Phase 4

Group 12 Investigating Group 14

Intro

To test this chat application, we made a new vApp with 2 virtual machines so we could set up a server and start multiple clients and send each other messages.

Testing Steps

- We attempted SQL injection, but the use of prepared statements prevented any successful attempts.
- We tried performing a traffic capture and using the tcpreplay tool to try a replay attack which would allow an attacker to resend messages from one user to another without having the valid login credentials required.
- We tried adding friends, then removing them and seeing if both parties could still send and see each other messages.
- Used wireshark captures to verify that communications between clients and server were encrypted and not readable by an attacker without the proper decryption keys.
- We tried many combinations of adding friends and removing them in various different orders to see if there were any interesting results.

Findings and Recommendations

Man in the Middle Attacks are possible

In an ideal situation, the chat server for this application would be hosted at a known and trusted domain name and it would have its own certificate. But in this case the server wasn't attached to any domain name or IP address in particular. This means that an attacker could run an instance of the chat server or they could intercept traffic sent to a real server and man-in-the-middle it or just become the server. The solution to this would be to set up a chat server at a known and trusted domain as stated previously. Then you could ship the client application with the server's exact certificate and implement certificate pinning so that the client will only connect to the server that has the exact domain name and certificate, leaving no opportunity for a man-in-the-middle attack.

No salt used for passwords

The passwords are hashed using SHA-256 which is a good algorithm to use, but without using salts the hashes are weak to rainbow table attacks. Furthermore, if there are duplicate hashes (meaning two users chose the same password), an attacker would only have to crack one of the hashes and then they would know the password to all the accounts using the same hash. To counteract the rainbow table issue you could use a static salt of 20 or more secure randomly generated characters. To counteract the duplicate hashes problem you could use a dynamic salt (for example the username). The combination of a static and dynamic salt will prevent both of these attacks.

Connection between clients after defriending each other stays active

After adding and then removing another user as a friend, both users can still send each other messages and see them because the connection between them stays active essentially until one of the clients quits the application. This connection should be terminated or there should be some validation before a message is sent to somebody of whether or not the two users are actually friends.

Replay attacks are (theoretically) possible

When we tried a replay attack of partial communications between clients and client/server they failed, but there doesn't seem to be any checks in the code to prevent against a replay attack. Therefore, since SSL/TLS doesn't prevent against replay attacks on its own, an entire SSL/TLS transaction should theoretically be replayable. This would allow an attacker to get a packet capture and replay the traffic, performing all of the actions the user did during the transaction. The attacker might not know the contents of the traffic, but they can perform the actions that user did with this attack.

Reference:
http://security.stackexchange.com/questions/20105/are-ssl-encrypted-requests-vulnerable-to-replay-attacks/20106#20106

No password policy enforced for users during registration

Currently users can make short and completely insecure passwords. This makes cracking password hashes or guessing user passwords very easy because users tend to pick short and simple passwords. Enforcing a password policy to ensure that more secure passwords are chosen will increase security of the application.