Kevin Zhang
Application Security CS GY 9163

Homework 4: Mobile Mess

On NYU Classes, submit a link to your GitHub repository. The repository should contain all of the files of the Android project, plus the text files BUG.txt and difference.txt. For this section, your instructor is: **Kevin Gallagher**, GitHub ID **`kcg295`**. Your TA is: **Evan Richter**, GitHub ID **`evanrichter`**

Author note: Please note that all comments to code created by me are indicated by //KZ:. This is to assist in grading and locating changed parts of the code.

Part 1: Setting up Your Environment

Android Environment

- Set up Android Studio on Windows 10
- Set up Android emulator, Pixel 3a, image R, API 30, x86 ABI, Target Android 11.0 (Google Play)
- Imported "GiftCardSite" project
- Emulator run successfully, able to start up virtual Pixel 3a

Part 2.1: It's all about intent – What's the difference?

I have included *difference.txt* in the assignment repository; however, I have also included the answers to the questions below. Please note (#) refers to the specific questions referenced.

1. What are the two types of Intents?
2. Which of the two types of Intents are more secure?
3. What type of Intent is shown on lines 69 to 73 of SecondFragment.kt?
4. What type of Intent is shown on lines 68 to 70 of ThirdFragment.kt?
5. Which of these two Intents is the proper way to do an Intent?

a) Questions 1-5: The two types of Intents used for Android are Implicit Intents and Explicit Intents (1); in terms of security, Explicit Intents are much more secure, due to their definite specification of responding/interacting services. Implicit Intents do not allow identification of services - messages can be sent to wrong components (2). "Line 69-73 of "SecondFragment.kt" is an example of an Implicit Intent for a website, while lines 68-70 of "ThirdFragment.kt" is an Explicit Intent (3,4) - "Third Fragment.kt" and its Explicit Intent are the proper way, as they cause a web page external of the app to be opened (5).

b) Fixing the incorrect Intent: In order to fix this, we modify "SecondFragment.kt" to use Explicit Intent by specifying the action "scrollingprofileactivity" rather than utilizing the "intent.actionall" key, avoiding ambiguity and not utilizing unnecessary components. This is done with "var intent = Intent(activity, ProductScrollingActivity::class.jav".

Part 2.2: It's all about intent – Shutting out the world

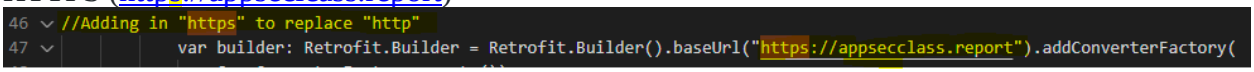
"AndroidManifest.xml" is flawed in that it allows other (potentially unverified) applications to use "GiftCard" to run Activities - patching this issue will secure a lateral vulnerability. I have modified ".UseCard", ".GetCard", ".ProductScrollingActivity", and ".CardScrollingActivity" with android:exported="false," which specifically defines that no other app can launch this activity (even the Android system itself).[1] Additionally, we also have added the XML attributes tool "tools," which also allow error handling. This will allow the android:exported="false" to fail safety. Please see accompanying HTML comments within the AndroidManifext.xml document attached in the assignment repository.

Part 3: Can you read me out there?

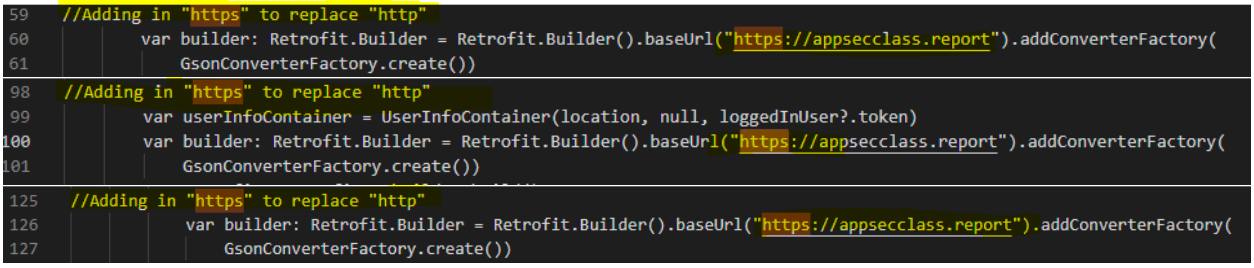
Please see below changes to the files to implement HTTPS. HTTPS provides additional security through TLS (SSL) public key encryption.[2] This provides authentication and, like changing Intents from Implicit to Explicit, will harden the mobile application.

1. SecondFragment.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)

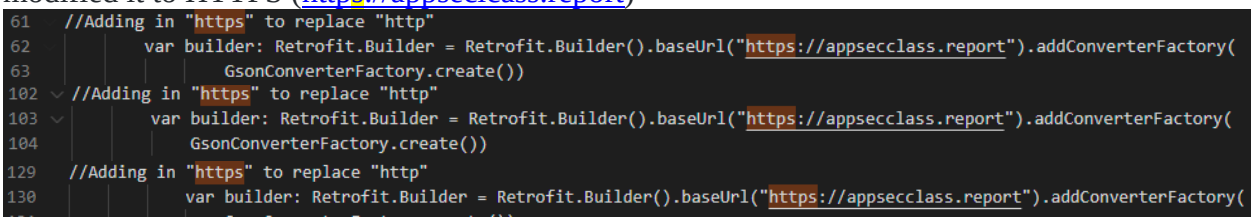
2. ThirdFragment.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)



3. CardScrollingActivity.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)



4. ProductScrollingActivity.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)



5. UseCard.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)

[1] Description ol android:exported="false": "https://stackoverflow.com/questions/49471423/android-manifests-androidexported-false-prevents-app-from-running-on-device"

[2] Description of HTTP: "https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/#:~:text=HTTPS%20is%20HTTP%20with%20encryption,uses%20HTTPS%20has%20https%3A%2F%2F."

```
35  //Adding in "https" to replace "http"
36          Glide.with(this).asBitmap().load("https://appsecclass.report/" + card?.product?.productImageLink).into(image)
37          val loggedInUser : User? = intent.getParcelableExtra("User")
38          var token : String = "Token " + loggedInUser?.token.toString()
39          Log.d("Token check", token)
40          val outerContext = this
41          var button: Button = findViewById(R.id.submit_buy)
42          button.text = "Use Card"
43          button.setOnClickListener{
44  //Adding in "https" to replace "http"
45              var builder: Retrofit.Builder = Retrofit.Builder().baseUrl("https://appsecclass.report").addConverterFactory(
```

GetCard.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)

```
31  //Adding in "https" to replace "http"
32          Glide.with(this).asBitmap().load("https://appsecclass.report/" + product?.productImageLink).into(image)
33          val productNumber : Int? = product?.productId
34          val loggedInUser : User? = intent.getParcelableExtra("User")
35          var token : String = "Token " + loggedInUser?.token.toString()
36          Log.d("Token check", token)
37          val outerContext = this
38
39          findViewById<Button>(R.id.submit_buy).setOnClickListener{
40              val amount : Int = parseInt(findViewById<EditText>(R.id.amount).text.toString())
41  //Adding in "https" to replace "http"
42              var builder: Retrofit.Builder = Retrofit.Builder().baseUrl("https://appsecclass.report").addConverterFactory(GsonConverterFactory.create())
43              var retrofit: Retrofit = builder.build()
```

6. CardRecyclerViewAdapter.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)

```
21  //Adding in "https" to replace "http"
22              Glide.with(context).asBitmap().load("https://appsecclass.report/" + card.product?.productImageLink).into(image)
```

7. RecyclerViewAdapter.kt – in segments referencing the http://appseclcass.report modified it to HTTPS (https://appseclcass.report)

```
23  //Adding in "https" to replace "http"
24              Glide.with(context).asBitmap().load("https://appsecclass.report/" + product.productImageLink).into(image)
```

Part 4: Oops, was that card yours?

I have included *bug.txt* in the assignment repository; however, I have also included the answer to "explain why this vulnerability may be occurring, and how it can be fixed":

Vulnerability: The vulnerability exists within the application mostly because the application itself does not check if the authorization matches the card user; only that the authentication token is valid. This is represented in the code when only variables card.id and user_token are used to validate. Taking two random users, User 1 (victim) and User 2 (attacker), for example: Running useCard() (e.g.UseCard.kt) with User 1's id and User 2's valid token, will allow User 2 to pay using User 1's gift card. Anyone with a valid token can then use whichever card they choose, if they know (or can guess) another card's ID.

Fix: To further increase security, authentication can be used to validate that a gift card does indeed belong to the individual who is using it. A fix can be implemented by creating additional checks of User ID of the card with the User ID of the Token in the application database (functionality exists already to pair User ID with Token, and User ID with Card ID). Additionally (outside the scope of this assignment) this potentially could be validated by pairing the gift card with a users' phone number or, more securely, through a 2-step mobile authenticator to avoid additional PII tracked in the App (i.e. Google Authenticator). In the end, if the user cannot authenticate, then the card operation can be rejected, printing a "Card was not successfully authenticated" message.

Part 5: Privacy is Important

I have gone ahead and conducted edits to the four below files in order to rid them of any privacy invasive code. Specifically, I have removed **metrics**/meta-data (M), **sensor** interaction (S), and extraordinary **permissions** outside the scope of the application (P). In addition to comments, I have included the edits below:

1. AndroidManifest.xml – Examining the code, there are number of references to obtaining location permissions (P) from the user. This is invasive and extraneous, as the app does not require location data to function.

```
10      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
11      <uses-permission android:name="android.permission.INTERNET" />
12
13  <!--This user information is invasive, requires permission to users' position (fine, coarse, and mock)-->
14  <!--
15      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
16      <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
17      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
18      <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION"/>
19  -->
```

2. UserInfo.kt – Removing with @Post("/api/metrics") is essentially removing an API call that gathers private data (via metrics) regarding the user (M).

```
10  interface UserInfo {
11  //This contains a metric gathering tool. Commenting this out will prevent this portion from gathering privacy invasive data..
12  //    @POST("/api/metrics")
13      fun postInfo(@Body info: UserInfoContainer, @Header("Authorization") token: String?) : Call<User>
14  }
```

3. CardScrollingActivity.kt – Please see 4. ProductScrollingActivity.kt
4. ProductScrollingActivity.kt – Both CardScrollingActivity.kt and ProductScrollingActivity.kt have very similar code, and therefore have very similar flaws when it comes to privacy invasive software. It has numerous mentions to metrics, sensors, and permissions. I have included screenshots below of changes made to CardScrollingActivity.kt. Very similar changes can be found on ProductScrollingActivity.kt; I have not included a second set of screenshots as they would be extremely similar. Please see the code file itself for the official changes.
   a. Lines 9-15: This is both (S) and (P) as sensors, location permissions are mentioned

```
8   //KZ: Removing any mentions of Sensors or Location to preserve privacy
9   //import android.hardware.Sensor
10  //import android.hardware.SensorEvent
11  //import android.hardware.SensorEventListener
12  //import android.hardware.SensorManager
13  //import android.location.Location
14  //import android.location.LocationListener
15  //import android.location.LocationManager
```

   b. Line 29 – Specifically (M) as there is an API call in this "import"

```
28  //KZ: Remove any API calls that contain user information.
29  //import com.example.giftcardsite.api.service.UserInfo
30  import retrofit2.Call
```

   c. Line 37 – a hidden mention of both Sensor and Location Listener (S), (P)

```
36  //KZ:  There is a mention of both a Sensor and Location Listener, both of which are note required for Card Scrolling.
37  class CardScrollingActivity : AppCompatActivity() { //, SensorEventListener, LocationListener {
38      private var loggedInUser : User? = null
```

d. Lines 41 & 42 – (S) as this interacts with the accelerometer

```
40  //KZ:  These sensors seem to interface with the accelerameter, collecting data based on a component of the phone.
41  //    private lateinit var sensorManager: SensorManager
42  //    private var mAccel: Sensor? = null;
```

e. Lines 48-55 –

```
47  //KZ:  Both Location and Sensors are mentioned in the below line of code. This matches the AndroidManifest.xml, including permissions for Fine locations
48  //        val locationPermissionCode = 2
49  //        var locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
50  //        if ((ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED)) {
51  //            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), locationPermissionCode)
52  //        }
53  //        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000, 5f, this)
54  //        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
55  //        mAccel = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
```

f. Lines 107-16 – Form one block of code referring to location, which is (P). Please note, the change from “http” to “https” is moot due to it being contained within privacy invasive code.

```
107 //KZ:  This entire grouping of code blocks from lines 103-173 utilize privacy invasive tools. Additionall comments within.
108 //
109 //    override fun onLocationChanged(location: Location) {
110 //KZ edit1: This is no longer necessary, as this section uses privacy invasive tools. KZ: Adding in "https" to replace "http"
111 //        var userInfoContainer = UserInfoContainer(location, null, loggedInUser?.token)
112 //        var builder: Retrofit.Builder = Retrofit.Builder().baseUrl("https://appsecclass.report").addConverterFactory(
113 //            GsonConverterFactory.create())
114 //        var retrofit: Retrofit = builder.build()
115 //        var client: UserInfo = retrofit.create(UserInfo::class.java)
116 //        client.postInfo(userInfoContainer, loggedInUser?.token)?.enqueue(object: Callback<User?> {
117 //            override fun onFailure(call: Call<User?>, t: Throwable) {
118 //                Log.d("Metric Failure", "Metric Failure in onFailure")
119 //                Log.d("Metric Failure", t.message.toString())
120 //
121 //            }
122 //
123 //            override fun onResponse(call: Call<User?>, response: Response<User?>) {
124 //                if (!response.isSuccessful) {
125 //                    Log.d("Metric Failure", "Metric failure. Yay.")
126 //                } else {
127 //                    Log.d("Metric Success", "Metric success. Boo.")
128 //                   Log.d("Metric Success", "Token:${userInfoContainer.token}")
129 //                }
130 //            }
131 //        })
132 //    }
133 //
```

g. Lines 136-161 – Another block of code referring to both (S) and (M), as both sensors and user-metrics are discussed. Again, please note, the change from “http” to “https”

is moot due to it being contained within privacy invasive code.

```
134  //KZ: This section deliniates the call backs and operations on user info in reference to a sensor. Again, we remove it to have privacy preserving code.
135  //
136  //    override fun onSensorChanged(event: SensorEvent?) {
137  //        if (event != null) {
138  //            var userInfoContainer = UserInfoContainer(null, event.values[0].toString(), loggedInUser?.token)
139  //KZ edit1: This is no longer necessary, as this section uses privacy invasive tools. KZ: Adding in "https" to replace "http"
140  //            var builder: Retrofit.Builder = Retrofit.Builder().baseUrl("https://appsecclass.report").addConverterFactory(
141  //                GsonConverterFactory.create())
142  //            var retrofit: Retrofit = builder.build()
143  //            var client: UserInfo = retrofit.create(UserInfo::class.java)
144  //            client.postInfo(userInfoContainer, loggedInUser?.token)?.enqueue(object: Callback<User?> {
145  //                override fun onFailure(call: Call<User?>, t: Throwable) {
146  //                    Log.d("Metric Failure", "Metric Failure in onFailure")
147  //                    Log.d("Metric Failure", t.message.toString())
148  //
149  //                }
150  //
151  //                override fun onResponse(call: Call<User?>, response: Response<User?>) {
152  //                    if (!response.isSuccessful) {
153  //                        Log.d("Metric Failure", "Metric failure. Yay.")
154  //                    } else {
155  //                        Log.d("Metric Success", "Metric success. Boo.")
156  //                        Log.d("Metric Success", "Token:${userInfoContainer.token}")
157  //                    }
158  //                }
159  //            })
160  //        }
161  //    }
```

h. Lines 163-180 – These final three individual sets of code have to do with (S), (S/M), and (P) respectively. All three are overrides that implement some factor of a sensor, metric gathering, or permission.

```
163  //KZ: This section refers to accuracy of sensors.
164  //    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
165  //        return
166  //    }
167
168  //KZ: THis section refers to the accelerameter, utilizing privacy invasive hardware
169  //    override fun onResume() {
170  //        super.onResume()
171  //        mAccel?.also { accel ->
172  //            sensorManager.registerListener(this, accel, SensorManager.SENSOR_DELAY_NORMAL)
173  //        }
174  //    }
175
176  //KZ: Finally, this section utilizes "Listener" which is a part of the sensor Manager. Anything related to sensors is taken out.
177  //    override fun onPause() {
178  //        super.onPause()
179  //        sensorManager.unregisterListener(this)
180  //    }
```