IT603

24 March 2014

SSL Attacks: Weaknesses in the Underlying Security Structure of the Internet

1. Abstract

TLS\SSL (Transport Layer Security/Secure Sockets Layer) are the encryption protocols that underpin the security of the Internet. They secure communications between clients and servers. They encrypt the back bone links. Modern commerce relies on the perception of security in Internet transactions. Business to business deals could not happen if either party didn't have complete trust in the security of the infrastructure over which they occur. With the news filled with various breaches of security and privacy, the average consumer is also becoming more conscious about how they conduct their day to day to affairs online (Jackson)

Unfortunately this back bone is under constant attack. Different groups have varying reasons for wanting to subvert your communications. Hackers just want to do because they can. Your school wants to do it to monitor your traffic. The RIAA wants make sure you aren't sharing. Your company doesn't want you selling secrets, and neither does your government. Your ISP wants to see if your viewing someone else's content. Each of the groups has a vested interest in monitoring your traffic. In this paper I'm going to discuss the mechanism of TLS/SSL, how it is suborned, and how you can protect yourself.

2. Background

The original plan for the Internet was much more focused on redundancy and survivability than it was on security or privacy. In the early 1990's the engineers at Netscape identified a growing need for security. Their answer was Secure Sockets Layer. SSL was released in 1994 by Netscape for their Navigator browser (IBM). The first version never made it

out into the wild and the second version had some major security flaws. It was supplanted fairly quickly by the third version, which became the most popular.

The SSL/TLS protocols occur between the Application Layer and the Transport Layer. This means any application that rides TCP/IP can make use of the technology. The fundamental idea is to use public key cryptography to exchange a shared key. Once both parties have an agreed upon key the rest of the communication takes place with symmetric encryption. This initial communication is known as the handshake.

To begin the process a client sends the Client Hello Message to the server it wants to have a secure conversation with. This hello message contains:

- The version of SSL/TLS that it wants to use.
- A 32 byte field, consisting of a 4 byte time and date and 28 bytes of random noise, which will be used later when the Master Secret is created.
- A list of all of the cipher suites that it can support.
- The compression algorithms that can be used (optional).

Once the server gets this information it will respond with the Server Hello Message. This message consists of the highest matching version number, its own 32 byte random number, a session ID (optional), the strongest cipher suite that both have in common, and the optional compression algorithm. Once all of the preliminaries are agreed upon the server can send its certificate[1] along with its public key. In some circumstances the server can also demand a client

[1] The certificates referred to are documents in the X.509 format that are issued by a trusted third party known as a Certificate Authority (CA). These are independent companies that are supposedly trusted and vouch for the identity of the entity applying for a certificate. The

certificate, though this is fairly rare. The server finishes with a Server Hello Done message. The client then uses both random numbers to create a pre-master secret, which it then sends to the server encrypted with the servers own public key. The fact that the server can decrypt this message assures the client that the server is who it says it is. Now that a symmetric key has been agreed upon and the parameters of the conversation have been set the client sends back a Change Cipher Spec message, letting the server know that all future conversations will occur with this shared secret symmetric key. After this it also sends the Client Finished message. The server responds with a Change Cipher Spec and Server Finished message of its own. The handshake is now complete and the rest of the conversation takes place using the faster symmetric key technology.

---

certificates contain the registrant's public key and are signed by the CA's private key. In theory these companies are supposed to verify the applicants' identity but is a poorly regulated industry. In practice there are a few top level CA's that have gained the public's trust over time. For further reference read this page.

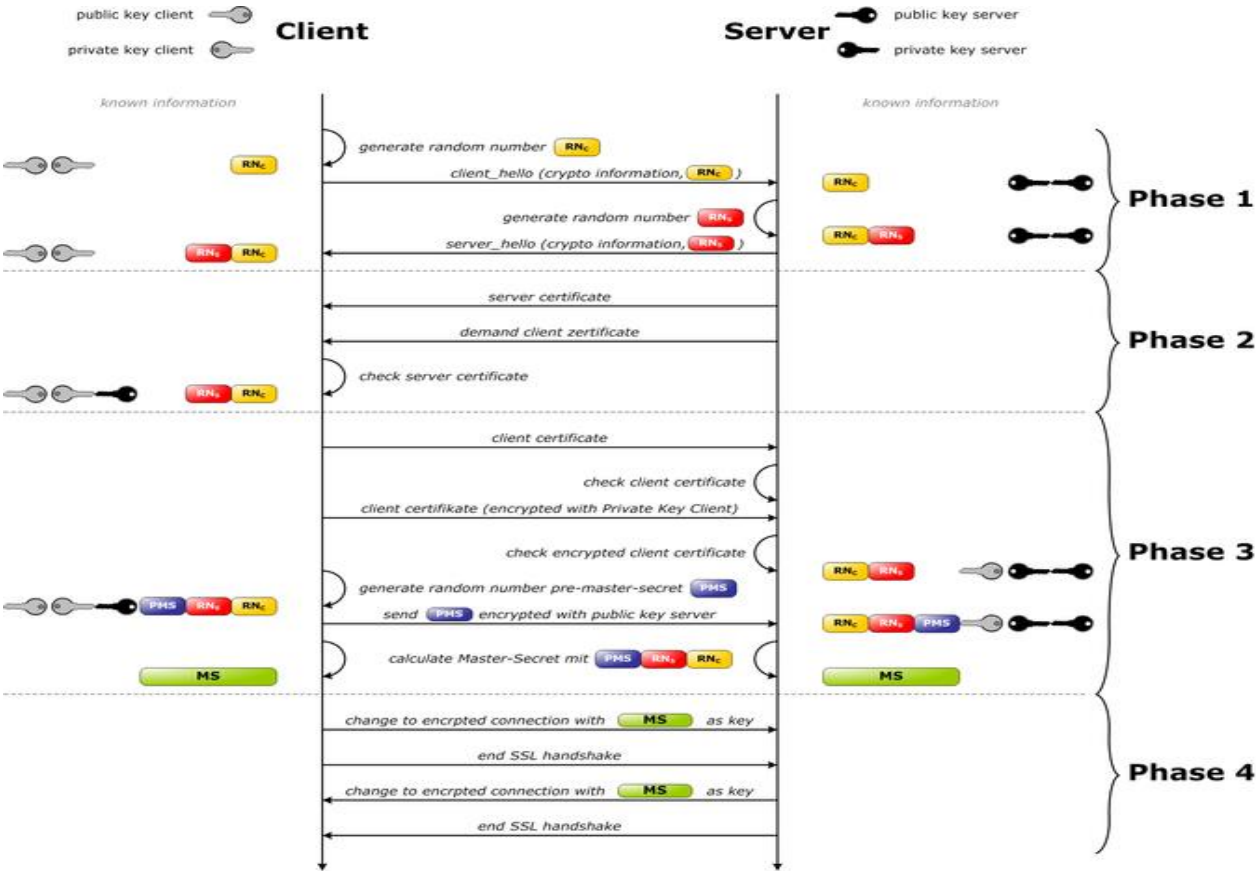

Figure 1. Friedrich, Christian. SSL Handshake with Certificates (Friedrich) licensed under CC 3.0

3. Some Known Man in the Middle Exploits

Because this protocol is central to everything sensitive we do on the Internet; from online banking to VPN's, it is a natural target for thieves and snoops. Over the years there have been many attacks on SSL/TLS. These attacks have ranged from the relatively minor roll back attack through the cooler named BEAST attack up to the more nefarious Man in the Middle vulnerability.

The encryption used for SSL is good. The math is proven and we've got that pretty well figured out, at least until quantum computing[2] becomes more practical. The problems come in during implementation. It's once you get away from pure mathematics and insert the human factor that problems arise.

One of the first major attack vectors showed up in the fall of 2009. This method, known as a **renegotiation attack**, required an attacker to have already established a man-in-the-middle situation on the gateway. The key with this attack is that the spec allows for either side to renegotiate the cipher-link at any time. If, for instance, you wanted to connect to part of a website that required additional authentication the server could request a certificate from the client to authenticate them. This would require a new handshake, this time with the server sending a *certificate request* message to the client after it forwards its own certificate. In theory this transaction should be encrypted, since it occurs after the *change cipher spec* message. Since all traffic after that message is encrypted, using the agreed upon symmetric key, it shouldn't require any new security. The problem occurs when you're in a situation where someone has

[2] Quantum computing is the probable next leap in computing power. Instead of using bits, 1's or 0's, a quantum computer uses qubits, 1's, 0's, or 1 and 0 at the same time. Because of this ability to hold superposition a quantum computer is much more parallel than a traditional computer. This parallel nature has profound implications for encryption. A quantum computer with enough qubits can factor large primes at a rate high enough to make the cracking of RSA style public-key cryptography practicable. By using Shor's algorithm a large prime can be factored in a fraction of the time it takes to do it using the traditional *general number field sieve* (Gerjuoy).

already established themselves as a man-in-the-middle, via attacking the ARP tables[3] or any of a number of other methods. Once a malicious user has insinuated themselves into the network like this they can begin to play with peoples supposedly secure connections. If the attacker sees one of their victims trying to initiate an SSL connection they can halt that packet while they send their own SSL request to the server. Once the bad guy has a secure connection with the server they can send a partial communication to the server and then end it with the client's original *client_hello* message. This makes the server think it's renegotiating an existing connection, but the client thinks it's starting a new connection. The bad guy can't read the newly encrypted channel, but he can now impersonate the client. Any traffic from the attacker is treated the same as traffic from the real client, therefore the server will believe that whatever traffic the attacker sent in that unfinished message came from the authenticated client. This doesn't sound all that useful but it can be parlayed into bigger things.

Due to the man-in-the-middle nature of the attack it will work best in an open environment, such as a public Wi-Fi hotspot or a business that offers free Internet to its guests. So for an example we have Bad Guy Bill, who works at the Sleep-no-Mor motel. As anyone who has worked a nightshift can attest to it sometimes gets a little boring, so B.G. Bill likes to plug his laptop into the main guest Ethernet switch for the motel. Using any one of several programs

[3] Because of the nature of IP addresses running over Ethernet it is possible for an attacker to substitute their MAC address in a client/servers ARP tables, allowing them to pretend to be the gateway or server to the client and the client to the gateway or server. This allows them to view and/or modify all traffic between the two.

designed for it[4] he insinuates himself as a man-in-the-middle for the motels guests. Along comes Innocent Irene, in room 42, who wants to surf to her Amazon Video account and watch some Fraggle Rock. She types the URL for Amazon into her web browser, surfs to Amazon Prime video and gets ready for some Muppets. In the background her browser needs to switch from HTTP to HTTPS to authenticate her as having rights to watch that video, so it sends out a TLS *client_hello* message on port 443. Meanwhile B.G. Bill is in the network closet reading all of the traffic into and out of the hotel. He sees the request to Amazon going out on 443 so he knows it has something to do with SSL. He captures the packet and sees that that the person in room 42 is trying to establish a secure connection with an outside server. He temporarily blocks that packet from going through and establishes his own secure connection to the server. Since it's going out to Amazon Video he sends a *get* request for a movie he wants to watch but leaves the final part of the request unfinished. Where the carriage return command to finish the header would go he inserts Ms. Irene's original plain text *client_hello* message into his previously encrypted stream. The server sees this as a renegotiate request since it's coming in on a secure channel, like it's supposed to and sends its *server_hello* message back to the original client. Ms. Irene's browser then finished the handshake procedure to establish an SSL connection. Once her browser thinks it has a secure connection to the server it sends its valid get request for Fraggle Rock, appending the valid authentication cookies that had previously been setup when Ms. Irene checked the box for stay logged in back when she was at home on a secure network. To the server this is all one

[4] Man in the Middle proxy, Ettercap, Cain & Abel, DSniff, or Driftnet to name a few.

long request, since it all happened over the original secure connection, and it authenticates both Ms. Irene's and Mr. Bill's movie requests.

The attacks not super dangerous because it doesn't break any encryption but it can give an attacker the same rights as a proper user on a network. The initial attack is pretty easy but any serious exploit requires some knowledge.

The easiest work around is to simply require servers not to allow renegotiations. This is obviously temporary and requires something else to make a permanent fix. The main difference between TLS 1.0 and SSL 3.0 is the addition of extensions, which can be added to the *client_hello* packet. By making a new extension for renegotiations and placing in it some information about the previous session, that only the real client and real server would know, a malicious renegotiation would fail. So in our example the server would refuse any renegotiation that didn't include some information from the original handshake. In this case the original handshake actually came from the bad guy, so when Ms. Irene's *client_hello* packet arrives without the proper value in the secure renegotiation field the server will reject it.

The IETF addressed this problem in RFC 5746 (Rescorla, Ray and Dispensa). This RFC defines a new extension called 'renegotiation_info'. This extension should contain a null value for a first time connection, the client_verify_data (the verify data from the last finished message in the last handshake) when the request is started by the client, or the server_verify_data and client_verify_data if the request is initiated by the server. In order for this to work it must be implemented in such a way as to require a value in this extension, otherwise the non-requesting party should immediately terminate the connection.

One of the newest ways to attack SSL/TLS connections is to spoof the certificates and Certificate Authorities that proper authentication relies on. The security of SSL/TLS relies on

public key cryptography for the initial key exchange. Public key cryptography in turn relies on trusted Certificate Authorities. If you can subvert the CA's, you can subvert the whole process.

Early this year a company called Netcraft discovered that the Internet was being flooded by fake certificates. The certificates look just like the real ones and use the legitimate common names of their targets. What they lack is a signature from a valid Certificate Authority, they are self-signed. Normally these wouldn't pass muster in any full-fledged browser, all of the popular ones check for this signature during the initial handshake and validate it against the stored hash of what it should be. The twist here is that more and more secure traffic is going through mobile phone apps, while traditional browsers don't see that much traffic anymore. How often do you do your banking from your smartphone these days? Because of their limited size, and the limited processing power of their hosts, a lot of smart phone apps don't have the horse power to actually validate those signatures. This leaves these apps open to *man in the middle* attacks. Much like the attack I described earlier, this vulnerability can only be exploited by someone in a position to listen (and insinuate themselves into) to your conversation, whether that be in an open Wi-Fi situation or somewhere further upstream. Your mobile app makes a connection to your bank. It gets back a certificate that says it's from your bank. Therefore it is secure, as far as the app is concerned. It never actually checked to see if that certificate was actually from your bank, it just assumed it was. A nefarious person can use this fact to intercept your communications; feeding you the fake certificate, while maintaining a real connection to the bank. According to researchers at IOActive 40% of banking apps made for iOS don't bother validating certificates (Sanchez). Another group of researchers at Stanford and the University of Texas found that many of the back end processes for the more popular cloud services don't validate either. Such things as Amazons java library, along with theirs and PayPal's merchant SDK's trust whatever

certificate is shown to them (Georgiev, Anubhai and Iyengar). With Internet enabled apps becoming popular on our more traditional platforms this problem is creeping into them also. Valve Software's Steam engine was allowing PayPal connections to be un-validated for three months before they fixed it (Mutton).

To give an example using our established characters we will return to the Sleep-No-Mor motel. B.G. Bill got tired of always sneaking into the wiring closet so he plugged in an access point and gave it an SSID that sounded official (The motel, being kind of cheap, doesn't actually offer Wi-Fi, only wired connections). When I. Irene checked in he noticed that she used a credit card from the First Bank of Sealand. In anticipation of some new shenanigans he generated a self-signed certificate with their name and has set up a redirect that sends any traffic to their IP address to his computer instead. Meanwhile, in room 42, Ms. Irene wants to get on her phone and check her balance; tomorrows going to be an exciting day of sightseeing after all. The app on her phone makes a connection to the IP address programmed into it, which it thinks is the server for her bank. Because of the redirect, however, that IP address now resolves to B.G. Bill's webserver. When the app sends it *client_hello* message to initiate an SSL connection the fake server replies with its faked SSL certificate. Since Ms. Irene's app 'knows' that the server at the IP address it connects to is the banks, and it even says it is right there in the returned certificate, it must be valid. So now Ms. Irene's phone has a properly encrypted, but not properly authenticated, connection to her 'bank'. B.G. Bill's server then sets up a channel to her real bank, creating a classic Man-in-the-Middle situation. When Ms. Irene's app sends her login information to the bank it is decrypted at B.G. Bills server, before being re-encrypted and sent on to the bank.

Unfortunately there's not much you can do on the client side to mitigate this attack. Developers and server maintainers must fix their systems so that they validate certificates. If that is not possible because of resources, the app developer can add certificate pinning to their app. Instead of the app verifying against a chain of certificate authorities, it verifies against some information stored in the app; such as the true certificates signature (Walton, Steven and Manico).

In the legal, but shady, department we have HTTPS Proxy Devices. These are (mostly) hardware solutions that can perform man-in-the-middle attacks, usually for law enforcement or corporate clients. Many corporations, schools, and government agencies have a vested interest in monitoring your web traffic. Your university might want to make sure you're not using their network to illegally download files. Read your corporate IT policy, there's probably a clause about monitoring your traffic. They don't want you looking at inappropriate sites, or emailing a client list to your buddy that works at a different company.

The problem (for them) is that in recent years, especially with the Snowden revelations, more and more content providers are going to SSL/TLS and enforcing its use for all connections to their sites. This development is putting a big dent into those interests' voyeuristic pastimes. There's even a name for it; The Going Dark Problem (Caproni). Their answer to this 'problem' is the aforementioned HTTPS Proxy devices. When you get your shiny new corporate laptop it's quite possible that it has an equally brand new Certificate Authority in its trusted list that shouldn't be there, installed by your corporations IT department. When you connect to an outside site it goes through the HTTPS proxy that was also installed by IT. That proxy device intercepts your communication and establishes an SSL connection back to you signed with the certificate they planted in your CA store and named after the site you're trying to visit. To keep you none

the wiser it the device forwards your traffic on to the intended server, encrypted with the server's certificate. Since browsers are inherently promiscuous when it comes to Certificate Authorities there is no warning to the user that something's fishy (Smith).

Several companies have been caught doing this in the wild. One of the first was Nokia, who was caught intercepting and decrypting cell phone user's traffic as it passed through their network. Ostensibly this was so they could compress your data and save you some money on your cell phone data fees. While this is undoubtedly true their privacy policy is at odds with this practice (Whittaker). Anecdotally I have heard several college students complain that their universities must be doing this also since they have gotten warnings about content that the school couldn't have known about unless they were seeing the decrypted traffic.

One of the few, if not only, ways to detect this type of interception is too look at the certificates 'fingerprint', which is derived from the remote sites private key (the appliance must use a different key pair since it can't know the proper private key). This method isn't foolproof as there is no real way to know what Certificate Authority the site you're visiting is supposed to be using, all you can really do is check that the fingerprint of the certificate you're getting matches the fingerprint of the certificate the sites sending to other parties (Horowitz). Towards this end Steve Gibson of Gibson Research has developed a tool that will allow you to check the fingerprint you're getting against the one you should probably be getting (Gibson).

4. Mitigation Trends

One of the technologies that's been around for a while (before SSL even (Duncan)) is called Perfect Forward Secrecy (PFS). This protocol ensures that even if a servers private key is compromised any communications, outside the compromised one, stay encrypted.

If you'll harken back to our original discussion about the SSL handshake you may remember that your connection is encrypted using a key derived from partially from the server's private key. In traditional SSL encryption an attacker can use the server's private key to unencrypt all traffic secured with that certificate. So if someone were to vacuum up a lot of your encrypted traffic they could hold on to it until some future date when they have obtained the key and go back and decrypt all of your past messages[5]. If you enable PFS you force the client and server to use new, and completely un-linked to any other session, session keys. This means that even if the servers private key is compromised your old messages will remain secure.

The other big trend is HSTS, or HTTPS Strict Transport Security. The problem that HSTS addresses is that there is no way for your browser to know which sites should be encrypted and which ones are less important. As far as your browser knows there is no difference between your bank and a recipe site. In steps HSTS. This standard allows web sites to send a message back to your client in an HTTPS header that tells you the site should be encrypted and not to connect to it unless it is. As of now all browsers support it except IE, which has promised to add it to the next release (Gillula). Unfortunately very few servers have implemented it yet for some reason.

5. Conclusion

[5] The NSA for instance.

While SSL/TLS remains our best bet for securing communications while they are en route to their destination, human ingenuity, and laziness, have greatly hampered that security. The underlying mechanism is mostly secure. The math that is done to do the actual encryption is solid. Implementation leaves a lot to be desired however. PFS might give you a little more overhead, but I think it would be worth it to a web site maintainer. There is no reason, beyond ignorance or laziness, not to enable HSTS. Why more developers don't use them is beyond me. With the recent announcement of the Heartbleed bug maybe more content providers will get on board. Both PFS and HSTS would have mitigated some of the damage that might have been done.

Works Cited

AV Comparatives. *IT Security Survey 2014*. Survey Results. Innsbruck: AV-Comparatives, 2014. Document on Web Site. 24 March 2014. <http://www.av-comparatives.org/wp-content/uploads/2014/03/security_survey2014_en.pdf>.

Bamford, James. "The NSA Is Building the Country's Biggest Spy Center (Watch What You Say)." 15 March 2012. *Wired.* Web Site. 10 April 2014.

Bonsor, Kevin and Jonathon Strickland. "How Quantum Computers Work." 8 December 2000. *HowStuffWorks.com.* Web. 7 April 2014.

Boyle, Randall J and Raymond R Panko. "SSL/TLS." Boyle, Randall J and Raymond R Panko. *Corporate Computer Security: 3rd Edition*. Upper Saddle River: Pearson Education, 2013. 173-180. Text Book.

Caproni, Valerie. "Going Dark: Lawful Electronic Surveillance in the Face of New Technologies." *Statement Before the House Judiciary Committee, Subcommittee on Crime, Terrorism, and Homeland Security*. Washington, D.C., 17 February 2011. Web.

Chirgwin, Richard. "Banking apps: insecure and badly written, say researchers." 13 January 2014. *The Register.* Web. 9 April 2014.

Ciampa, Mark. "Cryptographic Transport Protocols." Ciampa, Mark. *Security+ Guide to Network Security Fundamentals: Third Edition*. Boston: Course Technology, 2009. 421-429. Book.

Crall, Chris, Mike Danseglio and David Mowers. "SSL/TLS in Windows Server 2003." 31 July 2003. *Microsoft Technet.* White Paper. 26 March 2014. <http://technet.microsoft.com/en-us/library/cc781800(v=ws.10).aspx>.

Duncan, Robert. "SSL: Intercepted today, decrypted tomorrow." 25 June 2013. *Netcraft.* Web Site. 10 April 2014.

EKR. "Understanding the TLS Renegotiation Attack." 5 November 2009. *Educated Guesswork.* Web. 8 April 2014.

Friedrich, Christian. *Schematic representation of the SSL handshake protocol with two way authentication with certificates.* Illustration.

Georgiev, Martin, et al. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software." 2012. *University of Texas.* Web. 9 April 2014.

Gerjuoy, Edward. "Shor's Factoring Algorithm and Modern Cryptography. An Illustration of the Capabilities Inherent in Quantum Computers." *arXiv:quant-ph/0411184* (2004). Web.

Gibson, Steve. "Fingerprint: Is your employer, school, or Internet provider eavesdropping on your secure connections?" 2013. *Gibson Research Corporation.* Web. 10 April 2014. <https://www.grc.com/fingerprints.htm>.

Gibson, Steve. *Security Now!: #443 Sisyphus* Leo Laporte. 18 February 2014. Podcast.

Gibson, Steve. *Security Now!: Episode #223, A Security Vulnerbility in SSL* Leo Laporte. 19 November 2009. Podcast.

Gillula, Jeremy. "Websites Must Use HSTS in Order to Be Secure." 4 April 2014. *EFF.org.* Web Site. 10 April 2014.

Hodges, J, et al. "RFC 6797: HTTP Strict Transport Security (HSTS)." RFC. 2012. Web.

Horowitz, Michael. "Defensive Computing: Steve Gibson's Fingerprint service detects SSL man in the middle spying." 14 April 2013. *Computer World.* Website. 10 April 2014.

IBM. "History of SSL." n.d. *iSeries Information Center.* Web. 26 March 2014.

Jackson, William. "Do the security conscious see something we don't?" 28 March 2013. *GCN.* Cybereye Blog. 24 March 2014.

Jarmoc, Jeff. *SSL/TLS Interception Proxies and Transitive Trust*. White Paper. Amsterdam: Dell SecureWorks Counter Threat Unit℠ Threat Intelligence, 2012. Website. <https://media.blackhat.com/bh-eu-12/Jarmoc/bh-eu-12-Jarmoc-SSL_TLS_Interception-WP.pdf>.

Laporte, Leo and Steve Gibson. "Episode 243: State Subversion of SSL." *Security Now!* 8 April 2010. Podcast. 27 March 2014. <https://www.grc.com/sn/sn-243.htm>.

Microsoft. "PKI Technologies." 28 March 2003. *Microsoft Technet.* Web. 27 March 2014. <http://technet.microsoft.com/en-us/library/cc737264(v=ws.10).aspx>.

Mutton, Paul. "Fake SSL certificates deployed across the internet." 12 February 2014. *Netstat.* Web. 9 April 2014.

Pfleeger, Charles and Shari Lawrence Pfleeger. "Network Security Controls." Pfleeger, Charles and Shari Lawrence Pfleeger. *Security in Computing: Third Edition*. Upper Saddle River: Prentice Hall Professional Technical Reference, 2003. 439-440. Book.

Rescorla, E, et al. "RFC 5746: Transport Layer Security (TLS) Renegotiation Indication Extension." RFC. 2010. Web. <https://tools.ietf.org/html/rfc5746>.

Sanchez, Ariel. "Personal banking apps leak info through phone." 8 January 2014. *IOActive.* Web. 9 April 2014.

Sidhpurwalahuzaifa. "Transport Layer Security." 24 July 2013. *Red Hat: Security Blog.* Web. 27 March 2014. <http://securityblog.redhat.com/2013/07/>.

Singel, Ryan. "Law Enforcement Appliance Subverts SSL." 24 March 2010. *Wired.* Website. 10 April 2014.

Smith, Ms. "Certified Lies: Big Brother In Your Browser." 23 July 2010. *Network World; Privacy and Security Fanatic.* Web site. 10 April 2014.

Tyson, Jeff. "How Encryption Works." 5 April 2001. *How Stuff Works.* Web. 27 March 2014. <http://computer.howstuffworks.com/encryption.htm>.

Wagner, David and Bruce Schneier. "Analysis of the SSL 3.0 Protocol." *The Second USENIX Workshop on Electronic Commerce Proceedings*. 1996. Web.

Walton, Jeffrey, et al. "Certificate and Public Key Pinning." 18 March 2014. *The Open Web Application Security Project.* Web Page. 9 April 2014.

Whittaker, Zack. "Nokia 'hijacks' mobile browser traffic, decrypts HTTPS data." 10 January 2013. *ZD Net.* Web Site. 10 April 2014.

Zhu, Yan. "Why the Web Needs Perfect Forward Secrecy More Than Ever." 8 April 2014. *EFF.org.* Web Site. 10 April 2014.