NCS 495

NCS Capstone Project – DNSSec/DANE Web Browser Validator Reliability

Final Paper Final Draft

11/24/17

Shane Callaghan

Table of Contents

INTRODUCTION

Your surfing the web for research or shopping online for the holidays. Regardless of what you're doing online, you enter the url of a website into a web browser and you're brought to the front page. If I told you the website you're visiting was not what it appeared to be, would you believe me? What I mean is that facebook.com may not be the true website you're accustomed to. What if amazon.com suddenly gives you a pop up message that says “Your account has been suspended, enter your credentials to authenticate yourself into the system”, Could you tell whether if this was real or fake? Anything on the Internet is true, right? For a website to be acessed on the Internet, it has to use the Domain Name System, or DNS, to find its equivalent IP address. The IP address would allow a computer to access a certain service, such as web or email server across the Internet. So what this means is that every website has a specific IP address that resolves to their web server. Unfortunately, this protocol can be easily manipulated that could resolve a user to the wrong IP address, as DNS will accept any IP address as being “correct”. Thankfully, we have a solution to protect the vulnerable Domain Name System called DNSSec, or DNS Security. This layer of protection adds the capability for DNS to ensure that a domain always points to its legitimate IP address.

BACKGROUND

To explain how DNS ever came into existence, we have to travel back in time to the very begining of the Internet. Palermo and Cox (2014) tells us that the concept of the Internet came from a computer scientist named J.C.R Licklider who dreamed of an “Intergalactic Computer Network”. Fortunately, Mr. Licklider had the right idea at the right time, as the U.S. Government was trying to create a communication system that could survive a nuclear attack during the 1960's. The ideas of our computer scientist landed him a job as the head of the U.S. Department of the Defense Advanced Research Projects Agency, or DARPA. This new organization created the first ever computer network called the ARPANET. It would be the downfall of this project that would lead to the creation of the

DNS protocol we all take for granted.

Rader (2001) mentions that when the ARPANET first started out, their network was small enough that their users could easily navigate from one service to another. The project ran into issues when the network grew in size. This made finding the right service increasingly difficult for any user. There was one attempt to solve this problem from Request for Comment (RFC) 226 which called for using “host mnemonics” or Internet names. What evolved from this new concept was the introduction of the hosts.txt file which mapped IP addresses to their corresponding hostname in 1972. This was a temporary solution at best, as any new entry that was added, had to be mailed to the Stanford Research Institute (SRI), the organization running the ARPANET. When they received this new information, they would update their master copy of the hosts.txt file and send out the latest version to every computer operator. This method caused absolute chaos as computer operators needed to constantly update their hosts.txt file on their networks or their users would suffer from name collosions. This was a great start to ensure users can access the correct service using a certain name, but its scalability was very limited. A more versatile solution was badly needed if this system was to be expandable.

As our writer from Softpanamora (2001) states, the Domain Name System originated from RFC 799 written by Dr. David Mills in 1981. His ideas were used as the outline and requirements needed to have an Internet Domain System that would be able to handle the IP address resolution for thousands of hostnames. There was one attempt of developing this domain system from, Jon Postel, from the Information Sciences Institute (ISI), and, Zaw-Sing Su, from the SRI. They shared the first outline of the domain system and how it would allow for cross-network access according to RFC 819. Dr. Paul Mockapetris contributed to the DNS protocol when he wrote RFC 882 and 883. The most important components of these two articles were the concepts of delegation and authority. Authority refers to the

“sphere of influence that one has complete control over”, in the case of DNS we use this idea on a zone-to-zone basis. For instance, SUNY Polytechnic Institute has complete authority of sunypoly.edu, as well as any subdomains. A few examples would include: mail.sunypoly.edu, ftp.sunypoly.edu, etc. Delegation is the act of giving separate authority to certain parts of a domain. If google.com wanted to give part of their domain to their canadian offices, they could create the subdomain ca.google.com for them to control while working under the google.com domain. These two concepts would help a domain request find its answer by starting from the the root zone, all the way to the final destination.

During this time, there was a need to define a domain by the type of organization it originated from. When Jon Postel and Joyce Reynolds published RFC 920, they laid the foundations on how domains from a specific type of business or country were represented on the Internet. The way they achieved this goal was through the use of Top Level Domains, or TLD. A few examples include .ca, .us, and .uk representing Canada, the United States of America, and the United Kingdoms respectively. For the business aspect, they came up with some very recognizable TLDs such as .com and .org (Rader, 2001). To explain how this method is effective, let's share a few scenarios. If someone wanted to purchase something online they would go to a website like amazon.com, or ebay.com because they are the Internet entities of these commercial companies. If an individual needed to research the American Psychological Association, they could go to apa.org as this website represents their organization online. This addition to the Domain Name System made it easier to know the type of website a user is visiting.

DNS

Now that we know how DNS was created, it is time to explain how this protocol works to resolve a domain name to its equivalent IP address. Just to clear up any confusion some may have of the terminology, we will be defining the following according to www.daniellbenway.net, domain zones,

recursive,and iterative. A domain zone refers to a certain section of the Domain Name System namespace. (www.example.net., is the www server of the 'example.net' domain zone, found in the '.net' domain, which is also apart of the root domain ("."). Recursive refers to a DNS request where its told to find an answer and can ask anyone it wants (Benway, 2016). This is usually done by a local DNS server as it ask many domain zones the same exact question. Iterative is similar to a DNS request, only this time the question being asked can only be answered by one source (Benway, 2016). When the recursive DNS server is given a response, it may be told to ask only one source for more information on their domain request. If some of you are still a little lost, don't worry as we will explain each step of resolving a domain name according to dyn.com. As our sources from dyn.com states (2010), when a computer requests the IP address of a domain, it will check its local cache, a file stored on the computer that has a listing of domain names to IP addresses. If the requested domain name is not found, then it will send that query to the local DNS server. The local server will send the domain request to the root domain zone. The contacted root DNS server would then respond with an IP address of a TLD server that is apart of the domain request (such as .com or .edu). When we arrive at the TLD domain server, our query would then be pointed to the authoritative server of the requested domain (ex. www.example.net, example.net being the authoritative server). At our final destination, we ask the authoritative server for the IP address of the full domain request (i.e.what is the IP address of www.example.net?). Assuming that this domain entry exists, the local DNS server would be given a copy of the IP address for that domain. This new domain to IP address entry is stored in the cache file of the local DNS resolver. If any other computer request for the same domain, then the local DNS server would respond to said computer with the new answer from its cache file. If you need a more detailed explain refer to Figure 1 shown below. The way this system works is that we assume that every authoritative DNS server has the correct IP address for their domain and sub domains, which is very unlikely to say the least.

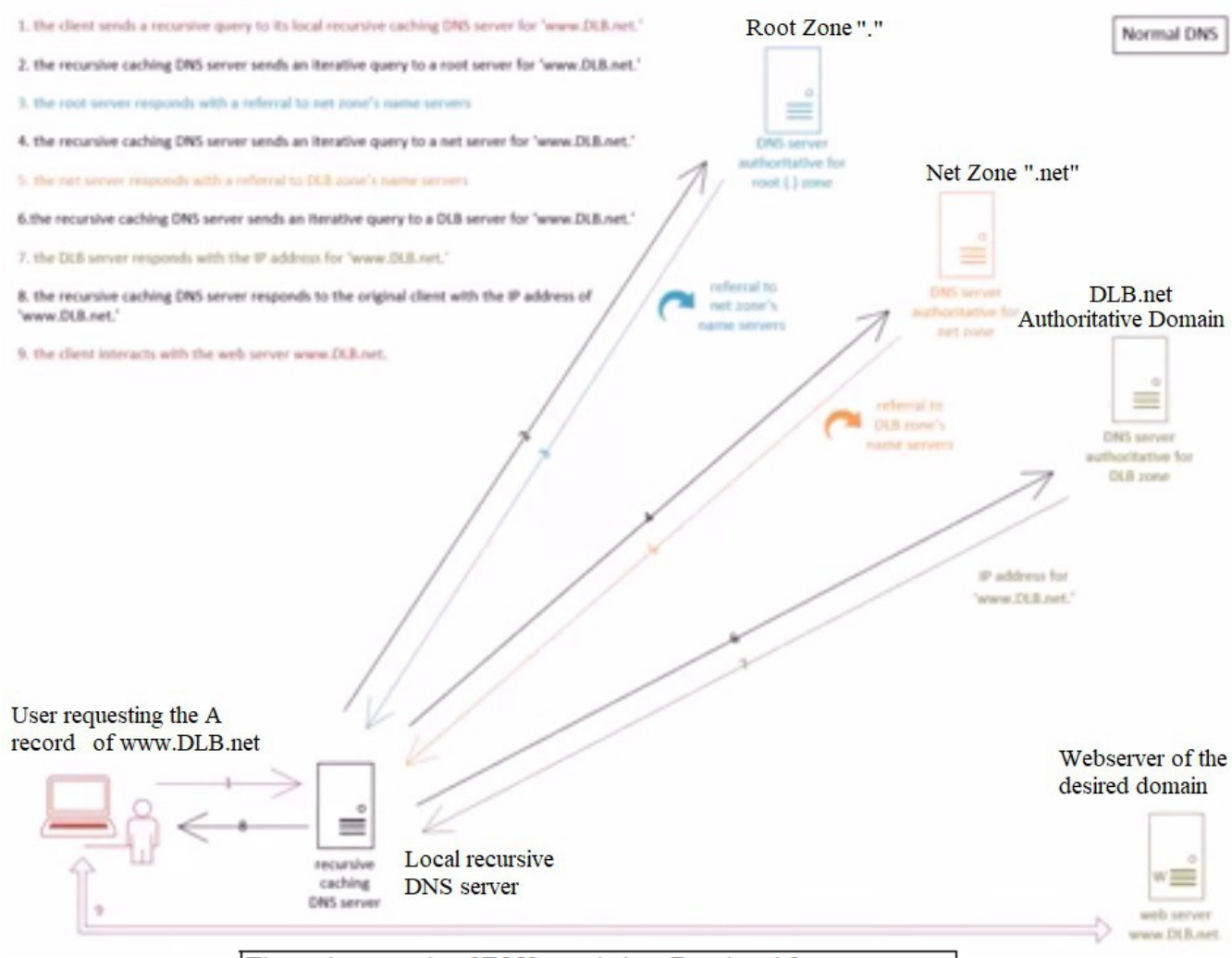

Figure 1. example of DNS resolution. Reprinted from www.daniellbenway.net, in Daniel L Benway, 2016., Retrieved September 18, 2017 from https://daniellbenway.net/DNSSec-explained/. Reprinted with permission.

DNS Spoofing

When this protocol was developed in the 1980's, no one ever thought about the security or vulnerabilities of DNS. Cyber Security was not a major concern at the start of the Internet, causing the development of these early Internet protocols, like DNS, to focus only on their main function. As it turns out, the Domain Name Sytem is very susceptible to DNS spoofing, an exploit where a user could

visit a website that appears to be legitmate, but is truly a fake (Pham, 2016). Some of us may be a tad skeptical to think any website could be untrustworthy on the Internet. But there has been plenty of evidence that can prove this statement.

Before we reveal how effective DNS Spoofing can be, we should look at how this service accepts incoming and outgoing DNS traffic. As we read in the previous section, a local DNS server will wait for the response of a domain request and store the answer it receives to its cache file. The criteria of an acceptable response has to have the following: the same UDP port it was sent from, a matching Question section, the Query ID number the server is expecting to see, and finally the Authority and Additions section need to represent names that are in the same domain as the question. Any DNS traffic that doesn't meet these requirements are dropped.

Starting from the top of list, if the response comes back on a port that is different from the one it was sent out on, then that connection will be dropped at the network layer. The Question ID of a DNS packet is the request of the domain being asked, such as “What is the A record of sunypoly.edu?”. Moving down the list, we have the Query ID, this is an identifier generated by the requesting DNS server to keep tabs of its pending queries. When a response that was made 30 seconds ago comes back to the requesting DNS server, it will know what exact request the response is referring to based on the Query ID number. The Authority and Addition section of a DNS packet should both have similar names that relate to the Question ID. Let's say a user requests www.cnyhackathon.org on their web browser. If a packet capture was running during that instance and someone inspected the contents of the DNS response, they would see that these sections would have domain entries very similar to the one stored in the Question ID. A few examples might something like score.cnyhackathon.org or www2.cnyhackathon.org (Friedl, 2008). Now that we have cleared up what a DNS server looks for in a

response, how can it be tricked into believing false information?

When DNS servers first came out, they would increment their Query ID by one for each request/response. If a false DNS response, given the right Query ID number, was able to contact the requesting nameserver before the legitimate one could, then the DNS server would accept that answer as being valid. The way this method works is that a request for domain A is sent out with a Query ID of five. The nameserver determines the response needs to have a Query ID of seven. If a hacker knew the current Query ID, that individual could send a flood of fake responses of domain A starting with a Query ID five and incrementing the value by one. If this person sends enough of them in a small amount, then they could trick the DNS server to accept their response for domain A (Friedl, 2008). To get a better visual perspective of this attack look at figure 2.

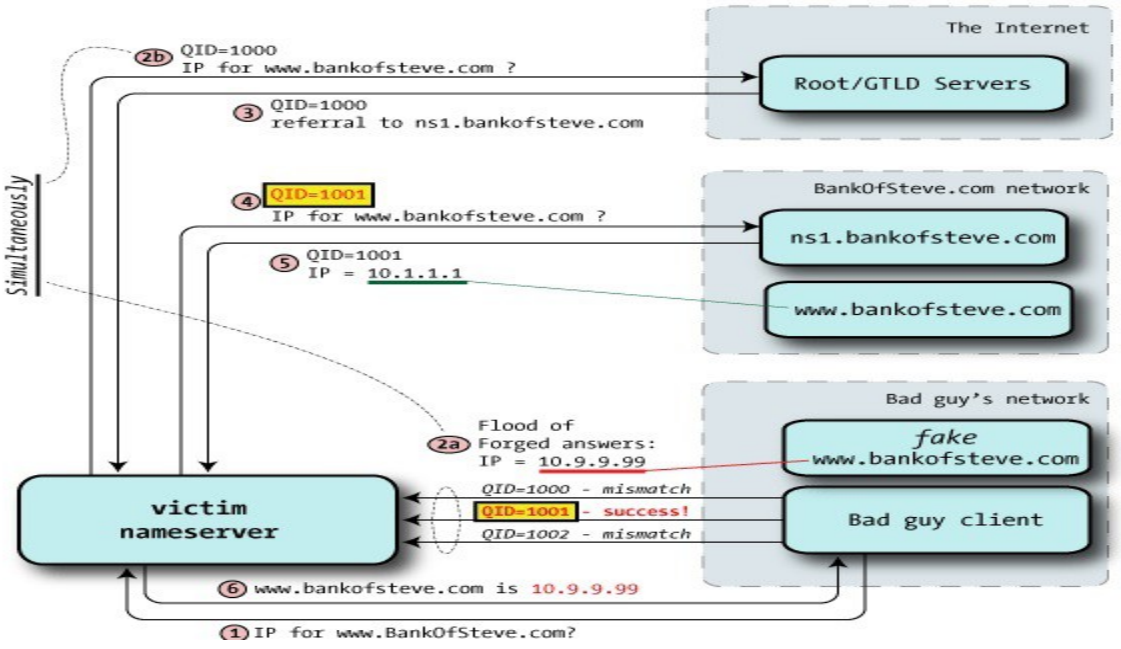

Figure 2. Example of exploiting the Query ID. Reprinted from An Illustrated Guide to the Kaminsky DNS Vulnerability, in unixwiz.net, 2008., Retrieved September 13, 2017 from http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html. Reprinted with permission

In 2008, Dan Kaminsky, a Cyber Security researcher, discovered another concerning issue with

the Domain Name System (Pham 2016). He found that someone could hijack the authority records of a DNS server by requesting a subdomain that wouldn't normally exist. The important part of this exploit is that the target nameserver would accept the IP address of the “authoritative” server from the DNS response. When the nameserver stores this false information in its cache file, then any request for this domain will be forwarded to the wrong authoritative server. Let's say we have a rogue device pretending to the authoritative server of amazon.com. This “domain” would have a number of entries that would be seen as odd such as 123.amazon.com or ftp.amazon.com, but there's one entry for www.amazon.com that points to web server hosting the false website. A DNS server will have a time limit for how long a domain to IP address entry will stay in its cache file, usually no more than a day be default. If a local DNS server had recently flushed its cache file and was asked to lookup the IP address for 123.amazon.com, it would treat that request as a normal domain request. Just like any computer on a TCP/IP network, it would send a broadcast message to its local network asking “Who is 123.amazon.com?”. Under normal conditions, there would be no answer from the local network for amazon.com. This is due to the fact that not everyone on the Internet would be on the same local network as the actual amazon.com domain. Although with our rogue authoritative server, pretending to be amazon.com, on the same local network, the nameserver would get a response and store the IP address of the “authoritative” server of amazon.com (Friedl, 2008). For a more detailed explaination, please direct your attention to figure 3.

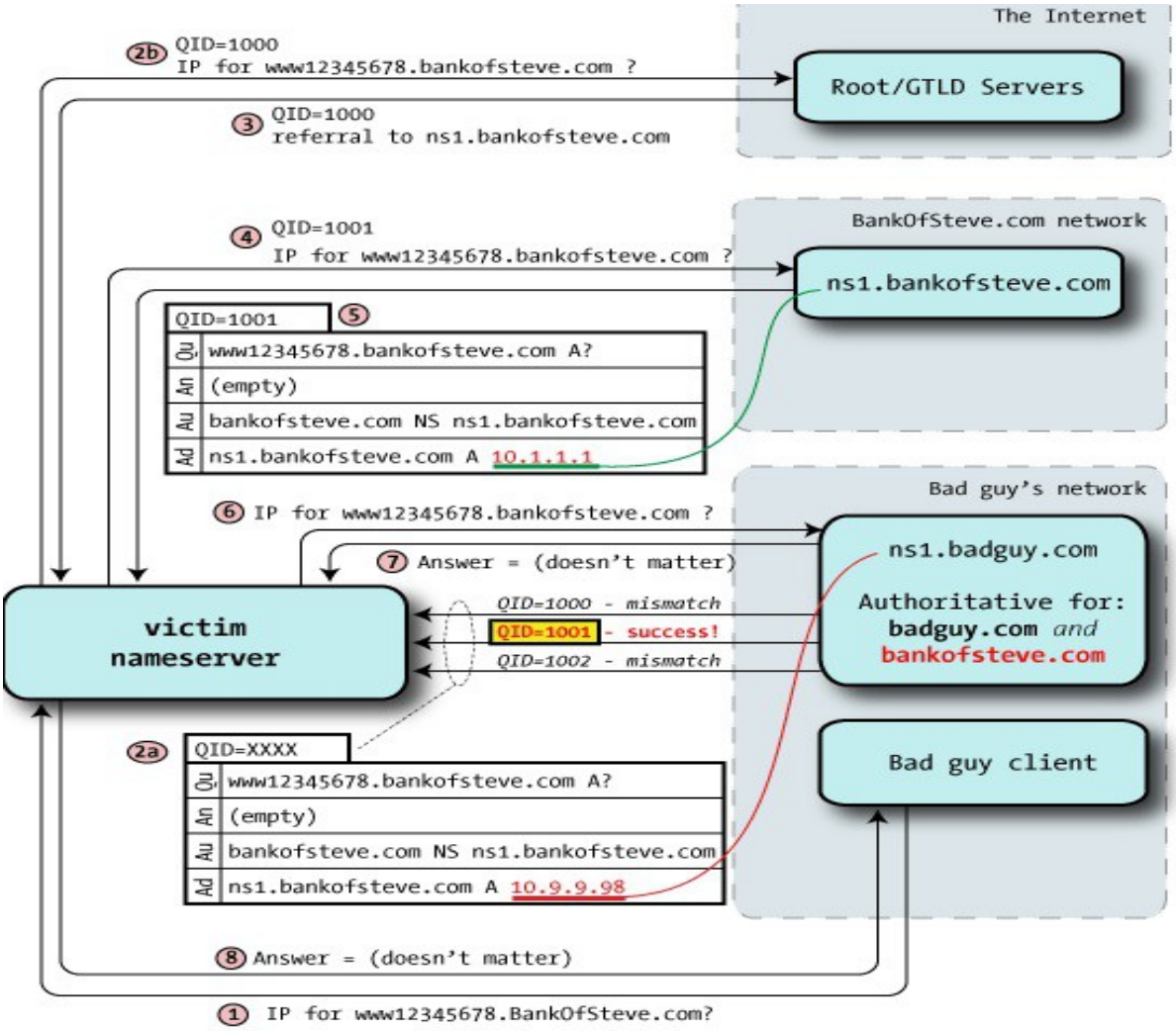

Figure 3. Example of exploiting the Authority IP address. Reprinted from An Illustrated Guide to the Kaminsky DNS Vulnerability, in unixwiz.net, 2008., Retrieved September 13, 2017 from http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html. Reprinted with permission

To prevent these threats from occuring, modern DNS servers randomize the UDP port being used in a range of zero to 2,048 (2^11) as well as the Query ID to be any number between zero and 65,535 (2^16 – 1),. If we were combine both of these countermeasures, we would have combination of 2^27 or 134 million possible combinations (Friedl, 2008). This just adds more to the fact that DNS is very insecure by design and there's a constant need to improve the protocol.

Another method to pull off DNS Spoofing is by a Man in the Middle attack (MITM). Unlike the previous vulnerabilities, this one targets the user instead of the local nameserver. A MITM attack will target a single machine by intercepting any communication between it and the default gateway and passing the information between the two hosts after capturing the contents of the traffic. Using a DNS MITM attack, the program waits for the victim machine to search for a specific website from a web browser. Once that user tries to reach said website, the program would be redirect them to the attacker's version of the website (Sanders 2010).

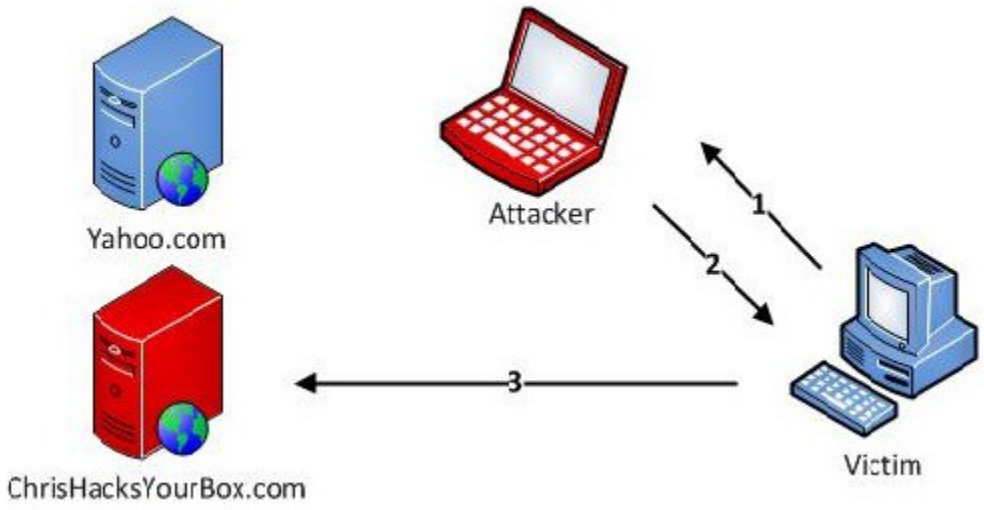

Figure 4. DNS MITM example. Reprinted from Understanding Man-In-The-Middle Attacks – Part2: DNS Spoofing, in techgenix.com, 2010., Retrieved November 11, 2017 from http://techgenix.com/understanding-man-in-the-middle-attacks-arp-part2/. Reprinted with permission.

One thing to keep in mind with it comes to DNS Spoofing, the act of spoofing a website is not exactly harmful. The reason why this is seen as malicious is what the hacker is trying to accomplish. A user may get a pop up message that says “Your account has been suspended. Please authenticate yourself please enter your credientials in this form” to steal usernames and passwords. Maybe a hacker

wants to install a virus on someone's machine when they visit their website. Regardless of how this attack is performed, DNS needs to be fixed in order to protect the trust of the Internet.

DNSSec

It may seem like DNS is horribly insecure and there's no way to prevent DNS Spoofing from occurring. Luckily, that statement is only half correct. The Domain Name System can still be easily exploited, but there is a solution that could verify a domain to be the correct entity. The needed solution for the issues surrounding the DNS protocol is called DNS Security, or DNSSec for short.The objective of DNSSec is to ensure that any domain on the Internet resolves to the correct IP address. With this added security, every domain zone on the Internet can prove who they claim to be. The way this goal is achieved is by introducing Asymmetric Cryptography into the domain resolution process (Benway, 2016).

This type of cryptography involves using two keys: one public and one private. One key is used to decrypt (public key), while the other encrypts (private key) information (Rouse, 2016). To show how this is used in the real world, here's a simple scenario. Alice attempts to send an email to Bob. She creates her message and hashes it into a digest. Alice would then encrypt said digest with her private key to get a digital signature. Afterwards, Alice would then include the message and digital signature into the email for Bob. When Bob receives this email from Alice, he is given two things: the original message and a digital signature of the message digest. Assuming Bob is well trained to be suspicious of anything he gets in his inbox, he doesn't really know if this message is actually from Alice. What if the the message from Alice was spoofed by someone posing as her on the Internet, or the content of the message was tampered with? Both of these problems can be solved by using the public key from Alice. For the sake of clarity, let's just say Bob already has a copy of her public key. He would then decrypt

the digital signature using Alice's public key to get the resulting digest. Followed by hashing the message from Alice for the resulting digest. Bob would then compare these digest values together. If their values equal to each othen, then Bob knows two things. The message was signed by Alice using her private key and the integrity of the message was unaltered. (Benway, 2016). For more details refer to figure 5. This sounds like it can prevent domains from being resolved to the wrong IP address using the correct key pair, but how can we implement this technique into DNS?

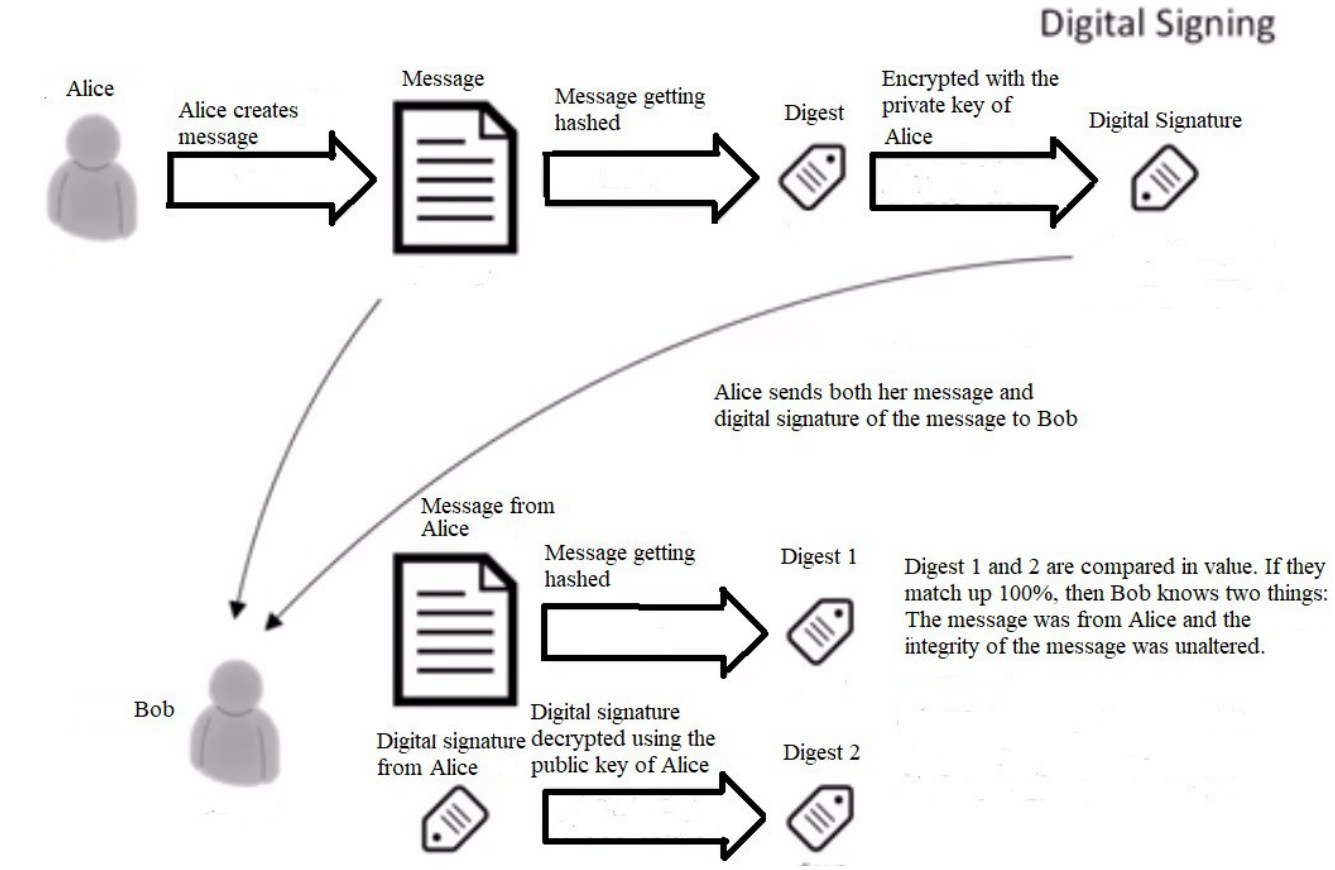

Figure 5. Example of Asymmetric Encryption. Reprinted from www.daniellbenway.net, in Daniel L Benway, 2016., Retrieved September 18, 2017 from https://daniellbenway.net/DNSSec-explained/. Reprinted with permission.

To show how Asymmetric Cryptography is involved with resolving a domain under DNSSec, we will refer to Daniel Benway, from www.daniellbenway.net. Mr Benway (2016) begins by telling us there are two types of cryptographic keys used in DNSSec: key signing keys (KSK) and zone-signing keys (ZSK). KSKs are used to verify a domain zone's keys, while ZSKs authenticates a domain zone's non-key records.

Before we explain how the ZSK and KSK help resolve a domain under DNSSec, I need to mention one requirement. In order for a recursive nameserver to be DNSSec enabled, it needs a copy of the public KSK from a root DNS server installed in its operating system. When a recursive nameserver contacts a root DNS server, it will receive the following items: A referral for the TLD zone based on the domain request, DNSKey records made up of the public/private ZSK pair, an RRsig of the DNSKey records signed by the private KSK, a DS record containing the digest of the public KSK of the TLD server, and its equivalent RRsig signed by the private ZSK. Using the public KSK, the DNSKey RRsig is decrypted for its digest. At the same time, the DNSKey records are hashed for their own digest value. These two digests are then set equal to each other. Assuming their values match each other, the recursive nameserver knows the ZSK keys, contained in the DNSKey records, are valid. We can know move on to verifying the DS records and their RRsig equivalent. Like the previous step, the RRsig of the DS record is decrypted by the public ZSK for its digest and the DS record is hashed for another digest. Once again, these hash values are compared to each other. If they're a match, then the recursive DNS server can trust the DS record for the TLD zone (Benway, 2016). If you need to see a more detailed description, please refer to figure 6 shown below.

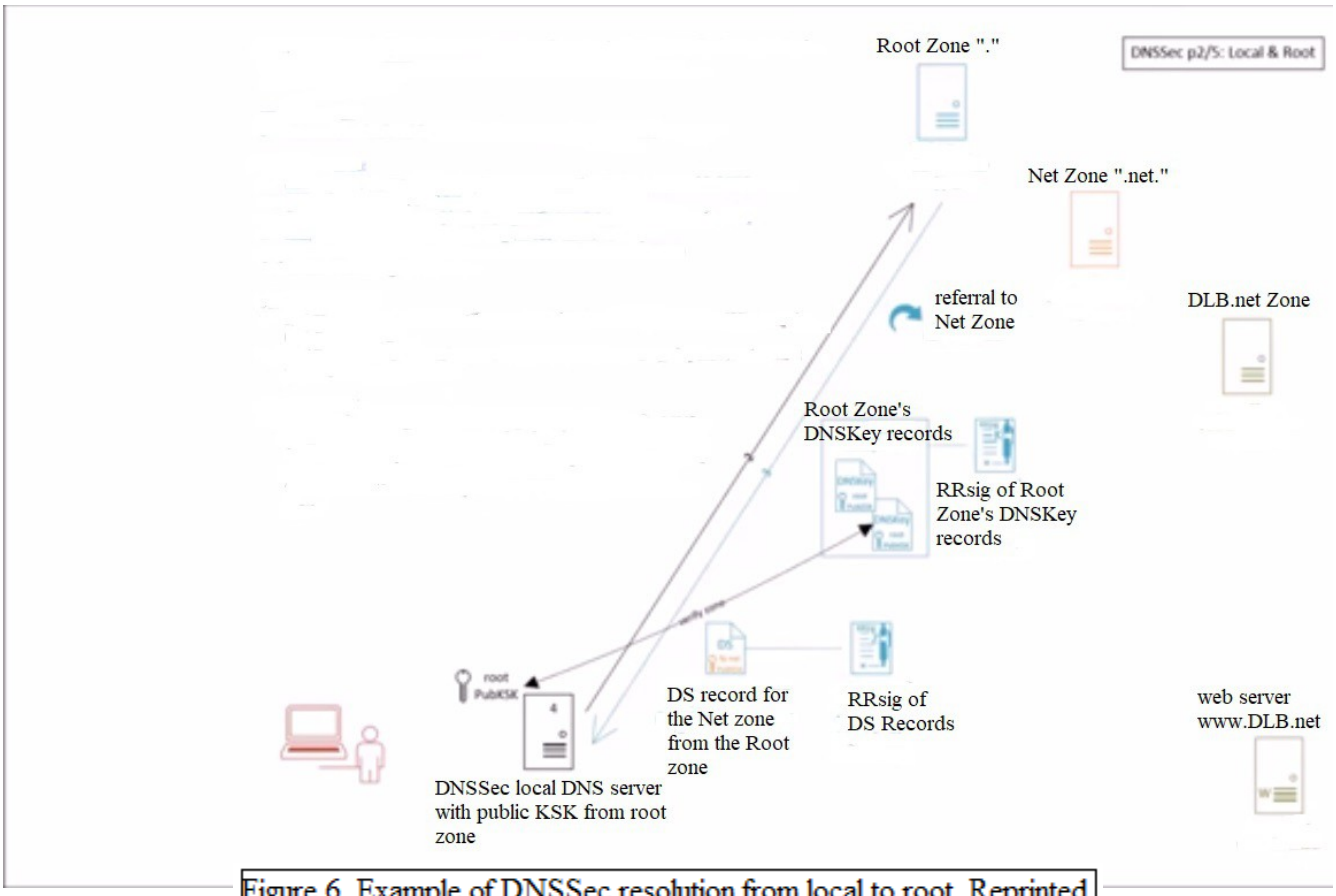

Figure 6. Example of DNSSec resolution from local to root. Reprinted from www.daniellbenway.net, in Daniel L Benway, 2016., Retrieved September 18, 2017 from https://daniellbenway.net/DNSSec-explained/. Reprinted with permission.

When the domain request started its adventure, we were able to confirm the identity of a root DNS server because we had its public KSK. The next question to ask is, how can we authenticate any domain zone past the root zone without its public KSK? Thankfully, Daniel Benway (2016) explains that as the recursive DNS server contacts the TLD zone, it will go through the same exact process as it did with the root domain. Only this time there will one exception. The DNSKey RRset of the TLD zone will hash the public KSK for a digest. We would then take this digest and compare it with the DS record, or digest, of the this server's public KSK. An important note to mention is that the DS record was given to us by the root zone. (Benway, 2016). To avoid sounding redundant, we are going to say

that our recursive DNS server successfully validated the TLD server as being genuine and moved on to the last stop of the domain lookup. To see the entire process on paper, refer to figure 7.

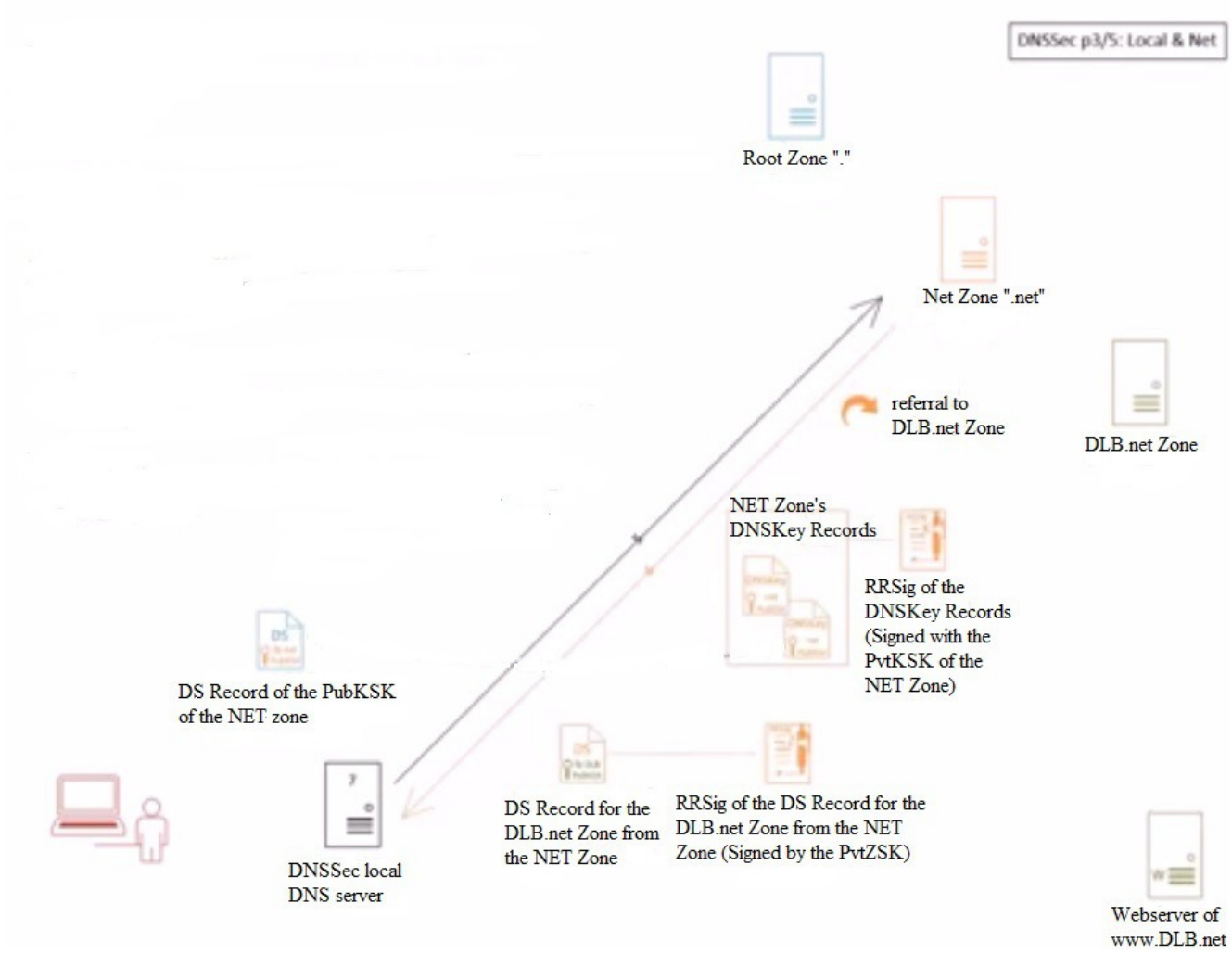

Figure 7. Example of DNSSec resolution from local to tld . Reprinted from www.daniellbenway.net, in Daniel L Benway, 2016., Retrieved September 18, 2017 from https://daniellbenway.net/DNSSec-explained/. Reprinted with permission.

At the final destination, the recursive DNS server is given the DNSKey records, as well as their digital signature (encrypted by the private KSK), the RRset of A records and equivalent digital signatures (encrypted by the private ZSK) from the authoritative domain (Benway, 2016). Assuming we're asking the server for the IPv4 address of a the domain. The RRsigs are decrypted for their digests

using their separate public keys and their identical RRsets are hashed. If each pair of digests are compared and they match up successfully, then the authoritative domain and its internal records are considered to be trustworthy. The recursive nameserver will then take the answer for the desired domain and store the entry in its local cache file (Benway, 2016). Refer to figure 8 to see this in more detail.

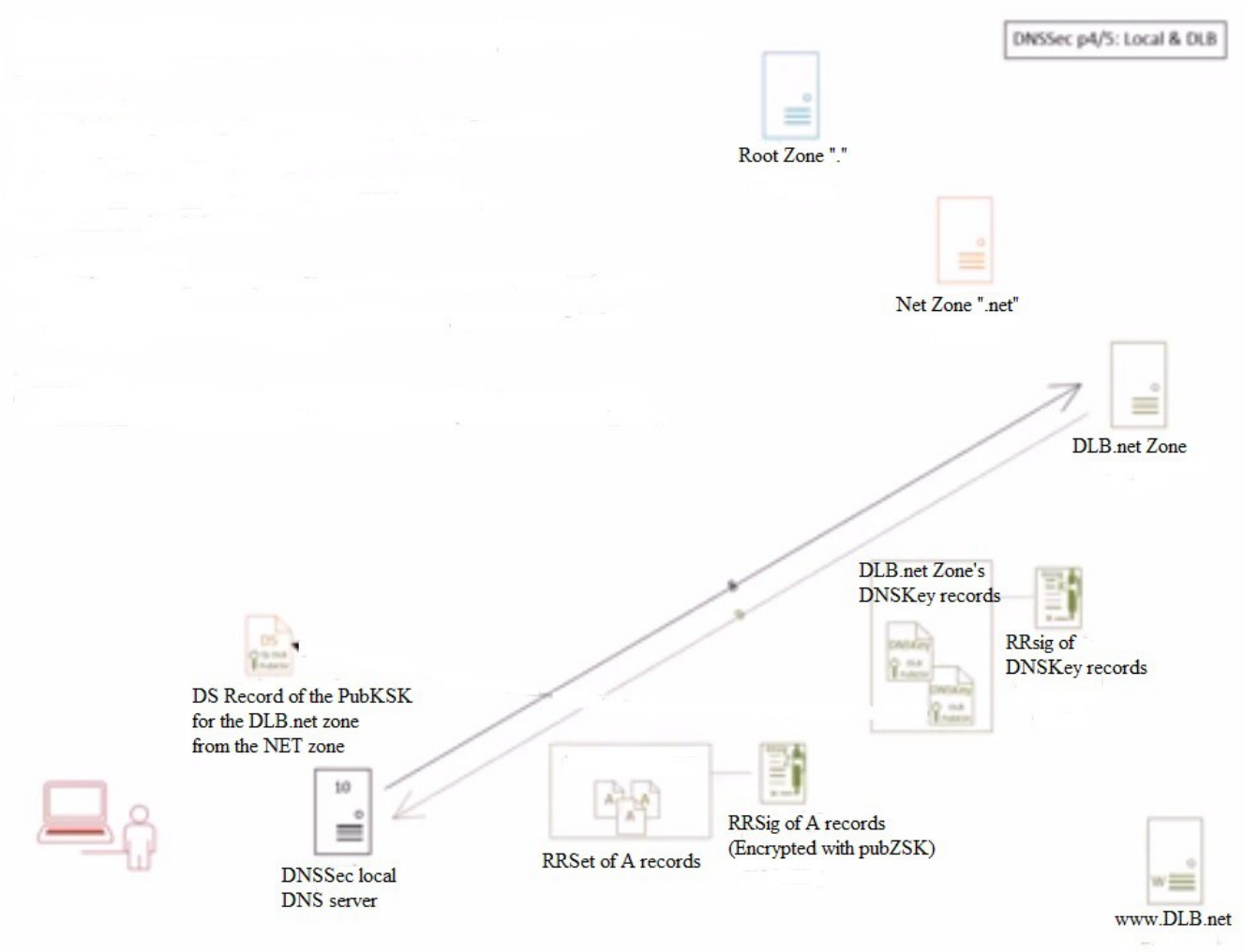

Figure 9. Example of DNSSec resolution from local to dlb. Reprinted from www.daniellbenway.net, in Daniel L Benway, 2016., Retrieved September 18, 2017 from https://daniellbenway.net/DNSSec-explained/. Reprinted with permission.

Before we had DNSSec, when a domain request was sent out on the Internet, the local DNS

server would take whatever answer that it was given as being the correct IP address. This allowed the DNS protocol to be taken advantage of for nefarious purposes. There were countermeasures created to make it increasingly harder to trick the gullible Domain Name System. Although, these attempts never created a feeling of safety that this security measure had. Through the addition of DNSSec, the Cyber Security community has created an anchor of trust. The symbolic "anchor of trust" is based on the fact that as long as each domain zone trusts the domain zone above them, then any information shared has to be absolutely correct. This methodology only works by enabling trust into the root domain and having this trust carry itself throughout the entire Domain Name System. Overall, DNS has come a long way from its very vulnerable beginning, to its more secure present.

DANE

We can add an extra level of security to DNS by using the power of DANE, or DNS-based Authentication of Named Entities ("DNS-based Authentication of Named Entities", 2017). According to Wes Hardaker, a Cyber Security researcher from Parsons, DANE is supposed to solve the current TLS trust anchoring problems. TLS ensures that no one can eaves drop on an https connection by encrypting the communication as well as making sure the user is talking to the correct web server. They can provide this level of trust by using certificates. When a web browser connects to a website they will check the certificate to see if it has the right domain name, the correct IP address, and the certifcate itself is from a trusted Certificate of Authority (CA). If all of this information is true, then the web browsers treats it as the legitimate domain (Hardaker, 2014). It may seem like this system is fine on its own and there's no need to improve anything. But as Hardaker mentions, the TLS suffers from the "Too many CAs problem", a CA is the organization that authorizes theses certificates to prove a domain is who they claim to be. Most web browsers have about 1300 trusted certificates loaded by default, causing them to trust any certificate is sees as long as its from a trusted source. According to Hardaker,

there have been instances where CA's have been tricked into giving a certificate to the wrong business or group (Hardaker, 2014). If a CA such as GoDaddy, Verisign, or Symantec is coerced into giving a certificate to the wrong people, those groups could run their false copy of a domain with a valid certificate. Causing modern web browser to think this false website is legitimate. Mr. Hardaker mentions that DANE fixes this problem by acting as a trusted certificate, called a TLSA record. This new version of a certificate would only work for a domain that is DNSSec protected. How this works is when a DNSSec domain is being looked up in the DNS tree, the only entry that would be valid to a web browser would be the one that has a TLSA certificate pointing to that domain (Hardaker, 2014). The reason why I mentioned this technology was that part of my capstone involved seeing what would happen to a domain that had a DANE certificate when it came under attack by a MITM DNS spoofing attack according to the DANE web browser validator.

DNSSec & DANE Web Browser Validators

Unfortunately, modern web browsers are unable to validate DANE certificates or DNSSec domains. The companies that run these applications are not interested implementing these technology in their web browsers. Their reasoning behind this decision was that it would add more time to load the webpage for their users. At one time the company Mr. Hardaker worked for, Parsons, had started developing a new web browser called Bloodhound designed to validate a domain that had a DANE certificate and was DNSSec signed (Hardaker, 2014). Sadly the funding for the project had run dry causing it to come to a complete stop (Hardaker, 2017). But, there is another way to check if a domain is DNSSec or DANE enabled and that is through web browser plugins from dnssec-validator.cz (“DNSSEC/TLSA Validator”). An interesting note about these plugins, Wes Hardaker mentions that they are not the most reliable as they don't check the TLSA record of an image or javascript url (Hardaker, 2014). Even though this was something to keep in mind, it didn't really affect my project as

I wanted to see how these validators stood up to DNS Spoofing attacks.

Project Setup

When it came to setting up the project, I needed a few components: a list of DNSSec signed domains, a few DNS Spoofing tools, and finally web browsers with DNSSEC validator plugins. Finding domains that were DNSSec signed was very difficult for me when I tried searching for them on the Internet. Oddly enough, I found them by sheer coincidence either by surfing the web for fun or research. The DNSSec websites I used were the following: raspberrypi.org, deelen-it.nl, dnssec-validator.cz, joscor.com, and dawiweb.com. The DNS Spoofing tools used were Dnsspoofing and Ettercap. According to dnssec-validator.cz, there are only a select number of web browsers to install these plugins on such as Google Chrome, Mozilla Firefox, Internet Explorer 8 through 11 32-bit, Apple Safari, and Opera ("DNSSEC/TLSA Validator"). The only web browsers I was able to use were Firefox, Chrome, and Opera. I tried to setup a 32-bit Windows 7 virtualbox VM, but for whatever reason I couldn't figure out how to install the DNSSec plugins for it. I did the same thing for Safari, but I found out you need to have specific hardware in order to install a MAC OS on virtualbox and I didn't feel like buying a $200+ MAC book for a capstone project, so I choose not to tinker with Safari for this project. Once I had gathered all of these necessary items, I was ready to start testing the reliability of the DNSSec and DANE web browser plugin.

Testing Phase

The way I went about performing my project, I would use either DNS Spoofing tool, Ettercap or Dnsspoofing, on one of the three web browsers that had the necessary plugins installed and visit every DNSSec signed domain to see how the plugins would react to my attacks. Given I had two

pentesting tools to choose from, three web browsers with the necessary plugins installed, and five DNSSec domains to choose from, I had about 30 trials for this entire project. For each trial, I would first clear the cache history of the web browser I was using and my Windows 10 laptop. Next, the apache2 service, hosting my spoofed website, would be started followed by the Ettercap or Dnsspoofing program. Once both of those programs had officially started, I would go to the web browser I was using and go to every domain from my list of DNSSec websites. I performed atleast five separate attempts for every trial to see how the domain and validator reacted from DNS Spoofing attacks.

Results

The outcome of this project came from the following: how the validators reacted from the DNS Spoofing attacks, the reliability of the DNS Spoofing programs being used, and the network connection of the web browsers being tested on. When the testing first began, I thought there would be mixed reactions from these validators as I went from one web browser to the other. On the contrary, I found that the validators behaved the same way on every web browser using either DNS Spoof program. Some of the issues I came across while running these tests, were how effective Dnsspoofing and Ettercap was when it came to spoofing DNSSec domains. The first program, Dnsspoofing, proved to be very ineffective throughout the entire capstone. Ettercap, on the otherhand, proved to be the absolute champ for majority of the project. Another problem that came to my attention was how effective DNS Spoofing was given the amount of bandwidth in certain locations. I recall giving a tutorial on DNS Spoofing during the Fall 2017 Open House for SUNY Polytechnic Institute in Kunsela Hall A129, a classroom. When I tried to run Ettercap during the Open House, I was unable to spoof a domain to resolve to my fake web server. Which made me wonder if it was the bandwidth available to my Kali VM. I later went back to my dorm room, an area where I had significantly more bandwidth available. I

ran the same test again using Ettercap and it worked just as normal. It made me realize that for a DNS spoofing attack to be successful using a Kali VM, it needed a strong connection on the network to do so.

Summary

At the conclusion of this capstone project, I had discovered that even if a domain is DNSSec signed, it can still be spoofed. After performing DNS Spoofing MITM attacks in my dorm room and Kunsela A129 from a Kali machine, these programs are the most effective when the virtual machine is on a network with a lot of bandwidth available. As far as the reliability of the web browser plugins, they reacted very quickly to a change in the IP address of the DNSSec domain when a DNS Spoofing attack occurred. In order to protect the every day user visiting a DNSSec signed website that is being altered by DNS Spoofing, they should be informed of how to interpret what the different changes from their plugins mean.

References

Benway, D. (2016, April 25). DNSSec explained. *Daniel L Benway's IT Blog*. Retrieved September 18, 2017 from https://daniellbenway.net/DNSSec-explained/

DNS-based Authentication of Named Entities. (2017, October 27) Retrieved November 14, 2017 from Wikipedia: https://en.wikipedia.org/wiki/DNS-based_Authentication_of_Named_Entities

DNSSEC/TLSA Validator. (n.d.). *CZ.Nic.* Retrieved November 15, 2017 from https://www.dnssec-validator.cz/

Friedl, S. (2008, August 7). An illustrative guide to the Kaminsky DNS vulnerability. *Unixwiz.* Retrieved September 13, 2017 from http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

Gonyea, C. (2010, August 25). DNS: Why it is important & how it works. Retrieved September 7, 2017 from https://dyn.com/blog/dns-why-its-important-how-it-works/

Hardaker, W. [Wes Hardaker]. (2017, May). Re: Tutorial on DANE and DNSSEC [Comment]. Retrieved November 14, 2017 from https://www.youtube.com/watchv=BhvU19RJrPY&t=1420s?

Hardaker, W. (Presenter) (2014, December 8). Tutorial on DANE and DNSSEC [Video file]. Retrieved November 14, 2017 from https://www.youtube.com/watch?v=BhvU19RJrPY&t=1420s

Palermo, E. & Cox, L. (2014, January 15). Who invent the Internet? *Live Science.* Retrieved September 9, 2017 from https://www.livescience.com/42604-who-invented-the-internet.html

Pham, T. (2016, April 26). The great DNS vulnerability of 2008. *Duo.* Retrieved September 13, 2017 from https://duo.com/blog/the-great-dns-vulnerability-of-2008-by-dan-kaminsky

Rader, R. (2001, April 25). History of DNS. *Soft Panorama*. Retrieved September 7, 2017 from http://www.softpanorama.org/DNS/history.shtml

Rouse, M. (2016, June). Asymmetric Cryptography (public key cryptography). *Tech Target : Search*

Security. Retrieved November 13, 2017 from http://searchsecurity.techtarget.com/definition/asymmetric-cryptography

Sanders, C. (2010, April 7). Understanding Man-In-The-Middle attacks – part 2: DNS Spoofing. *TechGenix.* Retrieved November 11, 2017 from http://techgenix.com/understanding-man-in-the-middle-attacks-arp-part2/