Steam Stream Team Supreme
Daniel Jacobo
Emil Evangelista

Phase I

Application Properties

Our chat is a 1 to 1 user messaging application. The features included will be user accounts, clickable web links in the chat, and basic chat. Our extra features planned are pictures, 2-factor authentication, and group chat. The platform is planned to be a web application. The language we will be using is Spring for Java.

High Overview: We plan to implement our RESTful server with a class for the server and user. For the database, we will use SQL. For encryption, we will use AES-256. For message integrity, we will use HMAC-SHA256. For key exchange, we will use the Diffie-Hellman method. For client authentication, we will use TLS handshakes using certificates from both parties.

Stakeholders and Assets

The stakeholders for our application are the users.

The assets include: messages, User Accounts, Passwords, private emails, the server itself, and the encryption/decryption keys used.

Adversarial model

We're choosing a fairly low level adversary, no government agencies, or large corporation spies. Our adversary will be a regular person with some knowledge of security, capable of only basic attacks. We chose this adversary because this is only a class project and therefore our user base will be small, therefore having almost no valuable information. This gives us the advantage of being not perfectly secure, but computationally secure.

Possible Vulnerabilities

Client Side

- Keyloggers
- Evil Maid Attack
- Shoulder Surfing
- Weak Security Questions
- Weak passwords
    - Bruteforcing Passwords
    - Password Reuse
    - Easily Guessed

Server Side

- DDOS
- SQL Injection
- Cookie Theft
- Harmful Links
- Man in the Middle
    - Tampered Messages
    - Eavesdropping

Possible Related Previous Work

There are many similar services, all with their own philosophies, and therefore their own pros and cons. The two biggest fish are Signal, and Whatsapp, and the main difference between them is their inception.

Whatsapp was a simple messaging system that was bought out by Facebook in 2014. As of April 2016 Whatsapp, with the help of Open Whisper Systems, implemented the signal protocol in order to provide end-to-end encryption. Even though all messages are now secure in transit, they aren't completely safe from prying eyes. Whatsapp stores all messages locally on the device, therefore someone with an insecure phone can become an easy target. Local storage also creates the problem of backups, any backups of your device will also backup all saved messages, meaning that service now has access to all of your communications. On one hand Whatsapp protects your messages, but on the other hand it automatically collects information about you, including your contacts, and metadata. Whatsapp, being a proprietary

client, breeds doubt within many security experts. With no way to check and verify the code itself the only thing that is ensured is the open-source signal protocol used to encrypt messages.

Signal released in 2014, developed by Open Whisper Systems. The client is open-source and easily reviewable by anyone, meaning any security holes are quickly found and dealt with. Signal employs its own signal protocol in order to encrypt all messages from end-to-end. Similarly to Whatsapp it requests all of your contacts to compare with its user base. That's where the similarities end. Your contacts are hashed, and therefore more secure, your messages are not transferred when your phone is backed up, and they collect as little metadata as possible.

Complete Description

A large amount of attacks will be prevented through the use of HTTPS, but for the rest:

Evil Maid attacks is when someone else chats instead of the original account holder. These types of attacks are prevented by automatically logging out the user after a certain period of time.

Keyloggers are very difficult to get around as the username and password is the cornerstone of many account-based services. The best way to stay secure is using a good 2-factor authentication, but our team isn't planning on implementing all bonus features. As a result we are assuming that the client computer is not infected with a keylogger for the time being.

Shoulder surfing is when a person physically looks over the user's shoulder to steal information. This can be prevented by having asterisks (*) as placeholders instead of displaying the password while it is being entered. For the regular messages, a hide chat button can be used to quickly hide chat if the user has noticed a shoulder surfer.

Weak Security Questions are a boon for social engineers. They take control of people's accounts by attempting to guess common answers to common security questions. Again the best way to prevent this type of attack is to have a 2-factor authentication system. We will continue to assume users pick good, secure answers until a 2FA system is implemented.

Weak Passwords can be prevented by having a minimum length of 12, or 14, and requiring the use of at least one special character.

Reused Passwords can be discovered when another service gets hacked. This is difficult to protect against as most services use similar password requirements.

Easily Guessed Passwords will create a security risk from a large group of people. A friendly reminder will be displayed to not have common information as your password.

DDOS attacks are impossible to prevent, but there are some things we can do to reduce the possibility of an attack. Amazon has a set of "best practices", such as only listening for specific traffic on certain ports, and there are hardware solutions such as load balancers to mitigate the effectiveness of such attacks.

SQL Injections are when people insert SQL commands into a user-entry field. This is easily avoided by using parameterized statements, escaping, pattern checking, or applying database permissions.

Cookie Theft AKA Session Hijacking allows other users (usually on publicly available networks) to read cookies, and copy the legitimate user's session ID. As a result the attacker gains access to the same service, and to the server looks like the actual user. Can be defeated by encrypting all traffic between the client and server.

Our chat is planned to have links in the chat. Many will undoubtedly be used in attacks including phishing links, or ones leading to virus downloads and other malware sites. By blocking known harmful sites we can prevent the majority of these attacks.

Man in the Middle attacks can be prevented when the system can correctly authenticate the recipient of the message. This accounts for the integrity of the messages so that it is sent to the right person. Messages should not be editable by anyone else other than the user that sent the message. For eavesdropping, measures need to be taken into account to prevent third party users from listening into the conversations of the original parties.

Rigorous Analysis

Our chat application meets many requirements for security, but not all of them. We will apply many security protocols and practices in order to increase our effective computational security. We only have a small user base, none of which are high-profile targets, so the value of our information is minimal at best. Our security features are meant to increase the cost of retrieving this information to unreasonable heights. As a result we deter the larger adversaries, and focus only on those with a limited amount of time and resources. Confidentiality will be met by the encryption of messages and the passwords in the database. Users can only communicate with those who are also part of our ecosystem, therefore ensuring security throughout the entire transmission process. All information in our system will be secure against third parties looking to alter messages. We employ many measures in order to protect all users and their communication.

We protect against:

Angry Maid attacks by limiting idle time. If users are not actively using the service we can terminate the session and generate a new key the next time they try to login.

Shoulder Surfing by hiding sensitive information about our users. Passwords will be masked by asterisks (*), and having a "hide chat" feature to be activated at the user's discretion.

Weak Passwords can be prevented by applying strict rules. A minimum length of 12 or 14 characters, with at least one special character is enough to prevent brute force by most adversaries. Further these passwords will be stored in a hash, to allow extra security in the case of a security breach.

Passwords that include personal information are often weak due to close friends and family having indirect knowledge of your password. We urge all users to pick secure passwords that contain no personally identifiable information.

Similarly there are passwords that meet a large amount of security requirements, and yet offer no security value. These are lazy combinations of common patterns. Hackers have identified over previous attacks that certain passwords like asdf123 are very common, and therefore are often tested in preliminary attacks. We will show examples of “Secure Passwords” that are weak for this reason and encourage users to refrain from choosing similar passwords.

DDOS attacks will happen, and affect the Availability of our service. Amazon has a set of “best practices”, such as only listening for specific traffic on certain ports, and there are hardware solutions such as load balancers to mitigate the effectiveness of such attacks. Amazon also offers some protection against “Layer 1-4” attacks, and any higher are largely unavoidable.

SQL Injections can be preventing by implementing small changes in the code itself to sanitize user inputs. Parameterized statements mean the user can only input a certain type of data, that will never be read as a SQL Instruction. Escaping is when you edit any inputs to neutralize any “blacklisted” characters, for example a single quote (‘) would be neutralized by adding another (‘’). Pattern Checking only accepts certain data types when their value fits a certain pattern, such as dates or alphanumeric characters only. Database Permissions can restrict selection of certain tables from the database logon.

Cookie Theft can be defeated by encrypting all traffic between the client and server. If the adversary is unable to see the information in our system they will be unable to impersonate valid users.

For harmful links, URL shortening websites are suspicious to click on as it hides what the website really is and can be used for phishing attempts. Google’s “Safe Browsing Lookup API (v4)” should work in blocking the majority of links that can be considered harmful if clicked on. The user should not click on links that can cause suspicion and should only trust websites that the user knows.

Man in the middle attacks can be defended by methods to authenticate the user and have means to protect the alteration of information. HTTPS will encrypt the chat to provide secure communication.

We are not currently protecting against these issues due to extra features needing implementation which may or may not happen:

Keyloggers are difficult to defend against without implementing 2-Factor Authentication. A unknown device with the correct username and password is just as likely to be the User, or not. 2FA would prevent any attackers from knowing the additional code necessary, and allowing the legitimate user to log in securely by connecting their account to another service such as their phone number or email.

Weak Security Questions are difficult to defend against, because correct answers are meant to be indicative of their true Identity. This type of Social Engineering is very effective when people spread information over many accounts with the same username/email. Oftentimes mother’s names, or elementary schools are freely available on facebook or other sites. We can mitigate these attacks by asking very unique and specific security questions, but even those have the possibility of being discovered. 2FA would prevent the majority of intrusions should we get the chance to implement.

Reused Passwords can be discovered when another service gets hacked. Users often prioritize convenience over security and reuse both username and password combinations across multiple sites. This creates a security issue because even if we are entirely secure their information is still compromised. Two-Factor Authentication would protect the user's account in case of Password Reuse.