computer will use that response instead of a response from the DNS. The only issue with this attack is the DNS ID nonce. If the attacker

cannot replicate a nonce that the computer is expecting, then the computer will not accept the response. This is not a large issue as the nonce is only a 16 bits and easy for computers to quickly go through.

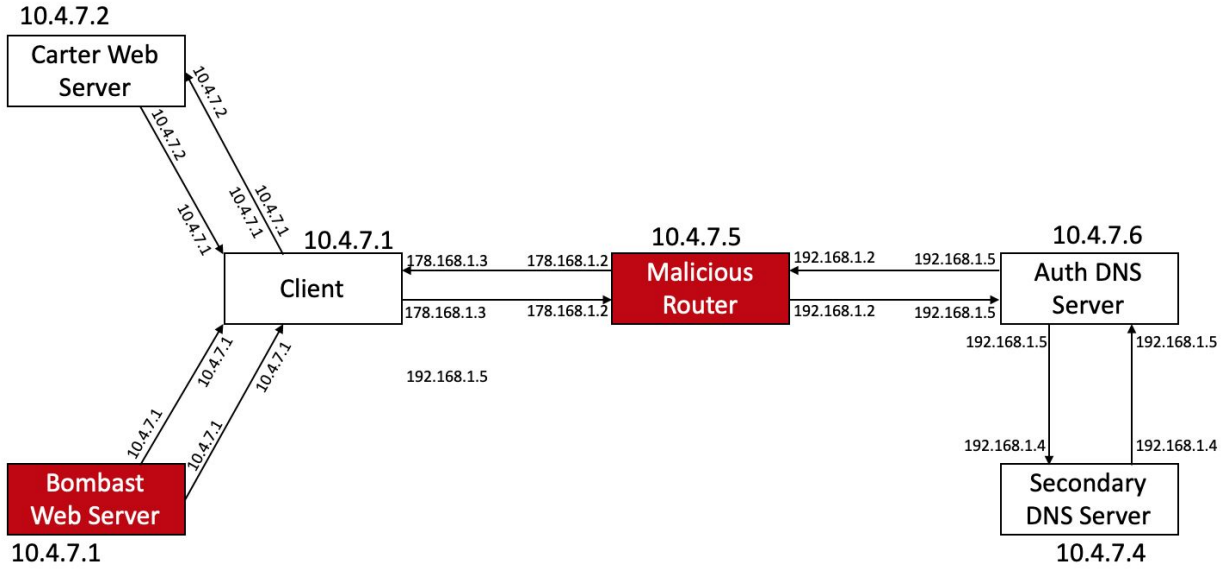
There are also two more attacks that can misuse DNS infrastructure. There is DNS amplification and similarly DNS reflection. DNS Amplification is when an attacker sends a request to the DNS servers that requires the DNS to respond with a much larger response. The attacker sends the request with the address of the victim. This causes the DNS to send the huge amount of records to the victim causing the victim or organization to completely shutdown from the size of responses (11, 12). A way the ISP handles DNS amplification attacks is blackholing the IP of the victim. This causes all traffic, legitimate or not, to not go to the victim which could be the goal of the attacker (12).

DNS amplification causes issues to two main goals in the greater internet. The internet relies heavily on authenticity and availability. Authenticity is the goal to make sure all internet traffic is authentic. Whether it is checking if the correct user is sending or receiving traffic or if the provider of the data is correct. Authenticity is broken as the request to the DNS is not authentic as it did not come from the actual user. The DNS has no way to determine if the request is a real request which causes DNS amplification attacks to be possible (11, 12). An attacker can pretend to be the victim and send a request to the DNS servers. The DNS server has no way to determine if the request is real and thus must send the responses. This can cause the victim's resources to be taken up and possibly the victim's IP to be blackholed. This also coincides with the other goal that is broken: availability. The goal in the general internet believes that all addresses should be available when they need to be. Whether it is the DNS, host, or web servers. This is broken by DNS amplification by causing a host, isp, or web server to be unavailable as the attack is occuring. (11, 12).

Infrastructure Building Phase

Our infrastructure models a client and DNS interacting with each other through a router, along with two web servers that the client connects to. The client is a virtual machine which requests for webpages can be made through. This was implemented through a simple python script using the *dnspython* resolver method. After the client VM makes a request it is sent, using ip routing rules, to the dns server via the router. The router is fairly simple, lacking any routing rules. It does however have a python script that can be used to test that packets are being sent through it by printing them out. The DNS infrastructure we setup using bind9, which is explained in appendix A, is composed of two VMs. A secondary server which the user communicates with and is used to complement a primary DNS server by serving a copy of the zone(s) configured on the primary server. The primary or authoritative DNS server uses bind9 to configure and hold the records for our registered domain name, www.bombast.com. When the DNS server responds to the client it is routed through the router back to the client, which then knows the IP of the webserver and makes a request for the web page from that VM. The two web server VMs are simple web pages with their name in an html file so that the program can be easily tested for

success in the attack and defense phases. In the figure below you can see a diagram of how this infrastructure is setup and how different VMs communicate. One aspect to notice is that we separated the client and DNS server into different subnets, so that they could not communicate without first going through the router.



Infrastructure Diagram

While working on this project, we also ran into problems with using “apt-get install” on VM 10.4.7.3, which we were originally intending to be the client. Because of this, we needed to move managing our client to 10.4.7.1, which doubles as one of our web servers. While we are using the same VM for them, we are pretending that they are two separate entities and do not affect each others functionality. If we had another functional VM, we would easily separate them and it would still work.

Attack Phase

For our attack we decided to do a Man-in-The-Middle attack. The man in the middle is the ISP, Bombast. As all traffic goes through the ISP, Bombast can queue all of the packets going through it and modify them as necessary. To achieve this we use NFQUEUE and Scapy as python packages. We routed all traffic going through port 53 on the gateway vm to be put into the queue created by NFQUEUE. This allowed us to drop every packet, a blackhole attack, and using scapy we are able to sniff the packet before it is dropped. This allows us to recreate the packet and if it is deemed by the script replace the rdata in the dns packet with the ip for Bombast. The script determined this by looking at the qname in the dns query. If the qname was ‘www.carter.com’ then the ISP would replace the rdata in the ‘an’, ‘ns’, and ‘ar’ sections of the dns packet with the IPv4 of ‘10.4.7.1’. This allowed the client to still see the url as ‘www.carter.com’ while still receiving the incorrect IP address.

To start testing, first start all relevant programs:
Client VM (10.4.7.1):
python test.py www.carter.com (do last)
python encryption_attack.py (results will appear here)
Router VM (10.4.7.5):
python router.py
Web Server VM (10.4.7.2):
python encrypt.py
python3 server.py
DNS Server (10.4.7.6):
python ecrypt_attack.py

In appendix B you can see the effectiveness of this test by observing the packets passing through the router and the bombast web page appearing instead of the carter webpage that was requested.

Defense Phase

For our defense we used RSA, asymmetric encryption. RSA is used in a lot of transmissions including SSH. We made our own function to generate private and public keys in Python. We are assuming that the client's public key is known to the DNS server and that the DNS server's public key is known to the client. To protect against the blackhole attack and the rewriting of IP addresses from the DNS packets from the router, we will encrypt the essential parts of the packets and doing a bit of routing manipulation in order to confuse the router. The client first sends a message to the DNS server with an encrypted qname. At this point we are assuming that the malicious ISP controls all of the routers out of the DNS server. We did not reflect this due to lack of VMs. Therefore, we assume that there would be a router that goes between the real web server and the DNS server. Our DNS server sends a signal to the real web server that the client would want to reach due to the fact that it is waiting for the ISP to black hole the next outgoing packet by the DNS server when it sees an encrypted packet. What is really happening is the decryption on the side of the DNS server with its own private key. It then can encrypt the response and send it back to the client which can use their private key to read the response. We are assuming that the ISP (router on VM 5) now has their blackhole shut off due to the fact that it is waiting for traffic that is not encrypted to run its attack again. The packet containing the IP resolution would be able to be sent over the network without the black hole in place and catch the ISP off guard.

We ran into a bit of trouble along the way which ended up making our defense stronger than we anticipated. As it turns out, DNS packets are only a specific size. We generated public keys of 2048 bytes. We ran into errors that our DNS response packet was too long. Our one downside, which makes the defense a bit insecure, is the fact that we had to generate a 128 byte key to be able to make it work. Although this could be brute forced, it would take a few days to brute

force, making it rather useless to DNS attacks that takes less than a second to carry out. On top of the bouncing, we also adjusted the areas of the packet that we hid the response. We were able to manipulate the resolve of the qname. In the 'rrname' part of the DNS response packet, there is the RSA encrypted version of the response IP address. This is decrypted by the client. The fake IP resolve from the DNS we set to be 1.1.1.1, however this can be set to any other IP address that DNS wishes to provide. We learned about RSA encryption as well as how to restructure DNS packets and hide information in places that would not be expected. This helped us get a jump on mission 3. Although it is not standard to hide information this way, we decided to create a more unique defense than DNSSEC. DNSSEC would not be the best defense for our attack due to the fact that the IP resolve would still be out in the open and the ISP could see where it is that the client was connecting to. Having these types of signed, hidden, and redirected messages applies a defense that is more unique to our attack.

To start testing, first start all relevant programs:
Client VM (10.4.7.1):
python test.py www.carter.com (do last)
python encryption.py (results will appear here)
Router VM (10.4.7.5):
python router.py
Web Server VM (10.4.7.2):
python encrypt.py
python3 server.py
DNS Server (10.4.7.6):
python encrpt.py

In appendix B you can see the effectiveness of this test by observing the packets passing through the router and the carter web page appearing instead of the bombast page.

Conclusion

As this exploration has demonstrated, DNS infrastructure is only as reliable as its defense makes it, and if it does not have sufficient protection then there are many who would benefit from its exploitation. In our example, the DNS response was exploited because of the weakness in how the packet was sent unencrypted. This allowed for the malicious ISP to see the contents of the packets being sent over the network and can read the desired location of the victim.

As the ISP was essentially a Man-in-The-Middle opponent. Since the ISP can essentially see every packet that is going through the network it can sniff and capture the packets. This allowed it to act as a malicious router that can change the packets going through it. This created

an insecurity. We resolved this insecurity by creating an encrypted packet that allows the router to be unable to change the IP address. This solution is a very realistic solution. OS's now have keys that can be used for CA authorities so it would not be out of the realm to also require OS's to require new keys for DNS to verify.

Appendix A

Client Setup

The first virtual machine we setup is the Client VM. First, to set this up we added extra IP addresses to interface eth0. Next, we also added some ip routing rules for outgoing packets. Lastly, to assist in defense we had to add certain iptable rules.

```
# ssh cs4404@10.4.3.1
#
# sudo su
# ifconfig eth0:0 178.168.1.3 netmask  255.255.255.0
# route add default gw 178.168.1.2
# ip route add 192.168.1.5 via 178.168.1.3
```

In order to enable encryption (in the defense) we also needed to include some redirections so that we could encrypt/decrypt files:

```
# iptables -A OUTPUT -p udp -m udp --sport 53 -j NFQUEUE --queue-num 0
# iptables -A OUTPUT -p udp -m udp --dport 53 -j NFQUEUE --queue-num 0
# iptables -A INPUT -p udp -m udp --sport 2000 -j NFQUEUE --queue-num 0
# iptables -A INPUT -p udp -m udp --dport 2000 -j NFQUEUE --queue-num 0
# iptables -A INPUT -p udp -m udp --dport 5300 -j NFQUEUE --queue-num 0
```

Router Setup

The next step is setting up the Router VM. The only setup required for the router is to add extra IP addresses to interface eth0.

```
# ssh cs4404@10.4.3.5
#
# sudo su
# ifconfig eth0:0 192.168.1.2 netmask  255.255.255.0
# ifconfig eth0:1 178.168.1.2 netmask  255.255.255.0
```

In order to enable encryption (in the defense) we also needed to include some redirections so that we could encrypt/decrypt files:
iptables -A FORWARD -p udp -m udp --sport 53 -j NFQUEUE --queue-num 0
iptables -A OUTPUT -p udp -m udp --dport 53 -j NFQUEUE --queue-num 0

DNS Secondary Server Setup

The next step is setting up the DNS secondary server. SSH into the server. First, to set this up we used an online tutorial (14). Second we added another IP address to the interface, so that there would be network separation between the server and client. Third, we added some IP routing rules so that packets for the client were sent through the router. Lastly, we added iptable rules to reject packets from the client attempting to connect.

ssh cs4404@10.4.3.6
#
sudo su
apt-get install bind9 bind9utils bind9-doc
ifconfig eth0:0 192.168.1.5 netmask 255.255.255.0
route add default gw 192.168.1.2
ip route add 178.168.1.1 via 192.168.1.2
-- we setup using the online tutorial --
sudo service bind9 restart

The files we set up with the online tutorial were and can be found in our file submission:
    /etc/bind/named.conf.local
    /etc/bind/named.conf.options

In order to enable encryption (in the defense) we also needed to include some redirections so that we could encrypt/decrypt files:
iptables -A INPUT -p udp -m udp --dport 5300 -j NFQUEUE --queue-num 0
iptables -A OUTPUT -p udp -m udp --sport 53 -j NFQUEUE --queue-num 0
iptables -A OUTPUT -p udp -m udp --dport 2000 -j NFQUEUE --queue-num 0

DNS Auth Server Setup

The next step is setting up the DNS Auth server. SSH into the server. First, to set this up we installed bind9 then used an online tutorial. Second we added another IP address to the interface, so that there would be network separation between the server and client. Third, we added some IP routing rules so that packets for the client were sent through the router. Lastly, we added iptables rules to help with our defense.

ssh cs4404@10.4.3.4
#
sudo su
apt-get install bind9 bind9utils bind9-doc
ifconfig eth0:0 192.168.1.4 netmask 255.255.255.0
route add default gw 192.168.1.2
ip route add 178.168.1.1 via 192.168.1.2
-- setup using the online tutorial (14) --
sudo service bind9 restart
iptables -A INPUT -s 10.4.7.3/32 -j DROP
iptables-A OUTPUT -s 10.4.7.3/32 -j DROP

The files we set up with the online tutorial were and can be found in our file submission (14):

/etc/bind/named.conf.local
/etc/bind/named.conf.options
/etc/bind/zones/db.www.bombast.com
/etc/bind/zones/db.www.carter.com
/etc/bind/zones/db.192.168.1

Web Server 1 Setup

ssh cs4404@10.4.3.1
cd mission2/server
python3 server.py

Web Server 2 Setup

ssh cs4404@10.4.3.2
cd mission2
python3 server.py

Off Site DNS Bounce Setup

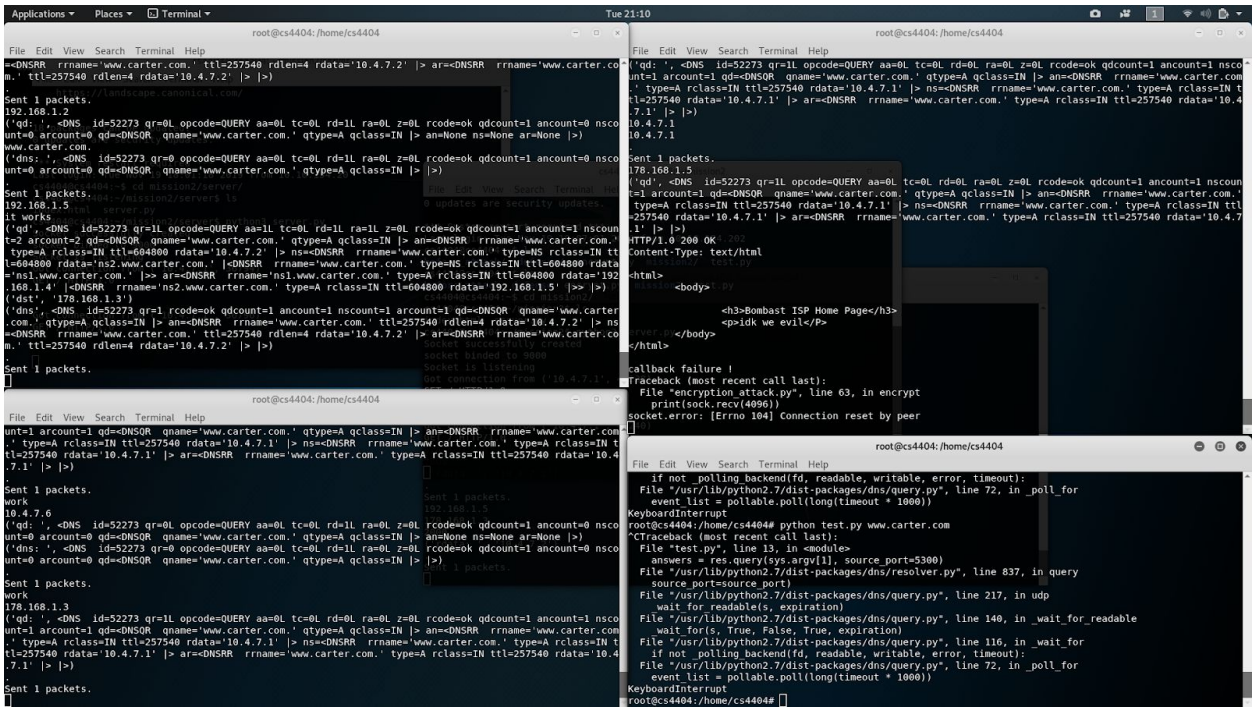
ssh cs4404@10.4.3.2
python encryption.py

ifconfig eth0:0 192.168.1.3 netmask 255.255.255.0
ifconfig eth0:1 178.168.1.3 netmask 255.255.255.0
iptables -A INPUT -p udp -m udp --dport 2000 -j NFQUEUE --queue-num 0
iptables -A INPUT -p udp -m udp --dport 53 -j NFQUEUE --queue-num 0

Appendix B

Attack:

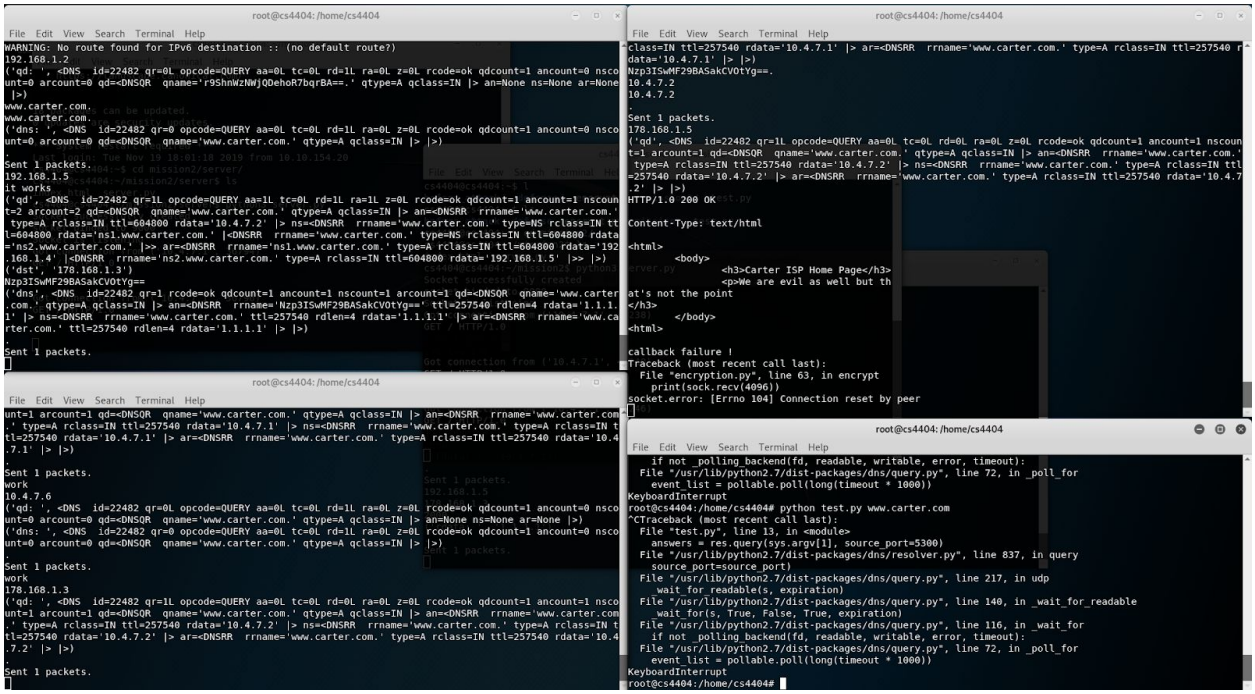
In the client image you can see the rname is www.carter.com (what was requested by client), but the ip and what is printed is the Bombast webpage.



(top left is the DNS, top right is the client)

Defense:

In the client image you can see the rname is www.carter.com (what was requested by client), and the printed web page is for Carter as well. You can also see the encrypted IP at the top of the client terminal.



(top left is the DNS, top right is the client)

-

Citations: APA format

1. Davidowicz, D. (n.d.). Retrieved from http://compsec101.antibozo.net/papers/dnssec/dnssec.html.
2. Admin. (2014, January 19). How does a DNS query works when you type a URL on your browser? Retrieved from https://www.golinuxhub.com/2014/01/how-does-dns-query-works-when-you-type.html.
3. Cloudflare. (2019) *DNS Security | Cloudflare*. Cloudflare Inc., https://www.cloudflare.com/learning/dns/dns-security/. Accessed Nov 13, 2019.
4. Syngress. (n.d.). DNS Security: Defending the Domain Name System. Retrieved from https://searchsecurity.techtarget.com/feature/DNS-Security-Defending-the-Domain-Name-System.
5. Team, S. T. (2018, November 22). SecurityTrails: The Most Popular Types of DNS Attacks. Retrieved from https://securitytrails.com/blog/most-popular-types-dns-attacks.
6. Jan, R. H. (n.d.). How to protect data and infrastructure with DNS security. Retrieved from https://gcn.com/articles/2016/01/14/dns-security.aspx.
7. Anderson, N. (2009, August 5). Comcast adopts DNS hijacking, imposes irritating opt-out. Retrieved from https://arstechnica.com/tech-policy/2009/08/comcasts-dns-redirect-service-goes-nationwide/.
8. Taylor, D. (2011, June 28). How can I stop Comcast hijacking my mistyped domain addresses? Retrieved from https://www.askdavetaylor.com/stop_comcast_hijacking_mistyped_domain_addresses_dns/.
9. Kravets, D. (2017, June 4). Comcast.net Hijacked, Redirected. Retrieved from https://www.wired.com/2008/05/comcast-servers/.
10. Poulsen, K. (2017, June 4). Comcast Hijackers Say They Warned the Company First. Retrieved from https://www.wired.com/2008/05/comcast-hijacke/.
11. DNS-Based Threats: DNS Reflection and Amplification Attacks. (2017, December 13). Retrieved from https://blog.verisign.com/security/dns-based-threats-dns-reflection-amplification-attacks/.
12. DNS Amplification Attacks: CISA. (n.d.). Retrieved from https://www.us-cert.gov/ncas/alerts/TA13-088A.
13. Cloudflare. (2019) *DNS Security | Cloudflare*. Cloudflare Inc., https://www.cloudflare.com/learning/ddos/dns-amplification-ddos-attack/ Accessed Nov 13, 2019.

14. DigitalOcean. (2019, September 18). How To Configure BIND as a Private Network DNS Server on Ubuntu 14.04. Retrieved from https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04.