Group 11 Implementation Write-up

Coen Valk, Shizhi Gu, Yu Wang

December 3, 2018

1 Introduction

Secure Sockets Layer (SSL) is a protocol for establishing a secure connection between two computers over socket connection. Over the years the SSL standard has improved to ensure secure channels between computers. Most people trust that information on the internet is secure, unfortunately nothing could be farther from the truth. Having a secure suite of encryption methods is essential to keeping personal messages and information private. In this paper we will learn more about the intricacies and difficulties associated with creating a version of SSL communication between computers.

Since the emergence of telecommunication, creating secure standards for communication has played an important role in research concerning the broader topic. many versions of SSL and Transport Layer Security (TLS), have been implemented, to address the constantly changing field of cryptography and cryptographic attacks. Now more than ever, It is essential that SSL and TLS standards are kept certifiably secure in order to promise security and privacy while sharing data over the internet.

2 Implementation

This project is written using Python 3. Its high-level convenience as well as its error checking and security against overflow attacks make it a clear choice. Additionally, Python has many libraries for mathematics which make it even more convenient to write fast yet secure implementations. Of course, Python also has its drawbacks. As a high level interpreted language, it is not quite as fast or efficient as some other lower level programming languages such as C. Because of this, the server we have written might not be able to handle as large of a bandwidth of traffic compared to if we were to have implemented this project in C. However, because the purposes of this assignment is creating a secure implementation of SSL over having a fast responding server. While C is a faster language, C is more vulnerable to overflow attacks and attacks on memory.

In order to communicate, the project consists of one client and one master server which will facilitate key distribution between clients so they can communicate safely together. Every full successful connection between client and server is divided into 4 parts: Negotiation of encryption method to use and authorizing both side, Distribution of public, private, or symmetric keys, communication, and disconnection from the server. All communication between client and server is done with formatted packages with necessary information to send and receive messages and keys. Fig. 1 depicts the traffic and packages sent between the client and server for successful communication between agents.

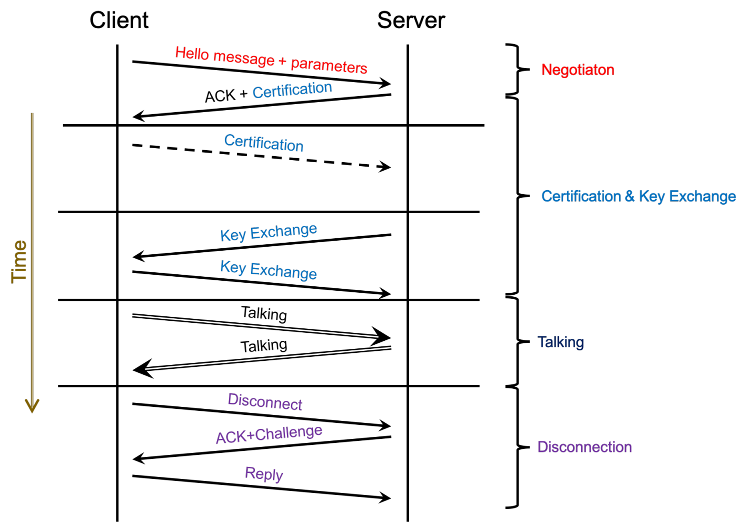

Figure 1: Normal Traffic Between The Server and Client

2.1 Negotiation

Negotiation is the first step to selecting a secure server connection between agents. This specific Negotiation implementation is based on the SSL handshake protocol. The goal of the negotiation phase is to correctly certify that agents are talking properly, and the most secure available cryptosystem is used to communicate. The client sends information on itself to the server, as well as the supported ciphersuite available to the server, in order of preference. The server responds with the chosen cryptosystem and begins the process of distributing the necessary keys and certifications.

The negotiation phase is crucial as it creates the framework that will be used to communicate between client and server. In this project we adopt 3 cryptosystems: the Data Encryption Standard (DES), Rivest-Shamir-Adleman (RSA), and Blum-Goldwasser.

DES, while when used alone is not considered secure by any means, when used three times in a row with multiple keys does become certifiably secure. This form of DES is no longer vulnerable to meet in the middle attacks, as is the case with double DES. However, while it is decently secure, the more messages are sent through DES, the more information is leaked about the three keys. This can be taken advantage of in a differential cryptographic attack, which analyses the change from plaintext to ciphertext with respect to the S-boxes to derive information about the keys. Because of this, DES will be rather low on the list of preferred ciphersuites, and will most likely only be used if it is the only option available to us.

As an encryption method, RSA has made a name for itself as a common standard cryptosystem across the internet. Its simplicity paired and efficiency paired with its relatively strong security makes it a popular choice for many cryptographic projects. While very useful, RSA is not completely infallible. Recent developments in hardware speed have pushed the required size of the private and public key to become larger and larger to be secure against a brute force attack. 32 and 64 bit keys have now become extremely susceptible to brute force attacks, so 128 and 256 bit keys have now been required to ensure the security of RSA. Furthermore, base RSA is not semantically secure, making it vulnerable to chosen plaintext attacks. Fortunately, RSA can be made semantically secure using random padding. For the purposes of this assignment, we will be padding RSA randomly to gain this extra security. However, in encryption setups involving a free decryption oracle, even semantically secure cryptosystems can be broken with a chosen ciphertext attack.

Also, RSA is not semantically secure out of the box. While it is relatively simple to randomly pad encryption, it is typically rather inefficient to pad a message securely. Blum-Goldwasser is a cryptosystem that offers nearly the same efficiency as RSA, but is semantically secure. As a probabilistic encryption algorithm, when an message is encrypted multiple times, based on a random seed, the ciphertext will look different every time. Despite this, these ciphertexts will still decrypt deterministically to the original plaintext, if certain conditions are met for the private key primes p and q. Both of these primes should be blum integers, meaning that they are equivalent to 3 mod 4.

Need to point out that, for Blum-Goldwaser encryption algorithm, the decryption result may not necessarily be the same as the original plaintext because of the zero padding. In our case, we force the last bit of the plaintext to be "1" and get rid of all the "0"s from back.

In the negotiation process, another issue to consider is the authentification. The client and the server must send each other their valid certifiction with signature to prove that "the person you are talking to is the one you want to talk". So in the negotiation package, user's public key is also included.

2.2 Key Exchange

After a cryptosystem has been chosen, the next step is to determine the public and private keys necessary to begin communication between the server and client. A popular method for creating public and private keys is using Diffie-Hellman key exchange. Figure 2 represents how Diffie-Hellman can create a private key without either of the parties making sending the private key in the first place. This method is quite efficient, however there are still many vulnerabilities in the base Diffie-Hellman key exchange. Attacks including man in the middle and replay attacks need to be addressed before a key exchange protocol can be considered safe. In this project, we will be making use of machine authentication codes (MAC's) and nonce's to certify the client and server's identity, as well as nonce's to prevent replay attacks.

If a package is found to be fraudulent, the user will be punished based on the severity of the offense. If too many offenses have been found, the server will disconnect with the client and experience a cooldown time.

As we are making use of multiple cryptosystems, the Key Exchange algorithm has to be broad enough to send packages with symmetric keys like that of DES, up to multiple coefficients and private and public keys for ECC. To do this, packages will be sent as concatenations of multiple keys together at once, and parsed to be multiple keys again when received by the other computer.

2.3 Communication

Communication between computers after negotiation and key exchange is the longest portion of the process. Because of that it is also the part of the process that is the most vulnerable to attack. The more and more an eavesdropper records messages sent between clients, the more information is leaked about private key information. Furthermore, adversaries can make use of chosen ciphertext and plaintext attacks as well as man in the middle attacks to spoof someone's identity. As with other packages, MAC's and nonce's are used to certify the identity of users and certify that there are no attacks possible to imitate another person's identity.

In communication, packages are going to be encrypted using the cryptosystem that was chosen in the negotiation phase of the process. In addition to this, the message is also hashed to ensure that original message has not been tampered with. For this project, we are making use of the SHA-1 hashing algorithm. Nowadays, SHA-1 is considered to be insecure based on the power of current computer hardware. A birthday attack can find a collision between a message and changed message that happen to have the same hash relatively quickly.

2.4 Disconnection

When communication is over, it is also important to close all socket connections between user and client and ensure that all communication is ended securely. One large concern was the possibility of de-authentication attacks, which could have an adversary hijack another users key information. In order to prevent an attack such as this, when a request is made to the server to disconnect, the server responds with a challenge to authenticate the client's identity. The client must respond correctly to the challenge to successfully close connection. If this does not happen, we can conclude that this disconnection request was fraudulent, and the disconnection request is ignored.

2.4.1 Challeng Strategy

To verfy the disconnection request is proposed by the actual user, the user who receives the request (not necessarily the server) sends back a challenge. The challenge is a list of 3 integers $[x_1, x_2, x_3]$. The requester randomly choses one of the integer and signs it. For the other 2 unchosen integers, they are replaced with random numbers uniformly chosen from the outcome domain. So the requester replies with three integers as well. For example, if x_1 is chosen to sign, the reply integers are $[sign(x_1), random\ number\ 2, random\ number\ 3]$. The receiver of the request can easily check whether the requester is valid if at least one of the number is correctly signed.

3 Security

When it comes to security, there are a lot of concerns to consider to ensure a truly secure system. It is often important to be able to put oneself in the shoes of an adversary attempting to break a system to find vulnerabilities. Without this process, many potential attacks may be overlooked and result in a less secure system. In this project many of those concerns are being taken into consideration and prevented to create a more secure cryptosystem.

3.1 Semantic Security

Semantic security introduces the idea that no notable information about the plaintext is extracted from the ciphertext. As mentioned earlier, a textbook public key cryptosystem (PKC) is not semantically secure. this can be remedied by incorporating random padding, or finding a semantically secure PKC without padding.

3.2 De-authentication Attacks

De-authentication Attacks, also known as Deauth Attack, is to force the client sending the handshake package by injecting a fake disconnection package. If the protocol doesn't have an authentication part in the disconnection process, the attacker can inject one valid disconnection package to the valid communication package stream.

3.3 Replay Attacks

An important part of creating a secure key generation and distribution program is preventing the possibility of replay attacks. Nonces are appended to the key distribution to verify that the package has not been tampered with and that the package has not been sent again to receive the same private key. The nonce simply looks as follows:

{{time}}

To prevent the package from be falsified, nonce together with HMAC provides both time information and verification. Nonce and HMAC are checked by the package receiver. if the time on the timestamp has elapsed too long ago, if the session id is incorrect, or if the hash result is different from the hash of the package, the receiver will record the “bad behavior” and send an error message (package) back.

3.4 DOS Attacks

A popular attack on servers is a denial of service (DOS) attack. To ensure that not all of the resources of the server are being used, if there are too many connection attempts in a certain amount of time from a single user, the user is disconnected from the server and is required to wait for a cooldown time before having access to connect to the server again.

3.5 State Machine

When a user is in any of the possible states of communication with the server, there are only a few ways to continue on the process. If there is ever a package received that is out of order with the typical progression of the state machine, it is considered a fraudulent package. Enough punishments will disconnect the client from the server. Fig. 2 depicts the

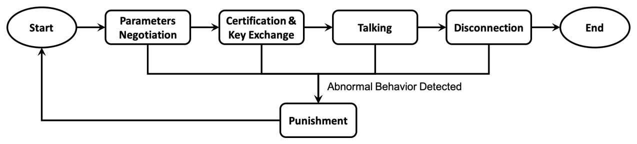

Figure 2: Diagram of the General State Changes

state machine that is formed to verify if every client is following the correct progression of the process.

4 Conclusion

Creating a secure cryptosystem is a great challenge. Years of research is dedicated to improving and attempting to break cryptosystems, in order to create a more secure system. Cryptography is a constantly moving field, and internet security deals with thousands of adversaries on a daily basis attacking any cryptosystem. Furthermore, creating a secure cryptosystem requires creativity and a keen idea of adversary strategies to be able to find vulnerabilities in one's own code.