Team Gamma
8/6/2017
ICS 491

Requirements and Design

Technical portion

1. *Meet with your group and find a programming language that you are all comfortable with.*
   - We decided to use Java

2. *Decide what type of program you would like to make. (i.e. a Java applet, a web-based application, etc.)*
   - Android Contacts List/Chat Room app

3. *Decide what your program will do (i.e. a game, a money-tracker, a personal site, etc.) You can be as creative as you want, but it should contain some sensitive data that must be protected (i.e. user names and passwords, user information, game top scores, etc.)*
   - Make a contacts list with chat room that will store information such as usernames/password. We also plan for chatrooms to be private and only joinable if password to chat room was given to participant so that they can enter chat room.

Requirements

1. *Establish security requirements: Define your own security and privacy requirements for your program. Also, come up with a system of keeping track of security flaws throughout the development process and describe this in your report as well.*

The security development lifecycle is important to our program as we intend to store personal information including usernames/emails, passwords, contact list and chatrooms of our users. Since we are dealing with usernames, it will be imperative that we implement a secure authentication mechanism that ensures only authorized users have access to the account where top scores could be managed. This would include requiring a strong password with a minimum of 8 characters, which must include an uppercase, lowercase, number and special symbol. Passwords stored should be encrypted with salt to mitigate successful brute force attacks.

It would also be required to timeout the user from their session if they are away and not playing after 15 minutes of inactivity and require re-authentication. This is to prevent unauthorized users from tampering with the account in case of using a public or shared device. As our application will be a simple Android mobile chat room app, we will not require data to be shared with any other apps nor would it need any permissions to the device features as they are not needed. This will keep user data of those using our application more secure. In order to track security flaws during development, we may use a tracking system such as github or bugzilla as we work out the flaws and make changes to the code.

2. *Create quality gates/bug bars : Define levels of security and privacy for your program. Here are some samples:privacy, security. Your security gates/bug bars do not have to be as complex, but you should choose which situations would apply to your program.*

Everything is Scaled to End-User Scenarios:

Privacy Bars: Looking at the the type of application we want to develop, there are no plans for making a revenue, so we are limiting the confidential data to scores and usernames.

Team Gamma
8/6/2017
ICS 491

At a Critical-Important level, these would be considered vulnerabilities:

- Lack of notification of messages sent or received
- Lack of notification when participants enter/leave chat rooms
- Lack of notification from logins from different devices for accounts (mobile app)
- Lack of data protection for account information
- Lack of the internal data management and control,
    - Example would be that the host of the chatroom cannot be changed unless done so by the host.
    - Chatroom participants cannot edit or delete messages that are not their own.

At a Moderate level:

- Lack of notification of messages sent or received
- Lack of data protection for account information

At a Low level:

- Lack of notification of messages sent or received
- Lack of notification when participants enter/leave chat rooms

Security Bars: Since our chatroom will use a server and our players will be communicating using an Android mobile device, we need bug bars that cover those scopes.

Server-Side

- At a Critical level:
    - Server should not allow any easy elevation of privilege that allows users to change host of chatrooms.
        - Therefore elevation of privileges for certain functions only when called involving the usage.
- At an important level:
    - Distributed Denial of Service attacks
        - For a chat room, it is detrimental for a DDoS as it will have an effect on the usage of the application.
    - Information Disclosure
        - Must not allow end users from mobile devices to access data that is meant for server side only.
    - Spoofing
        - Must not allow any risk of masquerading on admin side such that they access admin functions for more information.
    - Tampering:
        - Must not allow tampering that will have an effect on the chat rooms and account info.

- At a Moderate Level
    - Distributed Denial of Service attacks
        - There must be a minimal amount of protection.
    - Information Disclosure
        - Must not allow end users from mobile devices to access data that is meant for server side only.

- At a Lower Level
  - Information Disclosure
    - Leak of memory from server at random runtime.

Client-Side

- At a Critical level
  - Elevation of privileges from remote side using the client app

- At an Important-Moderate Level
  - Denial of Service
    - Should not allow enough to get to the point where the application needs to be reinstalled.
  - Spoofing
    - Mobile user should not be able to easily impersonate or spoof another user's account.
  - Permissions
    - In most mobile apps, permissions are needed in order to access parts of the phone's data. Must not allow unnecessary permissions for access to minimally important data.
- At a Low Level
  - Denial of Service
    - Application restart needed in order to work in the case a Denial of Service happens.

3. *Perform security and privacy risk assessments: Determine how you will assess which parts of your program need threat modeling and security reviews. Here is a template example.*

The parts of our application that will need threat modeling and security reviews include the data stored in the database related to our user accounts including username, password, and scores. Parts in need of the most attention would be our higher risk vulnerabilities.

Our app stores identifiable information in the form an email address and our app provides an experience (game) that is attractive to children. Which will put our app in the P1 category. We will store user information in a Firebase database. We will notify the user when signing up for an account that our app will only store their email, username, and password on an external database. No other information that is accessible through the Android API (phone info..etc) will be stored. User's will have the ability to delete their account at any time. We will prevent unauthorized access to PII through the use of Firebases's user authentication API.

Design

1. *Establish design requirements: Decide on standardized design requirements. For instance, look back to the privacy and security requirements in part 1. What design requirements should you implement to make sure that you meet these requirements?*

For personal account creation and management we will use Firebase. Firebase provides secure user authentication and their website has full API documentation on how to implement their technology to

be secure. The documentation for setting up accout authentication is located here:
https://firebase.google.com/docs/auth/android/start/

For setting up chat rooms and tracking message we will also use Firebase. Firebase has excellent documentation on how to manage a realtime database located here:
https://firebase.google.com/docs/database/android/start/

Firebase does allow offline data persistence in the case of lost network connection. This can be a security risk if the user is able to manipulate this local cache in anyway (change chatroom passwords). Because of this our app will not share data or store data on the device so internal or external storage management is not required. This does mean the user will have to have an internet connection to use our game. Which is easy to implement if user login is required before doing anything within the app.

How the user interacts with the database will need to be managed properly within application. Interfacing with a Firebase database is conducted through their API and any forms submitted this way must be authenticated properly according to their documentation.

2. *Perform attack surface analysis/reduction : Determine privilege levels for your users and perform an attack surface analysis. Since this the first version of your program, compile a list of vulnerabilities from a similar program that could apply to yours. This will give you an idea of how users may attempt to exploit the code.*

Privilege Levels

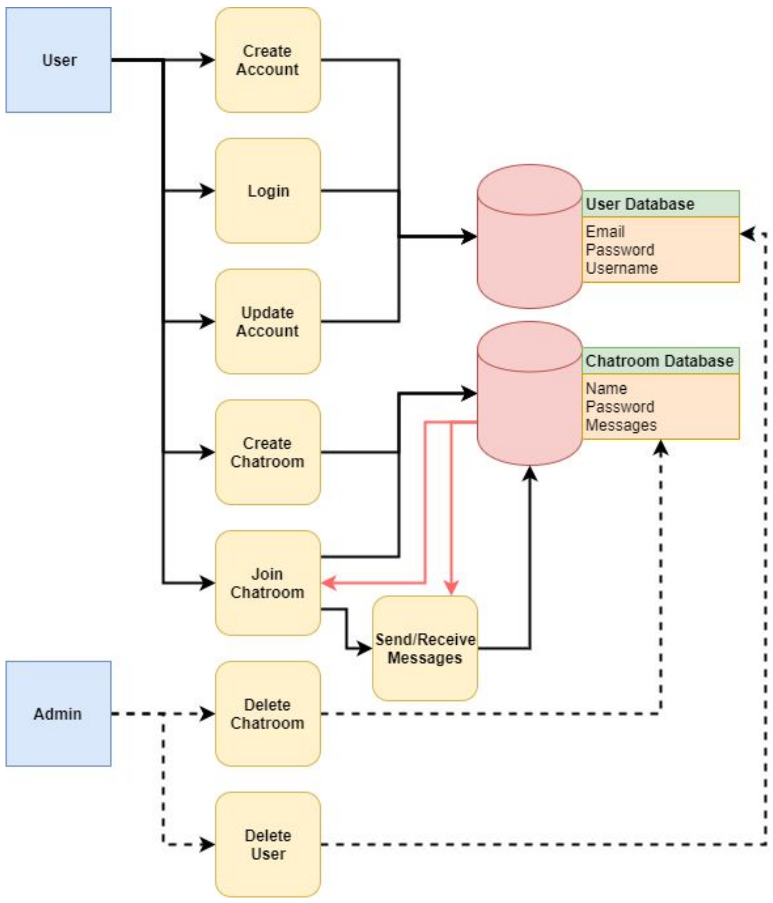
- Admin
    1. Delete/ban users
    2. Delete chat rooms
- User
    1. Create Account
    2. Login to Account
    3. Change/update their password and username
    4. Create chatroom with password
    5. Join chatroom with password

Vulerabilities

- Form data at account creation
- Form data at login
- Form data when updating user information
- Form data when creating chatroom
- Chatroom password storage
- Sending messages to chatroom
- Joining a chat room with password form data

3. *Use threat modeling: Create a threat model for your program. You may use the SDL threat modeling tool, but you are not required to. Your threat model should contain a graphic representation of all the information in your program and how it is all connected (a data flow diagram (DFD)). Add privilege boundaries to your diagram and categorize your possible threats. If you're confused, just complete steps 1 and 3 here.*

Team Gamma
8/6/2017
ICS 491



Spoofing

User impersonating another user.

Tampering

User could tamper with a chatrooms password

Information Disclosure

Exposing a user's information to another user.

Denial of Service

Attempting to crash server by sending database requests continuously or creating user accounts repeatedly.

Elevation of Privilege

A user elevating his privileges to the level of admin.

Implementation

This assignment is focused on implementation.

1. Approved tools: Compile a comprehensive list of approved tools for your project. Here is an example pertaining to Microsoft. Make sure to include everything required for your program. Note that versions are very important, because there can be huge differences between versions of the same compiler/IDE/etc.

Compiler/Tool	Version	Comments
Android Studio	Version 2.3.3.0	https://developer.android.com/studio/index.html

Team Gamma
8/6/2017
ICS 491

Firebase	JVM Client Version 2.3.0	https://firebase.google.com/docs/auth/android/start/

2. Deprecated/Unsafe Functions :
    a. Firebase Deprecated API: https://www.firebase.com/docs/java-api/javadoc/deprecated-list.html
        i. Deprecated Interfaces
            1. **com.firebase.client.Firebase.AuthListener**
                a. The interface that serves as a listener to changes in the authentication state of the Firebase. This library has functions from a security standpoint that may be used against the firebase as attackers could possibly see what functions are called. The methods in the interface are onAuthError, onAuthRevoke, and onAuthSuccess. You can infer what happens when these methods are called.

                b. Alternative: Firebase.AuthStateListener
                    i. This interface brings down every handle of the changes to the authentication state in one function, onAuthStateChanged() and takes in an AuthData object that handles how to respond to the call.
        ii. Deprecate Methods
            1. public void auth(String credential, Firebase.AuthListener listener)
                a. Since the AuthListener was deprecated, this method becomes deprecated as a result as well
                b. Alternative: public void authWithCustomToken(String token, Firebase.AuthResultHandler handler)
                    i. This is an alternative that will allow you to do the same thing without needing to use a deprecated interface.
            2. public void enablePersistence()
                a. This method is set by default to allow users to stay authenticated even after restarts of the Firebase. However it was only set to true so that means persistence was not disabled by default.
                b. Alternative: public void setPersistenceEnabled(boolean isEnabled)
                    i. Allows user to set persistence to true or false.
            3. public MutableData getParent()
                a. Method that returns the parent of a MutableData object, it will eventually to be plan.
                b. Don't plan on using MutableData object for this project.
            4. public Query limit(int limit)
                a. Method that returns a limited number of Child nodes for of a Query
                b. Alternatives:
                    i. public Query limitToFirst(int limit)

                1. Must set the anchor when calling, in this case it'll anchor to start of window
            ii. public Query limitToLast(int limit)
                1. Will set anchor to the end of the window
    5. public void unauth(Firebase.CompletionListener listener)
        a. Method that unauthenticates all credentials of a client, which also unauthenticates the connection of the credential at the same time.
        b. Alternative: public void unauth()
            i. Just use the unauth without parameter to do the samething.

3. Static Analysis: Choose a static analysis tool for your group to use during the development process. Use this tool each time you recompile the code. Briefly document the tool that you have chosen. After using it a few times, come back to this section of the report and describe your experiences using it (any difficulties? any specific successes?). Examples: Checkstyle for Java, *uhunix* is good for C and C++ style.

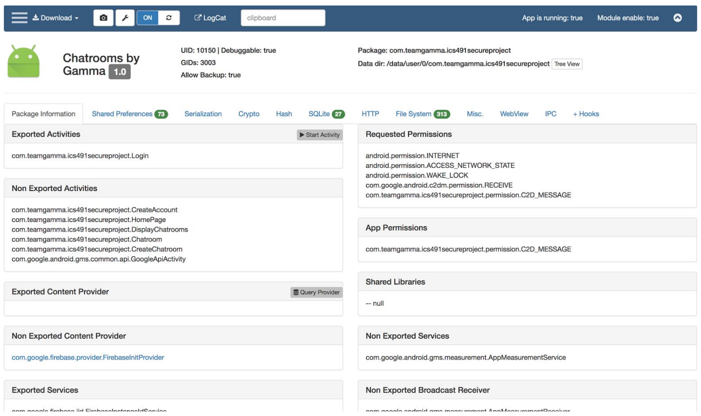
https://plus.google.com/+KevinTanHongAnn/posts/iCpCp1yzDt6

The Android Studio IDE has a built in static code analysis tool called Lint which can be called without the need for any additional set up. For ease of usage, it's very simple because once you choose the option within Android studio to "Inspect Code", it will call Android Lint and break down the the warnings and errors that occur between the files.

Verification

Compile a formal report with three sections as outlined below. The report should be about 3 pages(give or take). Where deemed appropriate, you may use lists or tables instead of paragraphs:

1. Dynamic Analysis: Choose a dynamic analysis tool for your group to use during the development process. Briefly document the tool that you have chosen. Use this tool each time you run your program after making significant changes to the code to verify that your program is responding in ways that you expect during input/output. After using it a few times, come back to this section of the report and describe your experiences using it (any difficulties? any specific successes?)



Chatrooms by Gamma 1.0

UID: 10150 | Debuggable: true
GIDs: 3003
Allow Backup: true

Package: com.teamgamma.ics491secureproject
Data dir: /data/user/0/com.teamgamma.ics491secureproject

Exported Activities

com.teamgamma.ics491secureproject.Login

Non Exported Activities

com.teamgamma.ics491secureproject.CreateAccount
com.teamgamma.ics491secureproject.HomePage
com.teamgamma.ics491secureproject.DisplayChatrooms
com.teamgamma.ics491secureproject.Chatroom
com.teamgamma.ics491secureproject.CreateChatroom
com.google.android.gms.common.api.GoogleApiActivity

Exported Content Provider

Non Exported Content Provider

com.google.firebase.provider.FirebaseInitProvider

Exported Services

Requested Permissions

android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.WAKE_LOCK
com.google.android.c2dm.permission.RECEIVE
com.teamgamma.ics491secureproject.permission.C2D_MESSAGE

App Permissions

com.teamgamma.ics491secureproject.permission.C2D_MESSAGE

Shared Libraries

-- null

Non Exported Services

com.google.android.gms.measurement.AppMeasurementService

Non Exported Broadcast Receiver

The dynamic analysis that was used for this iteration of development was Inspeckage (https://github.com/ac-pm/Inspeckage), which is an Android Package Inspector that monitored the activities of the code as it ran. It required a rooted Android device (can be rooted in emulator, however one member used their personal Android Device), and it required a framework to be installed called Xposed. When I run the application, I run it through the inspect application as you will see below in the screenshot. Through the experience in the beginning it was difficult to set up, once it was configured it was easy to see the activities and behavior of the application as I used it. I was able to identify where there were slight insecure places. In hashes I could see the usage of MD5 and SHA1, however they for information that is inconsequential. I was able to see SQL queries, portion of the shared preferences which held JSONs with username information. I could not see passwords however. Overall with the running of this dynamic analysis tool I could identify vulnerabilities easily as there are many tools that and options for API hooking. Much of the data I observed is salted so it will pose as challenges for potential attackers of this application.

2. Fuzz Testing : Attempt to break or hack into your program by using various techniques. You may use the ones in the weeks 3 and 4 lesson plan or you may find others. Document at least 3 of your attempts. (Describe the hack you did, your success rate, implications, how you fixed it/why it was secured, etc.)
   Disclosure: Hacking attempts came from the perspective that I do not know the internals of the application so there may be unconventional methods.
   1. Attempt #1: Intercept HTTP Request/Responses through BURPSuite to Masquerade and Change messages
      a. Success Rate So Far: 50% (100% after proper proxy configuration)
      b. Implications: Messaging Application does use HTTP to transport message JSON, I could easily impersonate as various users and also change the content of message so long as I intercept it. Even in real time I could change a message that has just been added to the chat. This also renders the accounts useless given that I can easily masquerade as someone else. I can also trick the firebase temporarily into changing the sender to a nonexistent name. However after some time the name will revert to the original sender's name.
      c. Secure: After the configuration of BURP Suite proxy, this vulnerability would require that I have to willingly intercept connect to a network that is capturing and forwarding request.
      d. Fixing it: So long as we encrypt the outgoing traffic, we could solve this issue.
   2. Attempt #2: Hack Application in Memory using Game Guardian
      a. Success Rate So Far: 0%
      b. Implications: Message Queries are not stored locally on phone and therefore the only real way to manipulate messages in application is to manipulate the firebase itself or the responses going to the firebase.
      c. Secure: Due to the fact that there are no real changeable values that affect the integrity of the application, the operational function is still secure and therefore no real threat overall to the intended use and purposes.
   3. Attempt #3: Capture web packets that held information through WireShark
      a. Success Rate: 0%
      b. Implications: Even when web packets are captured through wireshark, I cannot gain any compromising information from the packets. No users or passwords are shown in clear text. I can find the destination, but not anything else..
      c. Secure: Through network monitoring and no proxy involved, the information being sent is secure when going from application to firebase.

3. Attack Surface Review: Return to your report from last week and look at your approved tools. Have there been any changes, updates, patches..etc in any of these tools? Have there been new vulnerabilities reported? It is likely that nothing has changed given the time frame of this project.

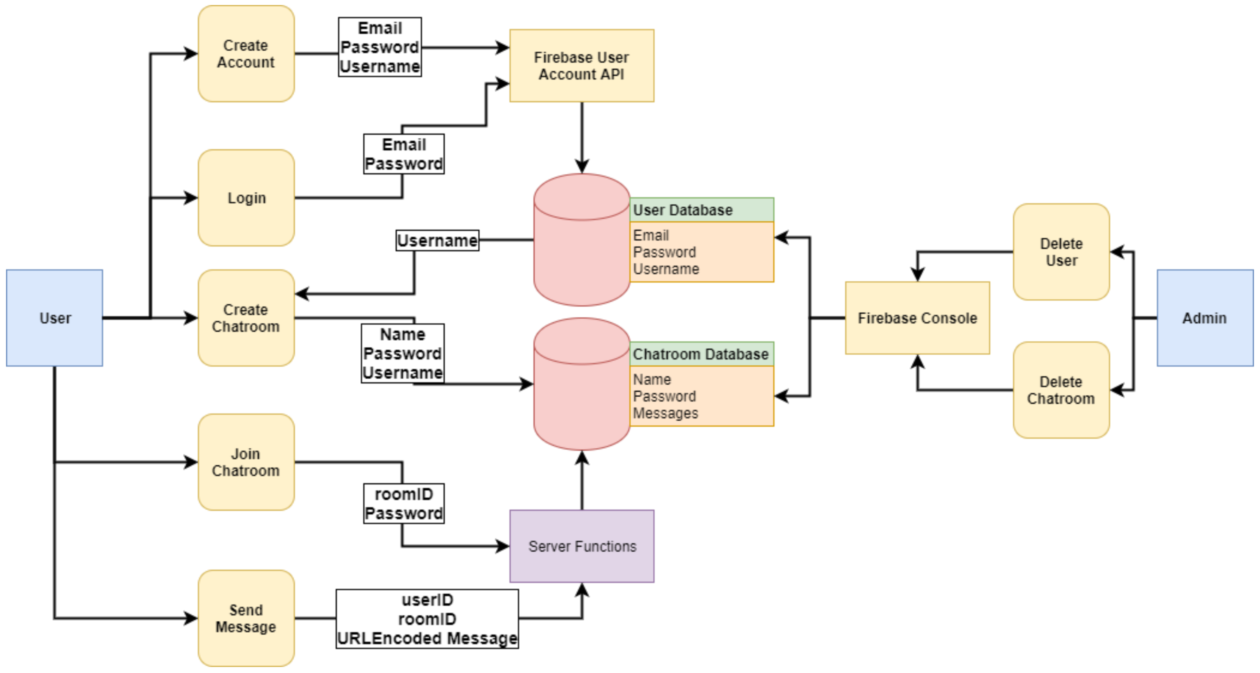
Team Gamma
8/6/2017
ICS 491

For our approved tools of Firebase and Android Studio there have been no changes, updates, or patches in the tools we are using. Any new vulnerabilities that we have discovered has been documented above and is from our own carelessness.

The biggest vulnerability was discovered using BurpSuite where a user could send a chat room message masquerading as another user. To fix this we are now validating any new messages server-side. When a user sends a message his userID, roomID and message are sent to the server using HTTPS. Both userID and roomID are unique and not shown in cleartext, the only thing in clear text is the user message. Before being sent to the server we use Java's URLEncoder on the message to mitigate any XSS. Server-side is where we now insert the message into the chatroom database which is more secure than before where we were performing this client side.

We have also implemented password protection for the chat rooms. This is also performed server side similar to the messaging. When the user enters a chat room he inputs a string. We then encode this using Java's URLEncoder and send the request to the server. The server validates that the password entered matches the chat room's password. If they match code 200 is returned to the client and he is allowed entry. If the password is incorrect the server returns code 403 and the user is returned to the chatroom list.

Below is an updated chart showing our applications current design.



Team Gamma
8/6/2017
ICS 491

Release

This assignment is focused on the release portion of the SDL. This is the final group assignment for this course, and should reflect the different concepts learned this semester about secure development and the SDL.

1. Incident Response Plan:

Privacy Escalation Team		
Escalation Manager	Receives the first notice of escalation and determines severity and course of action to resolve issue.	Anson Yu
Legal Representative	Help resolve any legal concerns of the application and assist with producing an approved privacy disclosure.	Kristy Woodward
Public Relations Representative	Handles public relation concerns regarding the application.	Keenan Kinimaka
Security Engineer	Focuses on the security aspects of the application, ready to identify and update vulnerabilities as they are discovered.	Ryan Theriot

Submitting Privacy Escalation Requests

All requests can be reported to: GammaChat491@gmail.com

Incident Response Process

Escalation begins when the first e-mail notification of the issue is received or as soon as a vulnerability or flaw is discovered. The escalation manager is the first point of contact, responsible for evaluating the issue to determine whether more information is required. If so, the escalation manager is responsible for working with the reporting party and other contacts to determine:

- The source of the escalation
- The impact and breadth of the escalation
- The validity of the incident or situation
- A summary of the known facts
- Timeline expectations

It is the duty of the escalation manager to disseminate this information to appropriate contacts and seek resolution. Appropriate resolutions should be determined in cooperation with the reporting party and other applicable contacts including the attention of the security engineer. The escalation manager should ensure that all aspects of the escalation are resolved. All necessary changes and updates to the application should be handled by the security engineer to ensure flaws and vulnerabilities pertaining to the issue have been resolved. The escalation manager should confer with legal and public relations representatives as necessary based on the severity of the incident to determine the proper action to take place. Appropriate resolutions might include some or all of the following:

- Internal incident management
- Communications and training
- Human resources actions, in the case of a deliberate misuse of data
- External communications, such as:
    - Online Help articles
    - Public relations outreach
    - Breach notification
    - Documentation updates
    - Short-term and/or long-term product or service changes

Closing the report

After all appropriate resolutions are in place, the privacy escalation team should evaluate the effectiveness of privacy escalation response actions. This includes An effective remediation is one that resolves the concerns of the reporting party, resolves associated user concerns, and helps to ensure that similar events do not recur.

2. Final Security Review:

Threat Model Analysis:

Team Gamma
8/6/2017
ICS 491

Spoofing

User impersonating another user.

Review: During development process was an issue, but is now no longer an issue because the server now verifies and corrects any attempts of impersonating another user or that of an existing user. It'll correct in the case where the user does not exist and automatically reverts to the original accounts username. As for the case when the username is being impersonated under another account, it will revert back to the original accounts username as well.

Tampering

User could tamper with a chatrooms password.

Review: Currently as is, the user cannot tamper with the password of the chatrooms, they cannot change it nor modify it to be their own password. It has never been the case so far and it never will be with the application as is.

Information Disclosure

Exposing a user's information to another user.

Review: Currently as is, the end users would only be able to see other user's emails and usernames as is but no other sensitive information through means of usage on the mobile phone application. However if the application is proxied through a computer and intercepted through a program such as BURPSuite, some information is disclosed such as Chatroom passwords and messages from other chatrooms. However application behavior through proxy is that of ineffectiveness. Chatroom application ceases to function and crashes if request for access to chat room is done through a proxy. Vulnerability only happens during the first load up of chatrooms and does not happen any further beyond that.

Denial of Service

Attempting to crash server by sending database requests continuously or creating user accounts repeatedly.

Review: Currently has not been possible at any point of the development cycle, client side application would crash before server side becomes compromised through denial of service so far.

Elevation of Privilege

A user elevating his privileges to the level of admin.

Review: Currently has not been possible at any point of the application development.

Final Security Review Grade:

Passed FSR with Exception

Exception Explained: Currently with information disclosure, although I am able to see the password of Chatrooms. However in the context of usage, a user cannot manipulate the conversation or masquerade as another user when entering the chat. Therefore currently it is functional in making sure to keep the integrity of the user's words. In a future release, this may be addressed.

Team Gamma
8/6/2017
ICS 491

Release Link
https://github.com/rctheriot/ICS491SecureProject/releases/tag/v1.0

README
https://github.com/rctheriot/ICS491SecureProject
https://github.com/rctheriot/ICS491SecureProject/wiki

Progress 8/6/2016

Completed Task:

- Code Review and Commented
- Wiki and User Guide
- APK Release
- Real Device Testing
- Icon Created

Pending Task:

- Unique usernames & room names

Individual Contributions

- Ryan:
    - Wiki and User Guide
    - Design Documentation
    - Code commenting and clarity

- Kristy:
    - Formal Report
    - Incident Response Plan
    - Escalation Response Process
    - Design Documentation

- Anson:
    - Final Security Review
    - Design Documentation
    - End User Testing