CS 4404: Tools and Techniques in Network Security

Team: John Faria,Maryia Hancharova, Shuxing Li, Joseph Hogan

Mission 2: DNS (in)Security

# 1. Introduction

For this mission, our team implemented the CEO’s vision of giving the users a “helping hand”. We simulated an infrastructure and an attack that redirects the client, who is trying to reach a Berizon web page to the IP address of Bombast, that Bombast as a malicious ISP wanted to send the client to. Later, we followed the instructions of the supervisor and explored a two step defense that organizations could use to implement a defense against Bombast, using a VPN and a tool similar to OONI to detect and avoid redirection. In the following report we will explore DNS, it’s infrastructure, as well as weaknesses and demonstrate how organizations can protect themselves from similar types of redirection with widely available tools.

# 2. Reconnaissance Phase

## 2.1 DNS Security Goals

In order to achieve our mission we researched DNS, it’s security goals, weaknesses and countermeasures that organizations can implement to protect themselves from common attacks.

The primary goals of DNS are origin authentication, integrity protection and availability. When it comes to DNS, confidentiality is not one of the goals that was built into the infrastructure because DNS data is supposed to be public [2]. If the communication needs to be private, IPSEC is generally used as a method with DNS for ensuring privacy.

The internet heavily relies on DNS servers to resolve IP addresses to domain names. Without reliable DNS servers, the availability and reliability of the internet takes a hit. Without working and available DNS servers, the internet might experience widespread outages and the attackers have more opportunity to redirect users to malicious website.

While there are inherent DNS goals, end-users and organizations also have a set of their own goals. For end users the primary goal is availability. The user expects to receive the IP address of the website they requested, they expect always available DNS servers that will promptly reply to their query. Integrity and authenticity are also of course important for the users. The information users expect to receive should be accurate and come from a trusted source. However, individual end users, especially those without a background in technology might be less aware of those goals and expect the information to be accurate all the time. Therefore integrity and authenticity are also crucial for end users, since they do not ever want to be redirected to a malicious website and have their data compromised, however they might not think about it as often as the goal of availability.

For organizations the primary goals are the same as the overall goals of the DNS infrastructure. Organizations care about availability, integrity and authenticity. Organizations expect their servers to be reliable and available to make sure the users are not redirected to another DNS server that might be poisoned. They also care a great deal about the integrity of the information. No organization wants the user attempting to reach their webpage to be redirected to a web page of their competitor. If the user is redirected to another competitor’s website, the organization might lose a portion of their profits if they are trying to sell a product. If the user is redirected a malicious website, posing to be a valid site, user’s data might be compromised and the user might end up blaming the company when their credit card information is misused. Therefore, organizations must dedicate a significant amount of effort in order to make sure they meet all three of the DNS goals.

## 2.2 DNS Vulnerabilities

While DNS has several security goals, those goals can easily become compromised due to DNS vulnerabilities that exist in the infrastructure. DNS was built with the goal to to accurately translate domain names to IP addresses required by the machines, servers, routers and other network devices. To get an IP address, the browser uses a stub resolves which forwards the request to a recursive resolver. The recursive resolver finds the answer to the DNS query by sending its own queries, usually to various authoritative name servers [1]. However, when DNS was built, security was not at the forefront of the concerns. When the recursive resolver sends a query to the authoritative server, the resolve in the basic DNS doesn’t have a way to verify the authenticity of the response. DNS can check that the response comes from the IP address the query was originally sent to, however this is not a good check since IP address in a DNS packet can be easily spoofed [1]. Therefore, attacker can pretend to be an authoritative server and redirect a user to a malicious website.

In terms of attack vector for origin authentication, one vulnerability of DNS protocol is that it does not authenticate the responses to recursive queries [8]. The only checks that DNS performs is that “the query section and 16-bit transaction ID of the response must match those of the query” and that “the source IP address and destination port of the response must match, respectively, the destination IP address and source port of the query” [8]. Therefore, the first UDP packet that satisfies these conditions will be counted as a valid response allowing DNS forgery attack to succeed. Furthermore, until later patches that were made, many DNS resolvers used a fixed port for sending queries making all the values used for the check predictable [8]. In 2008, Kaminsky presented a similar type of attack that exploits this vulnerability. The attacker can choose which domain he wants to compromise, choose a subdomain and query the resolver with any subdomain that does have a stored cache copy. Then the attacker sends a large number of forged responses to the resolver with different transaction ID. This type of attack is extremely effective and more dangerous, since if it is successful, it introduces a false mapping into the cache [8].

In terms of compromising the integrity of the data, attackers can use a variety of techniques such as cache poisoning to allow substitution of inaccurate information into DNS records. Recursive resolvers rely on cache to speed up the process of resolution, so they cache data they receive from authoritative servers so the resolver reduces the delay in the response to the query [1]. However, using cache has a downside, “if an attacker sends a forged DNS response that is accepted by a recursive resolver, the attacker has poisoned the cache of the recursive resolver. The resolver will then proceed to return the fraudulent DNS data to other devices that query for it.” [1]

Finally, attackers can also compromise the availability of the DNS server with a Distributed Denial of Service (DDoS) attack. In DDos attack the attacker overwhelms the server with too much traffic generally by using a series of computer systems that act as sources for the traffic. In a typical DDos attack, attacker can gain control of a network resources such as computers or other machines that have been infected with malware turning them into bots. Then the attacker can take remote control over that group of bots, forming what’s called a botnet. Once the botnet is established, the attacker can direct all the machines to send requests to the target DNS server, causing overload.

## 2.3 ISPs

If we are thinking about the resources that an attacker might have, ISPs have an advantage point that typical attackers will not have. Furthermore, ISP’s have practiced disturbing mechanisms to spy on their clients on multiple occasions in the past.

When a user sends out a query for accessing a website, ISP has an ability to view the packet request being sent given that the information is not encrypted because the user has to go through their ISP in order to access the internet. Therefore, ISP’s can generally gain a lot of information about the user and some ISP’s might choose to misuse that data. ISP’s can choose to sell consumer’s data such as “whether shoppers are checking out competitor prices” or “the age ranges and genders of the people who visited a store location between between 10 a.m and noon” [9].

While ISP’s can passively collect data, they also actively hijack user’s searches and in fact, in 2011 “several ISPs were caught red-handed working with a company called Paxfire to hijack their customer’s search queries to Bing, Yahoo! and Google” [9]. The type of hijacking that the ISP performed is similar to the type of attack we will explore in this report. In the attack, when the user entered a search word in a browser, the ISP would direct that search to Paxfire instead of the search engine [9]. If the term the user searched for matched a company that has paid for traffic to be directed to their website, Paxfire would send the user to the company’s website instead of giving the full list of search results.

If ISP’s are not actively redirecting the user, they can also snoop through the user’s traffic and insert additional information such as ads or injecting tracking cookies that will record all of the user’s searches. AT&T, Charter, CMA have all at one point injected ads into user’s traffic based on user’s browsing history [9]. ISP’s have the access to examine the packets user’s send out and building a profile that they can then use to create targeted ads or hire a company such as NebuAd or Phorm to do it for them [9]. AT&T and Verizon have also tracked all of their user’s traffic through inserting cookies. Specifically, in 2014 Verizon inserted cookies into all mobile traffic of their users and to make matters worse did not inform their customers and did not give them an option to turn off the feature [9].

ISP’s have an incredibly favorable position for tracking their user’s and making additional profit from the data. A typical attacker does not have access to the customer’s profile and cannot maintain constant surveillance without additional efforts. Customer outrage and FCC resulted in stopping and prevention of these tracking practices, however ISP’s still continue the practice today and continue profiting off their client’s data.

## 2.4 Exploiting DNS infrastructure

We also explored common methods for exploiting DNS infrastructure and the ramifications. Attackers can exploit DNS infrastructure by employing amplification and reflection attacks.

DNS amplification is frequently used by attackers during DDoS attacks. “In mid-March 2013, attackers used DNS amplification to launch a high-profile attack against Spamhaus, with attack traffic volume exceeding 300Gbps” [3]. DNS amplification attackers are especially favored by attackers because amplification allows attackers to create large amounts of traffic sent to the victim and conceal their identities by using IP address spoofing, limiting chances that they will be blacklisted [3].

Standard DNS amplification consists of an attacker sending DNS query packet to a DNS server. When an attacker creates the packet, he can replace source IP address field with the victim’s IP. Therefore, the DNS server after receiving the packet, will send a response back to the IP address indicated in the field, aka the victim hence the reflection part of the attack. The victim will receive the packet and disregard the packet after realizing that no request was actually sent from the victim for that particular packet. However, while the victim was processing that respond they did not request, part of their computational resources and bandwidth was used. And since “DNS response packet from the DNS server is larger than the query packet the attacker sent, the attack traffic at the target is increased by a certain amplification factor” hence this attack is called amplification [3]. When these attacks are used over a distributed system, they generate extremely large amounts of traffic and can cause server outage due to overload.

## 2.5 Countermeasures

DNSSEC is the security countermeasures that have been developed to address the security issues with DNS. DNSSEC helps with origin authentication and integrity security goals by encrypting the content of the DNS data that comes from the owner of the data. The DNS resolver can take the public key of the host and resolve the data that the query has returned. If the public key of the host resolves the data then the origin and integrity can be confirmed. However, if the returned data is not resolved by the Hosts public key, the resolver drops the data because it came from an invalid source. DNSSEC thus resolves the problem of an attacker sending falsified DNS responses to the resolver [1].

Another method for DNS verification is, in a sense, crowdsourcing. Each user of the internet are not guaranteed to have the same view. For example, some countries have heavily censored internet access. DNS servers that have been attacked or intentionally manipulated may contain different responses. By comparing all of the different responses across the world, users and organizations can identify which DNS servers are acting maliciously and which DNS servers appear to be working properly. The OONI Project [6] is a free software project under The Tor Project that provides both scanning software and records of scans for DNS consistency and other forms of internet censorship.

# 3. Infrastructure Building Phase

In order to demonstrate our attack and defense, we organized our infrastructure using the VM’s in the following way:

|  |  |
| --- | --- |
| 10.4.20.1 | Web Client that wants to reach a webpage on the web |
| 10.4.20.2 | Man in the Middle created by the ISP that monitor’s all the client’s traffic and re-directs it to the DNS belonging to Bombast |
| 10.4.20.3 | DNS for Bombast |
| 10.4.20.4 | Bombastteam20.com website that Bombast DNS will redirect users to |
| 10.4.20.5 | Normal DNS server, another DNS service that redirects the user to the webpage they were looking for |
| 10.4.20.6 | Berizonteam20.com that the client is trying to reach |

## 3.1 Client Setup

In order to simulate a client (10.4.20.1) connecting through the ISP we routed all of the network traffic from the client to go through the ISP’s meddler-in-the-middle (10.4.20.2). In order to redirect the network traffic we set rules on the client routing table using the following command: *“sudo route add -net 10.0.0.0/10 gw 10.4.20.2”*. The “route add” command creates a rule on the routing table that’s more specific than the default 10.0.0.0/8 route. The new rule directs all traffic on the client to be passed to our meddler-in-the-middle at 10.4.20.2. In order to prevent ICMP messages from interfering with the routing the command *“sudo net.ipv4.conf.all.accept\_redirects = 0”* was executed, preventing the client from accepting any ICMP redirects.

## 3.2 Meddler-in-the-Middle Setup

After the network traffic from the client (10.4.20.1) is redirected to the meddler-in-the-middle (10.4.20.2) the traffic should then be forwarded through the meddler-in-the-middle. We enabled IP forwarding on the meddler-in-the-middle using the command *“sudo sysctl -w net.ipv4.ip\_forward=1”*. After network traffic was forwarded to the meddler-in-the-middle and IP forwarding was enabled, the client was able to use the meddler-in-the-middle functionally as a router.

## 3.3 Bombast DNS Setup

First, we created our DNS server for Bombast.com using BIND configuration. We cleaned and reinstalled bind to make sure we were working with a clean set of settings. Note that if you are working with a clean install of Ubuntu this might not be necessary. The command we used is *“sudo apt-get purge bind9”* to purge and clean all the settings we were not using, and *“sudo apt-get install bind9”* command to install a new bind package. In addition we also installed utilities for bind9 by using *“sudo apt-get install bind9utils”* command.

Next we proceeded to create zones for our domain. Note that in the set up of our domain, we set up bombastteam20.com as our domain instead of bombast.com to make sure that we were not conflicting with the settings of any other group.

We navigated to the bind directory by using “*cd /etc/bind”* command and creates a new “zones” directory by using *mkdir zones* command. Then we created a new zone file for our domain by using “*nano db.bombastteam20.com command”*and opening it with a nano text editor. Then, we followed the instructions given in the “Linux DNS server BIND configuration” article [4]for the set up of the BIND file (see Figure 1 for detailed instructions) and modified all the fields containing linuxconfig.org with bombastteam20.com and all the 192.168.0.10 IP addresses with our 10.4.20.3 IP address, which is the address of one of our virtual machines that we designated to be our DNS server for ISP (see Figure 2 for our final BIND set up).

We also created another file using command “*nano db.berizonteam20.com”* in the same directory for the Normal DNS server and replicated the steps from the above except with the DNS server designated for Berizon hosted on 10.4.20.5 so when a client tries to access Berizon, they can access it’s webpage.

Next, we inserted the zone file name into a bind configuration file titled named.conf.local by navigating to bind directory and using the command “*nano bind/ nano.conf.local”* to open the file in the nano text editor. We again followed the instructions from the “Linux DNS server BIND configuration” article [4] (see Figure 3 for the suggested set up given in the article) and replaced the appropriate fields to match our configuration (see Figure 4 for our final named.conf.local configuration).

In this portion of the set up, Bombast DNS redirects the user to the webpage they were actually attempting to reach, not their own webpage that Bombast will redirect the user to. In the attack portion we made modifications to the db.berizonteam20.com file to ensure that Bombast DNS acts as a malicious server and will redirect the user to the Bombast site, instead of Berizon. At this point, we tested the validity of our set up by using the command */etc/init.d/bind9 start* for starting our bind9 DNS server and used the command “*dig @10.4.20.3* [*www.bombastteam20.com*](http://www.bombastteam20.com/)*”* and *“dig @10.4.20.5* [*www.berizonteam20.com*](http://www.berizonteam20.com/)*”* to confirm the working configuration.

## 3.4 Bombast Website

For the Bombast website, we simulated a webpage for Bombast on the virtual machine with the IP address 10.4.20.4. We created a script webserver.py that exchanges HTML code for the Bombast website with the client when it is requested (See Figure 9 for the code responsible for the web page). The web server was tested by running the command *“wget* [*www.bombastteam20.com*](http://www.bombastteam20.com/)*”.* The server script opens a socket at port 80, and respond with a piece of HTML code to any client that attempts to connect to it. The message responded was hardcoded in the script, so we had a similar copy that has a different response on the Berizon server mentioned below as well. To run the server, simply run the command *“sudo python webserver.py”*, and the server will keep on running until the process is terminated.

## 3.5 Normal DNS

We also set up a DNS server that will not falsely redirect clients hosted on the 10.4.20.5 machine using similar instructions to section 3.3.

We set up berizonteam20.com as the domain and created zones directory in the cd /etc/bind. We created two zone files by using *nano db.bombastteam20.com* and *nano db.berizonteam20.com*. We followed the instructions for the set up of those files similar to Section 3.3 and modified the appropriate fields in two of our “db” files (see Figure 6 and 7). We also inserted our zone files into a bind configuration file located in bind directory by using *nano bind/ nano named.conf.local* and pointed to our own configuration file (see Figure 8). At this stage of the implementation, both Bombast DNS server and Normal DNS act normally and redirect the user to where they are trying to go. However, the attack section this configuration is modified and the client is always directed to the malicious website set up by Bombast.

## 3.6 Berizon Website

For the Berizon website that the client truly wants to reach, we simulated web page on 10.4.20.6 machine. We created a webserver.py script similar to the one mentioned in section 3.4 that exchanges HTML code for the Berizon website with the client when it is successfully reached (See Figure 10 for the code responsible for the web page). The script functions exactly like the script mentioned in section 3.4. Web server was tested by running the command *“wget* [*www.berizonteam20.com*](http://www.bombastteam20.com/)*”.* The server script opens a socket at port 80, and respond with a piece of HTML code to any client that attempts to connect to it. To run the server, simply run the command *“sudo python webserver.py”*, and the server will keep on running until the process is terminated.

# 4. Attack Phase

For our attack we implemented CEO’s vision of altering DNS response without the knowledge of the client. In our implementation, the Bombast ISP intercepts and reroutes all of the client’s DNS queries to Bombast’s own DNS server. The Bombast DNS server incorrectly translates a domain query for the Berizon website and intentionally returns the IP address for Bombast’s website instead.

In Section 3.3 we demonstrated the set up of the infrastructure for the Bombast DNS server. In this section we will demonstrate the set up of the Man in the Middle that spies on all the client traffic and forwards it all to Bombast DNS, which proceeds to redirect the client to a malicious website.

## 4.1 Setting up Man in the Middle IP Routing Tables

As described in our infrastructure phases 3.1 and 3.2, we set up a custom router for our client (10.4.20.1) on the meddler-in-the-middle (10.4.20.2). For the attack we installed iptables and set up a set of firewall rules on the meddler-in-the-middle following some instructions from “How to redirect an incoming connection to a different IP address on a specific port using IP tables” [5]. Iptables was installed using the command *“sudo apt-get install iptables”*. An iptables firewall rule to redirect all DNS traffic being carried by the meddler-in-the-middle to Bombast’s DNS server was added using the command *“sudo iptables -t nat -A PREROUTING -p udp --dport 53 -j DNAT --to-destination 10.4.20.3:53”*. Creating this firewall rule forces every DNS query from the client, which is routed through the meddler-in-the-middle, to be redirected to Bombast’s DNS server, but the response will be rejected because of the mismatch in the server queried and the server responded. In order to mask where the response came from we added a firewall rule to the meddler-in-the-middle by executing the command *“sudo iptables -t nat -A POSTROUTING -j MASQUERADE”*. Through this process, the client will be told that the response from any DNS request it makes is coming from wherever it requested even though it actually came from the Bombast DNS server. The client will consider the redirected response to be legitimate and accept it.

## 4.2 Editing Bombast DNS

For the attack phase, we have also made modifications to Bombast DNS server. In the infrastructure portion section 3.3 we set up a normally functioning Bombast server that directs the client to the actual webpage they were trying to reach. Hence if the user was trying to reach Berizon, originally Bombast would allow them to access a site of the competitor. However, in this portion we have modified db.bombastteam20.com configuration file by navigation to bind directory using *“cd /etc/bind”* command and *“nano db.bombastteam20.com”* command and modified the IP address of berizon.com so the DNS server returns false domain name record. When the webserver is running on 10.4.20.4 and a wget request is made from the client for either [www.bombastteam20.com](http://www.bombast.com/) or [www.berizonteam20.com](http://www.berizonteam20.com/), the user will client will always receive the webpage for Bombast, even if the client specifically tries to access a different DNS. The success of the attack can be seen in figure 16 with the wrong webpage being given, and in figure 12 with the failure of OOONI to get the correct IP address for Berizon.

# 5. Defense Phase

For the defense portion of this report we explored a two step defense. First we wrote a tool that detects network redirection through DNS consistency similar to OONI [6]. Second, we created a virtual private network (VPN) to avoid Bombast redirection. In section 5.1 we demonstrate the implementation of the VPN on our virtual machines and in section 5.2 we describe the setup of the small scale script that functions similar to OONI.

## 5.1 OOONI (Our Open Observatory of Network Interference)

The Open Observatory of Network Interference (OONI) is a project that aims to detect and report internet censorship. OONI Probe is an application that end users can deploy to scan their internet connections for the blocking of websites and tools for avoiding surveillance such as Tor, and to detect any other types of systems that might be responsible for censorship [6]. One of the tests conducted by OONI is the DNS consistency check. In this check OONI will compare the DNS query results from a user’s DNS resolver against the IP addresses for those domains that OONI already knows and considers to be trusted. If OONI detects that the returned IP address from its DNS query is different from the IP address that it believes is correct, OONI notifies the user that someone may be tampering with their DNS requests and provides recommendations for how the user can protect themselves.

As the first part of our defense for this mission we implemented a smaller version of OONI with a “oooni.py” script (see “oooni.py” script in the 10.4.20.1 folder from the archive). In our version of OOONI (Our Open Observatory Network Interface) we wrote a python script which will conduct a DNS consistency check and notify the user if their DNS requests are being tampered. When OOONI is run on the client machine (10.4.20.1) during the attack it will alert the user that the DNS record for berizon.com is being tampered with (see Figure 12 for the output of oooni.py). After the VPN has been set up and the client’s traffic is being safely routed through the meddler-in-the-middle (10.4.20.2) without being redirected, OOONI will notify the user that none of the DNS records that it checked appear to have been tampered with (see Figure 13 for the output of oooni.py).

## 5.2 Additions to Network Infrastructure to Support a VPN

In order to support a VPN on our virtual network environment, while still having the traffic from out client (10.4.20.1) be routed through the meddler-in-the-middle (10.4.20.2), we needed to make a few additions to our network infrastructure. First the VPN server (10.4.20.5) was given a second IP address by executing the command *“sudo ifconfig eth0:0 193.168.4.1 netmask 255.255.255.0 up”*. Second, the routing table for the meddler-in-the-middle was modified to redirect all traffic heading for the subnet 193.169.4.0/24 to the IP address 10.4.20.5 by executing the command *“sudo route add -net 193.169.4.0/24 gw 10.4.20.5”*. Finally the client’s routing table was modified with the command *“sudo route add -net 193.169.4.0/24 gw 10.4.20.2”*. Because of these changes, when the client attempts to connect to the VPN server it will be able to make the connection to the server while still going through the meddler-in-the-middle. When the connection is made the client will be instructed by the VPN configuration files (detailed in 5.3) to connect to 193.169.4.1. The client will route this traffic to the meddler-in-the-middle, who will pass the traffic to the VPN server.

## 5.3 VPN

We believe this defense to be both realistic for a user who is having their DNS requests tampered with by a meddler-in-the-middle. In this second stage of our defence where we set up and used a VPN, we used the open source VPN software Open-VPN [7]. The VPN connection encrypts the traffic leaving the client machine (10.4.20.1) and securely transfers it to the VPN server (10.4.20.5), even though the traffic is being carried through the meddler-in-the-middle (10.4.20.2).

Open-VPN was installed by executing the command *“sudo apt-get install openvpn”* on both the client and the VPN server. A static encryption key for the VPN connection was generated on the VPN server by executing the command *“sudo openvpn --genkey --secret static.key”* from the cs4404 home directory. The static key was securely copied to the home directory on the client by executing the command *“scp static.key cs4404@10.4.20.1:/home/cs4404/”* from the same directory that static.key is currently located in on the VPN server. The directory “*/usr/share/doc/openvpn/examples/sample-configurations/”* was then navigated to on both the client and the server by executing the command *“cd /usr/share/doc/openvpn/examples/sample-configurations/”*. The static key was then copied to this directory on both the client and the server by using the command *“cp /home/cs4404/static.key .”* on both the client and the server from the “*/usr/share/doc/openvpn/examples/sample-configurations/”* directory. IP forwarding was then enabled on the server using the command *“sudo sysctl -w net.ipv4.ip\_forward=1”*. For the client VPN configuration, the file *“static-home.conf”* was modified to reflect the changes seen in figure 14. For the server configuration, the file *“static-office.conf”* was modified to reflect the changes seen in figure 15.

We ran the VPN by first starting the VPN server on the server machine (10.4.20.5) using the command *“sudo openvpn --script-security 2 --config static-office.conf”* which was run from inside the same sample-configurations directory. On the client the command *“sudo openvpn --script-security 2 --float 10.4.20.5 --config static-home.conf”* was run from the same directory to establish a connection to the VPN server. After the connection was established we were able to send DNS requests from the client, through the VPN to the VPN server, and to the Normal DNS that is being hosted on the same virtual machine. The success of using a VPN as a defense can be seen in the defensive OOONI scan in figure 17. The scan from oooni.py was run after the VPN connection was started, and the output confirms that the DNS queries do not appear to be tampered with.

# 6 Conclusion

DNS is extremely important for end users, organizations and overall working structure of the internet. While originally DNS was not designed to avoid all the security risks, currently it has three security goals: origin authentication, integrity and availability. In this report we have explored the DNS and it’s infrastructure, it’s security goals and countermeasures against different types of attacks. We have demonstrated one type of attack that puts origin authentication and integrity security goals at risk by redirecting the client to the website that ISP wants the user to use. As we have discussed in section 2.3, while the public outrage and the regulations that FCC has put forth has reduced the number of privacy violations by the ISP’s, it by no means eliminated them. ISP’s have an extremely advantageous position to spy on user’s traffic and use the data to generate additional profits and as long as that remains true, ISP’s will continue to try and find ways to collect and misuse user’s data. In this report we also demonstrated a successful mitigation of the ISP redirection attack using OOONI detection script and VPN. It is therefore our recommendation that the users use tools similar to OONI to scan their traffic and determine whether there is any censorship or redirection going on. While we do acknowledge that OONI is not a comprehensive tool that can detect all types of attacks, we still recommend it for the simplicity and ease of its use and as a good starting step. More importantly, we do recommend that both end users and organizations consider using VPN. End users can use VPN tools to avoid censorship and gain access to information in particularly oppressive countries such as China. However, we would like to note that VPN’s should not only be considered for the purposes of accessing sensitive or frequently restricted websites. We suggest that end users employ VPN’s in general to avoid their data collected and used against them which can result in monetary loss. Organizations can also use VPN for added security, to make sure their traffic is not easily readable and identifiable. Many work environments today use VPNs to give users the option to work remotely and they should consider widening the deployment for enhancing the security.

# 

# Sources

[1] "DNSSEC – What Is It and Why Is It Important? - ICANN", *Icann.org*, 2019. [Online]. Available: https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en. [Accessed: 10- Nov- 2019].

[2] Giuseppe Ateniese and Stefan Mangard. "A new approach to DNS security (DNSSEC)," ACM Conference on Computer and Communications Security, 2001. Available: https://ia.wpi.edu/cs4404/play\_event.php?page=attachment&id=27.

[3] Douglas C. MacFarland, Craig A. Shue , and Andrew J. Kalafut. “Characterizing Optimal DNS Amplification Attacks and Effective Mitigation”. Available: https://web.cs.wpi.edu/~cshue/research/pam15.pdf

[4] L. Rendek, "Linux DNS server BIND configuration - LinuxConfig.org", *Linuxconfig.org*, 2019. [Online]. Available: https://linuxconfig.org/linux-dns-server-bind-configuration. [Accessed: 10- Nov- 2019].

[5] "How to redirect an incoming connection to a different IP address on a specific port using IPtables - eSecureData Inc.", *My.esecuredata.com*, 2019. [Online]. Available: https://my.esecuredata.com/index.php?/knowledgebase/article/49/how-to-redirect-an-incoming-connection-to-a-different-ip-address-on-a-specific-port-using-iptables/. [Accessed: 14- Nov- 2019].

[6] "OONI - Open Observatory of Network Interference", *Ooni.torproject.org*, 2019. [Online]. Available: https://ooni.torproject.org/. [Accessed: 14- Nov- 2019].

[7] "How To Guide: Set Up & Configure OpenVPN client/server VPN | OpenVPN", *OpenVPN*, 2019. [Online]. Available: https://openvpn.net/community-resources/how-to/. [Accessed: 17- Nov- 2019].

[8] S. Son and V. Shmatikov, "The Hitchhiker's Guide to DNS Cache Poisoning", 2010. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.5i-025&rep=rep1&type=pdf. [Accessed: 14- Nov- 2019].

[9] Gillula, "Five Creepy Things Your ISP Could Do if Congress Repeals the FCC’s Privacy Protections", *Electronic Frontier Foundation*, 2019. [Online]. Available: https://www.eff.org/deeplinks/2017/03/five-creepy-things-your-isp-could-do-if-congress-repeals-fccs-privacy-protections. [Accessed: 17- Nov- 2019].

# Appendix

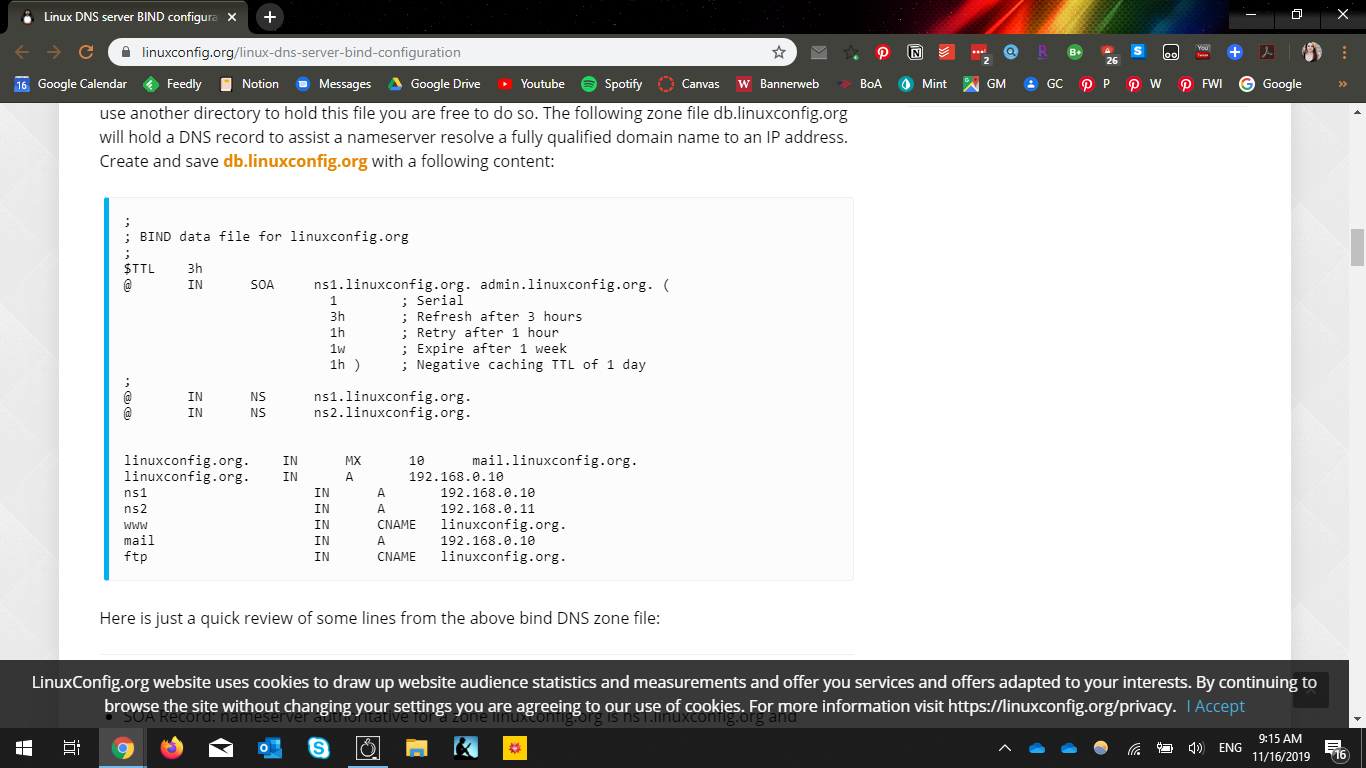
Figure 1. Instructions from “Linux DNS server BIND configuration” article on setting up BIND data file.  


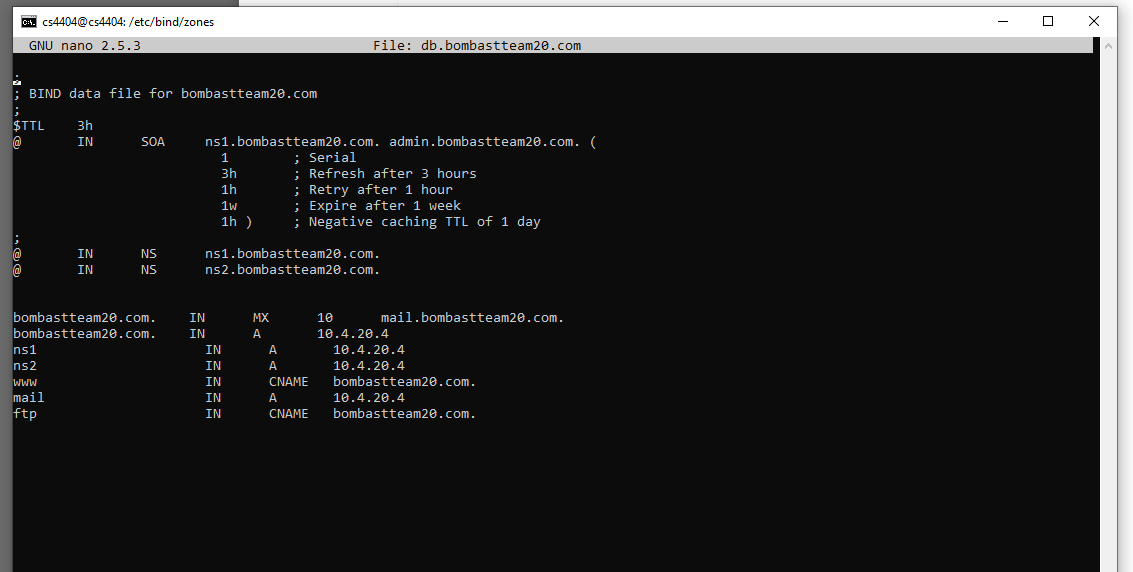
Figure 2. Modified BIND data file for our implementation, containing our bombastteam20.com domain and 10.4.20.3 IP address for DNS.   


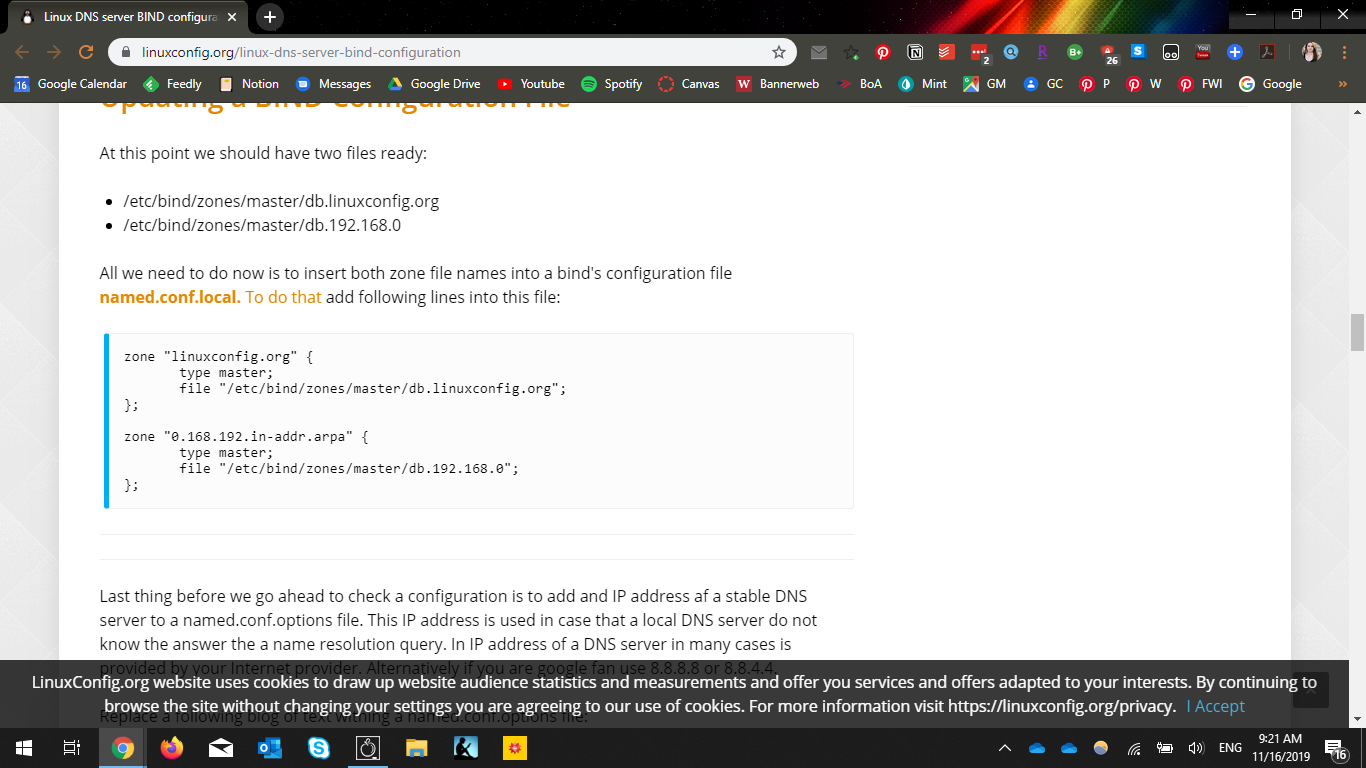
Figure 3. Suggested set up of the named.conf.local file from “Linux DNS server BIND configuration” article.   


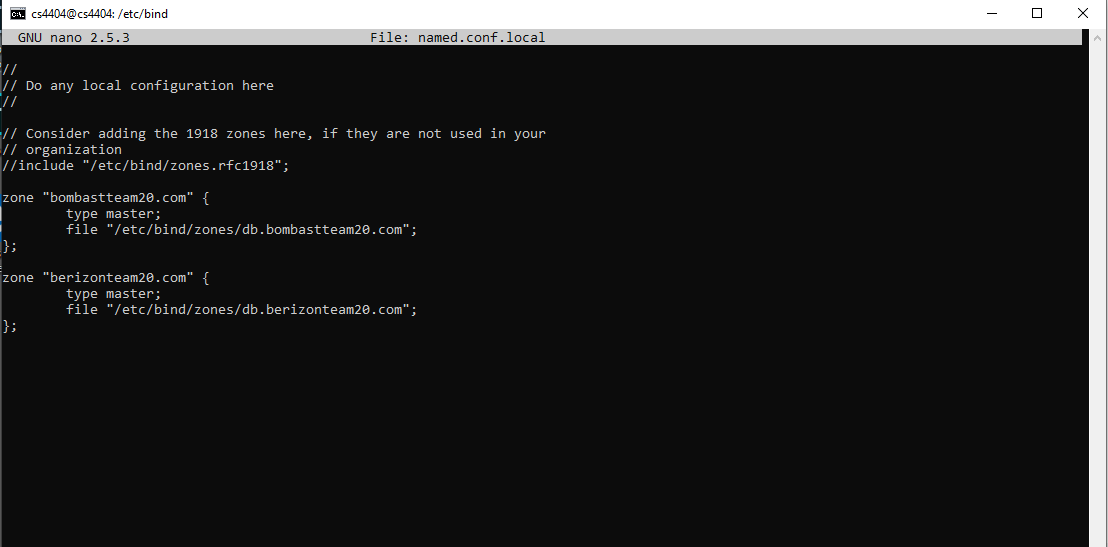
Figure 4. Modified named.conf.local file from our implementation for Bombast DNS and Normal DNS.   


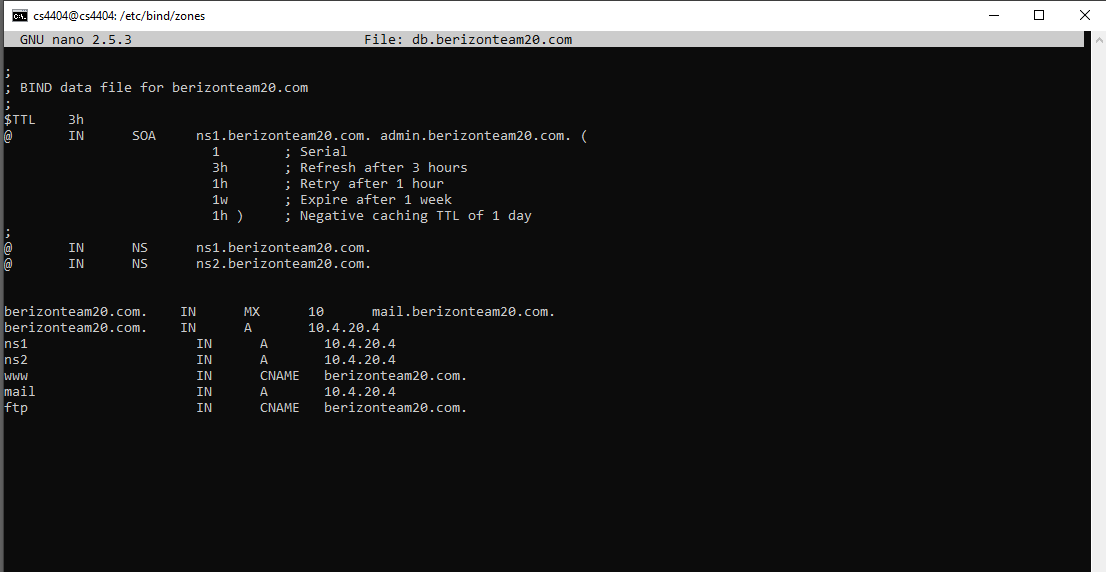
Figure 5. Modified BIND data file for our implementation, containing our berizonteam20.com domain and 10.4.20.3 IP address for DNS.   


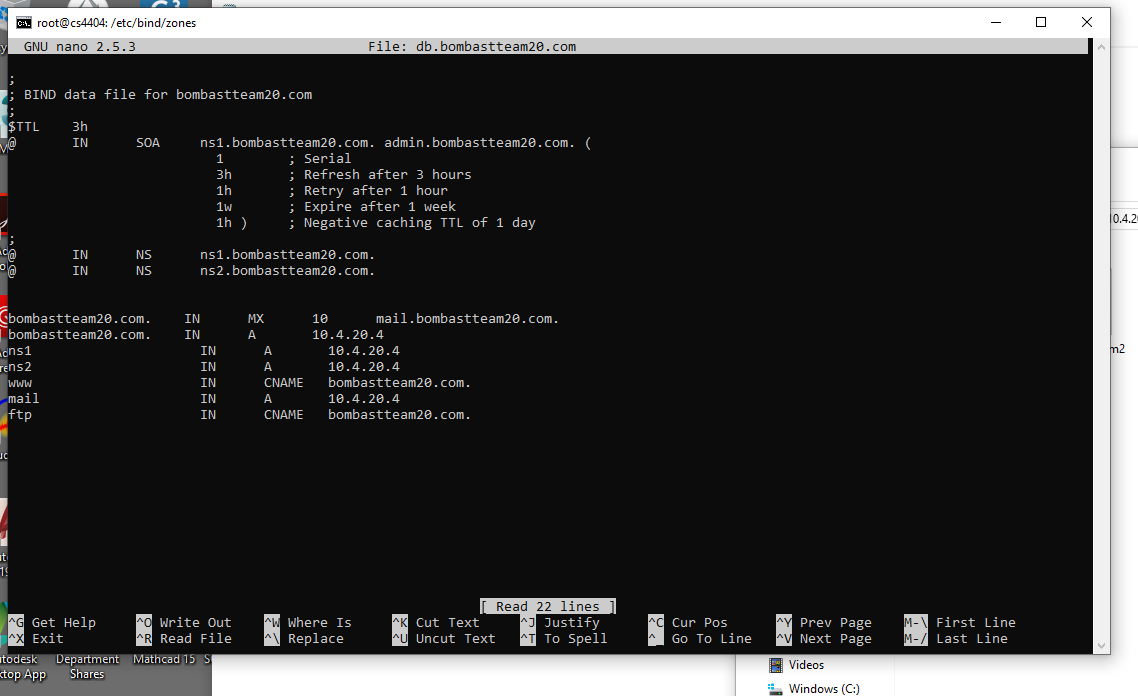
Figure 6. Modified BIND data file for our implementation, containing our bombastteam20.com domain and 10.4.20.5 IP address for DNS.   


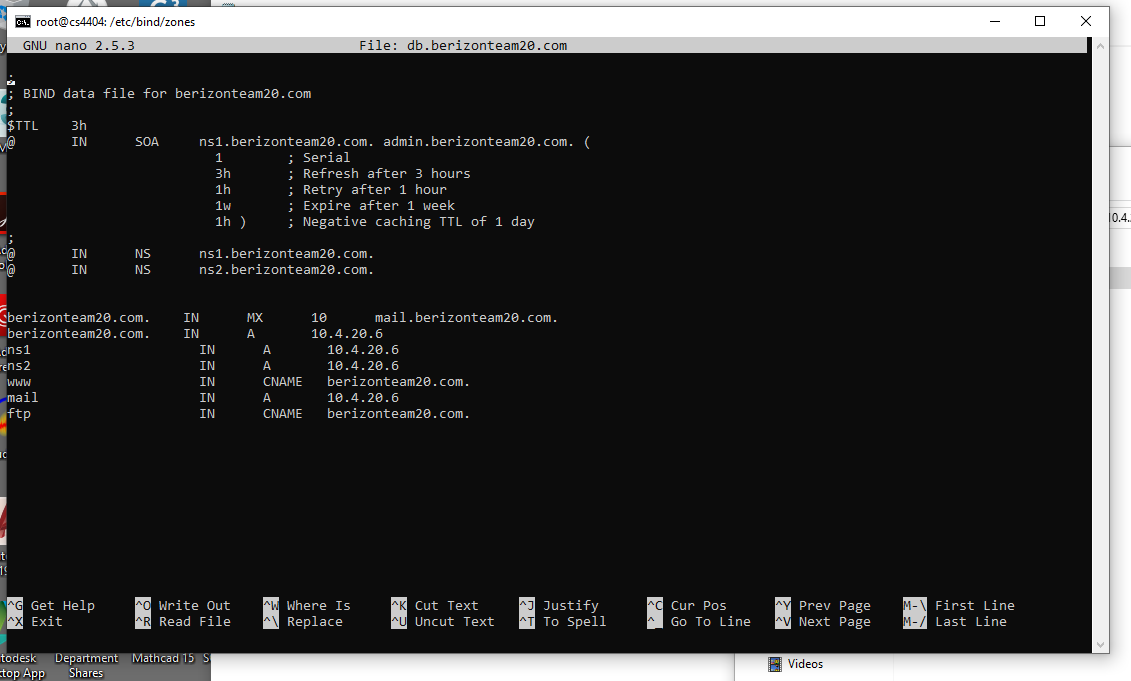
Figure 7. Modified BIND data file for our implementation, containing our berizonteam20.com domain and 10.4.20.5 IP address for DNS.   


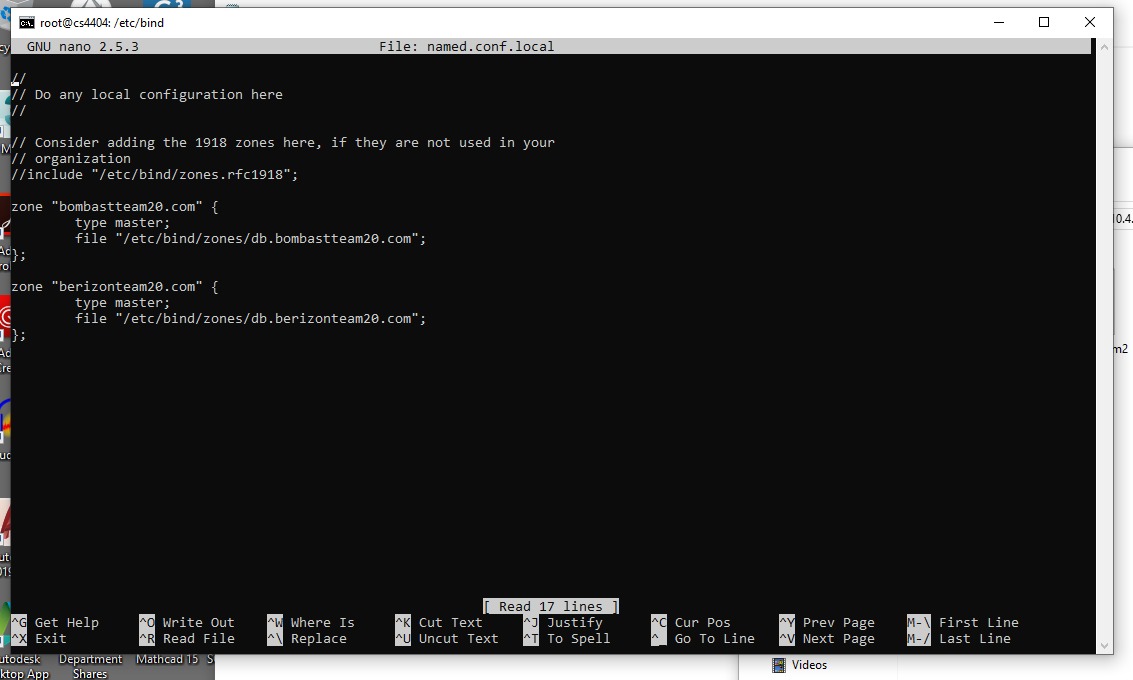
Figure 8. Modified named.conf.local file from our implementation for Bombast DNS and Normal DNS located on the Normal DNS server.   


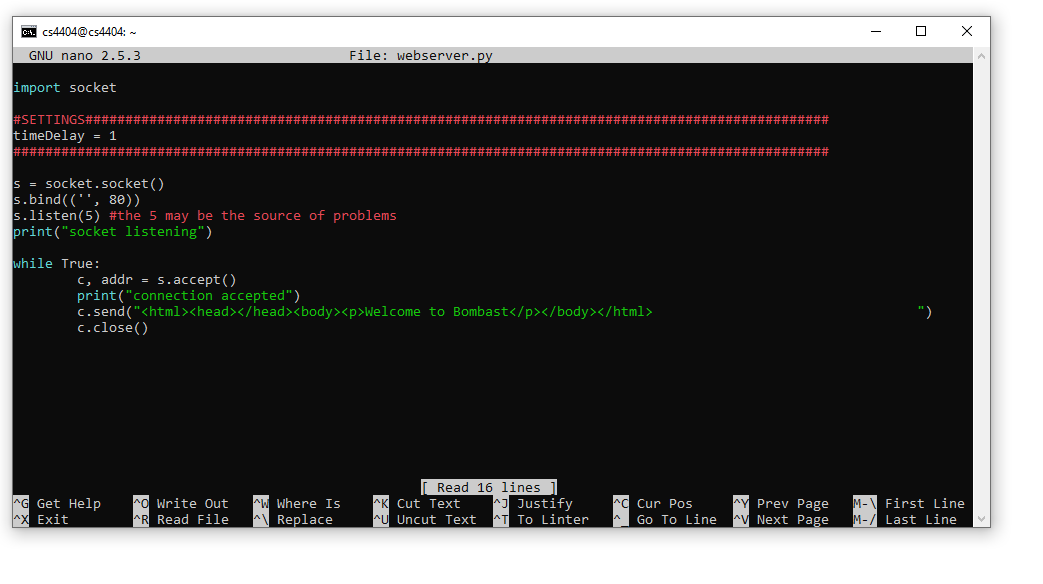
Figure 9. webserver.py code responsible for acting as a web page for the compromised website hosted on 10.4.20.4.  


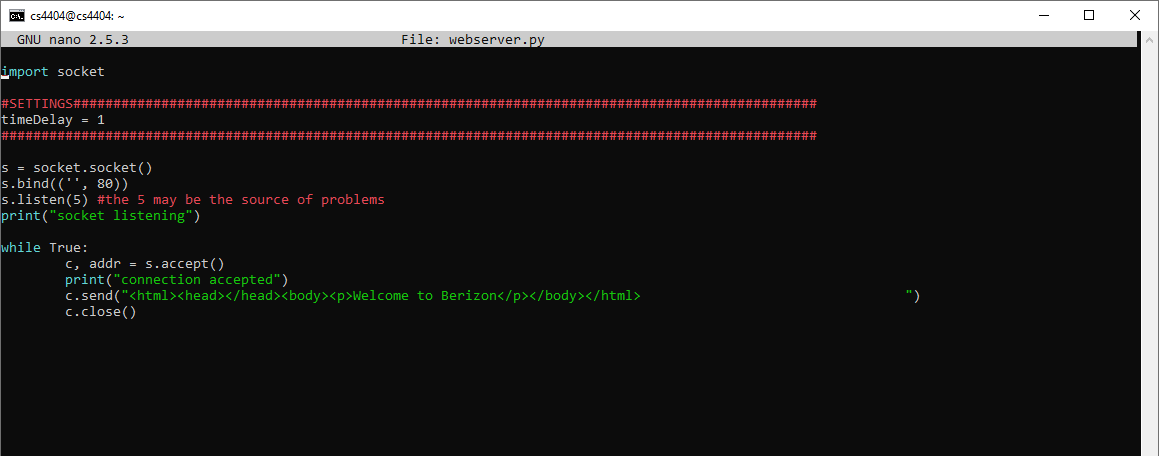
Figure 10. webservery.py code responsible for acting as a web page for the site the client is trying to reach hosted on 10.4.20.6  


Figure 11. Bad firewall rules file hosted on 10.4.20.2  


Figure 12. Result of oooni.py script with a man in the middle attack, with no defense.

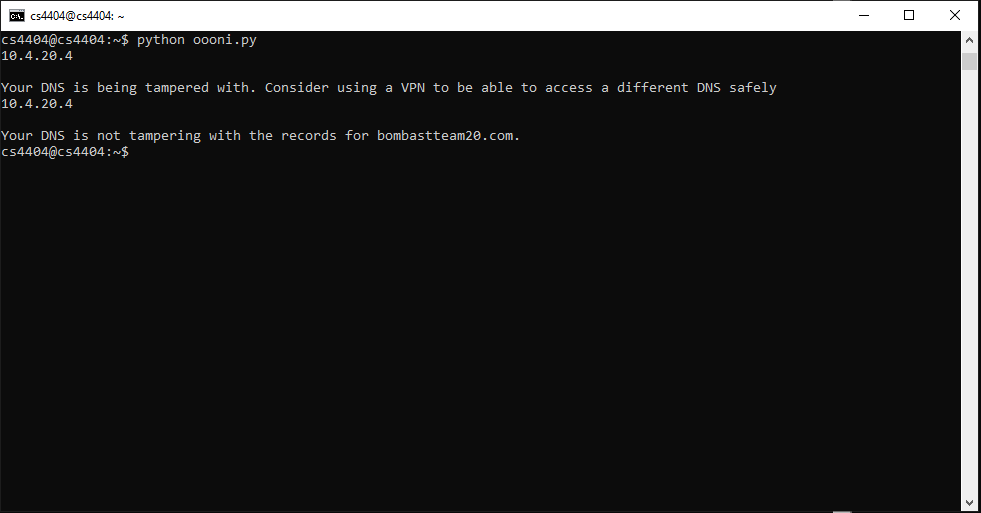


Figure 13. Result of oooni.py script with an active VPN in place

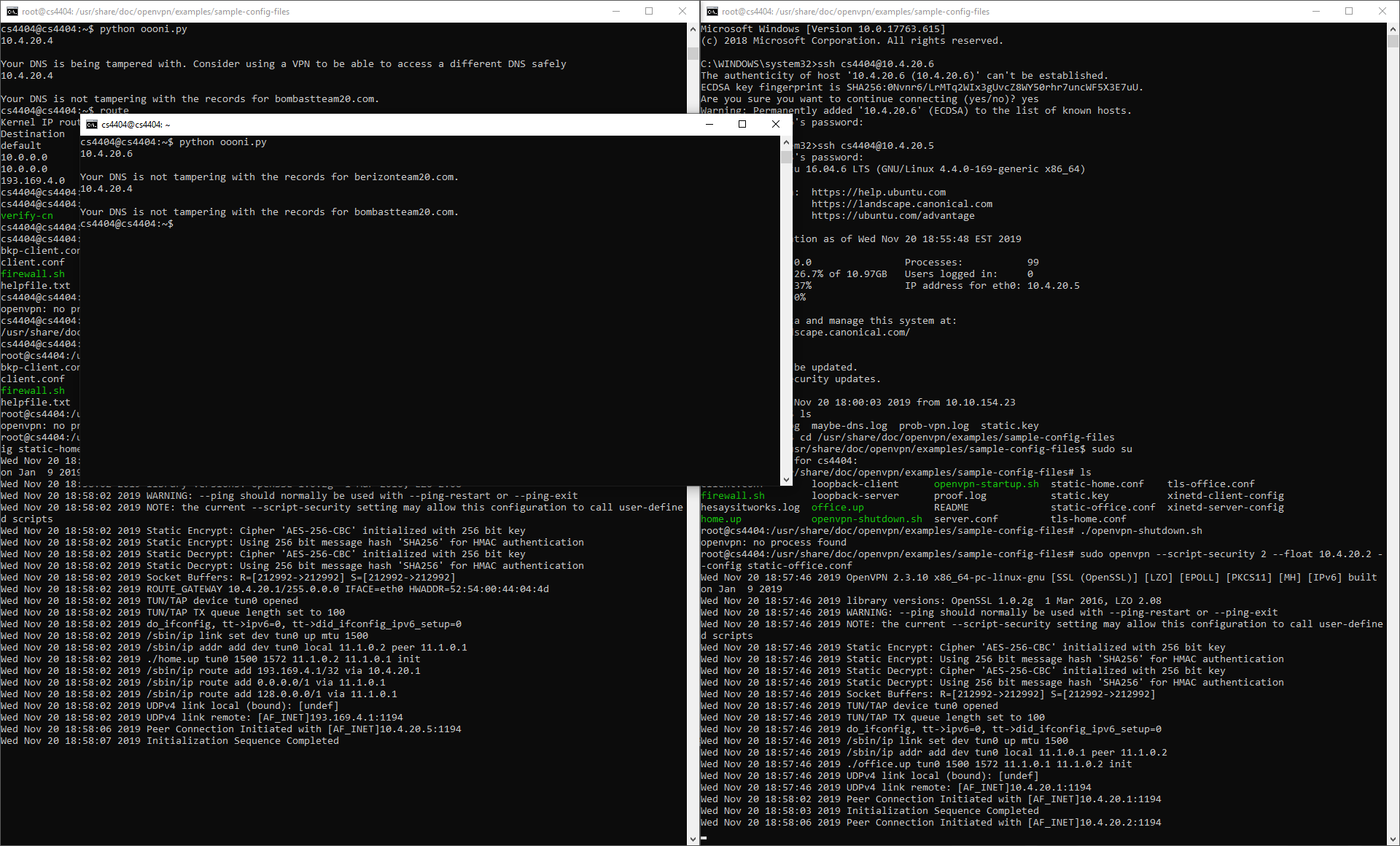


Figure 14. VPN configuration file for the client

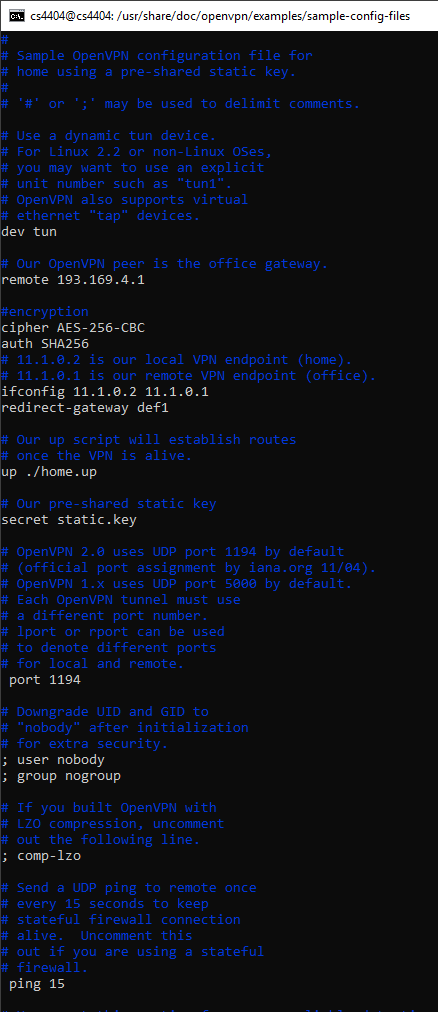


Figure 15. VPN configuration file for the server

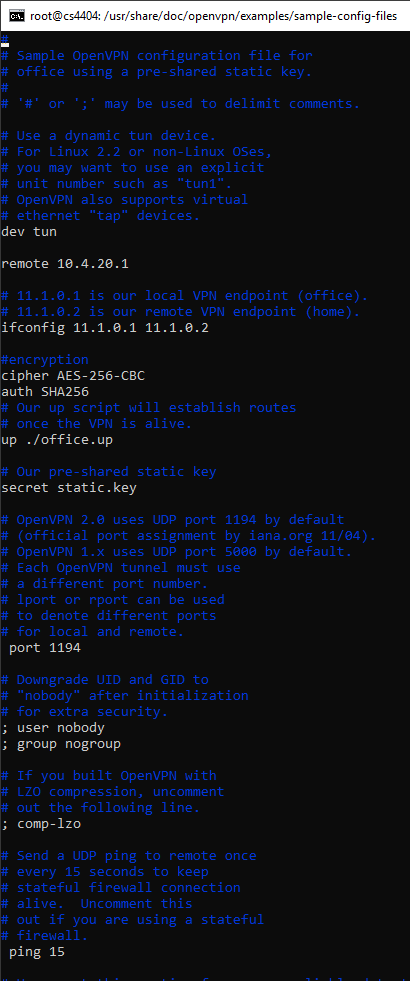


Figure 16. Client being redirected to the Bombast webpage instead of Berizon

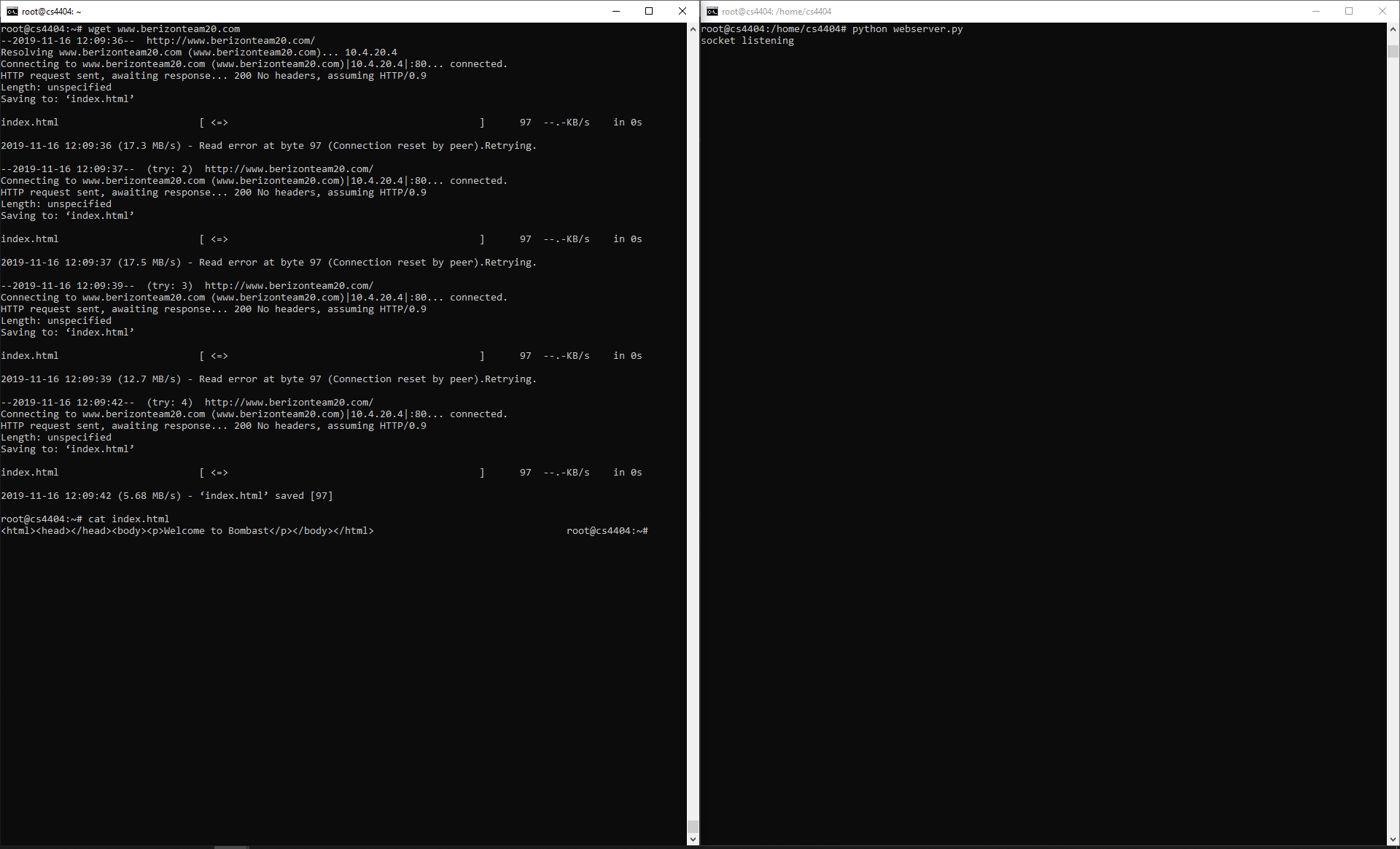


Figure 17. VPN running, and OOONI notifying the user that their DNS queries appear to be safe

