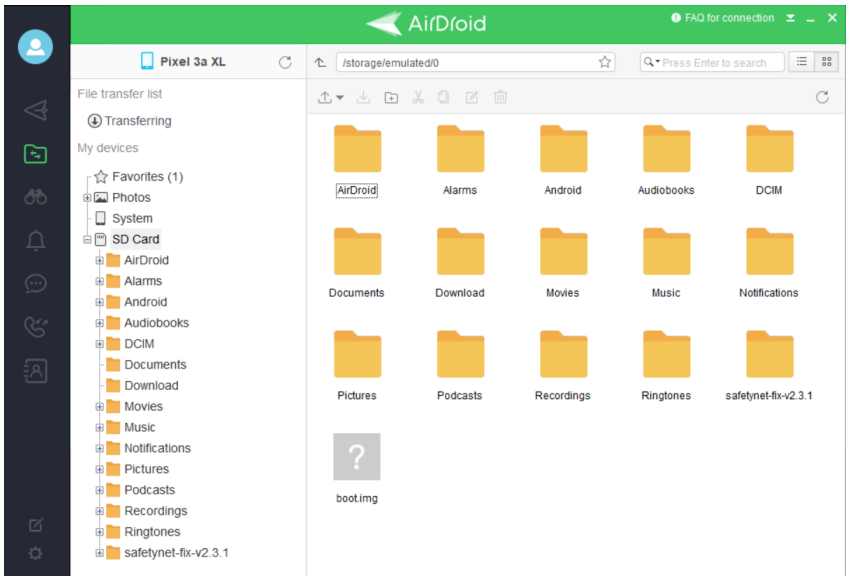
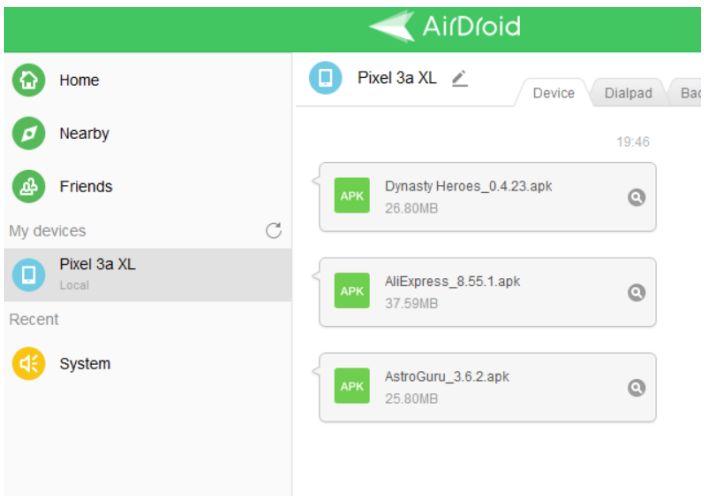
Forensic Analysis of Android Applications

Hands on Experiences using : AirDroid, Dex2Jar, and Android Studio

For round two of this project, I used various APK analysis software such as AirDroid, Dex2Jar, and Android Studio to analyze the APK files of the selected Android applications and see if there were any data vulnerabilities. For instance, some applications may not properly encrypt or otherwise protect sensitive user information. First, I installed AirDroid, a software that allows for



simple file transfer and exploration of Android devices, and I used this to transfer the APK files of the three applications I selected (AliExpress, Dynasty Heroes, and Astro Guru) onto my computer for further analysis. AirDroid only works on rooted devices, which is why I had to enable root access on my device in the previous round. After transferring the 3 APK files onto my computer, I zipped them and then installed Dex2Jar, which is used to decompile .dex files into readable classes. This way, I will be able to actually look through the files in a readable code format instead of trying to decipher uncompiled .dex files. After installing the software and updating the Paths,



```
C:\Users\pro_b\Downloads\dex2jar-2.0>cd dex2jar-2.0

C:\Users\pro_b\Downloads\dex2jar-2.0\dex2jar-2.0>d2j-dex2jar classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar
Detail Error Information in File .\classes-error.zip
Please report this file to http://code.google.com/p/dex2jar/issues/entry if possible.

C:\Users\pro_b\Downloads\dex2jar-2.0\dex2jar-2.0>
```

I then used the command prompt to transfer the .dex files into a .jar file of classes. Then, I installed another software, jd-gui, which is a Java Decompiler that presents the class files. Finally, I opened up the class files in jd-gui in order to analyze the APK. I repeated this process for each of the three APKs, combing through hundreds and hundreds of class files to discover any potential data vulnerabilities. First, I analyzed the APK file for AliExpress, a very popular Chinese e-commerce platform that has been known to host disreputable sellers. However, upon searching through the class files, I was unable to find anything compromising or otherwise unsafe for user information. I discovered one class function that seemed to store user information such as name and payment information into a hash table. Hashing is generally a secure way to store sensitive information, so although AliExpress may potentially host some sellers that are out to scam users, there appears to be nothing inherently unsafe about the application itself, from a user information standpoint.

```
a.add("member.deviceRegister");
a.add("invite.receiveNewUserCoupon");
a.add("member.accountActive");
a.add("order.placerechargeorder");
a.put("order.placeOrder", "1");
a.put("order.placeOrder_coin", "1");
a.put("payment.applyForPayment", "1");
a.put("task.exchangeRewardCoupon", "1");
a.put("MobileApiService.callApiWithTokenH5", "1");
a.put("MobileApiService.callApiWithoutTokenH5", "1");
a.put("order.orderConfirmEdit", "1");
a.put("coupon.sendCouponByAwardCode", "1");
a.put("wishlist.addToWishList", "1");
a.put("MobileMessageSeckill.seckillSubScribe", "1");
a.put("order.orderConfirm", "1");
a.put("gameAPI", "1");
a.put("member.register", "1");
a.put("member.login", "1");
a.put("member.loginandbind", "1");
a.put("member.snslogin", "1");
a.put("issue.cancelIssue", "1");
a.put("issue.createIssue", "1");
a.put("issue.agreeSolution", "1");
a.put("home.getFloorDataWithMtee", "1");
a.put("bricks.getLegacyFloorDataWithMtee", "1");
a.put("deviceInfo", "1");
a.put("coinGameService.doCoinTask", "1");
a.put("marketing.exchangeShoppingCouponWithCoin", "1");
a.put("marketing.obtainShoppingCoupon", "1");
a.put("Ugc.UgcGameMobileApi.receiveCoupon", "1");
a.put("Ugc.UgcLikeMobileApi.likeOrUnlikePost", "1");
a.put("member.deviceRegister", "1");
a.put("invite.receiveNewUserCoupon", "1");
a.put("member.accountActive", "1");
a.put("order.placerechargeorder", "1");
```

Next, I analyzed the APK class files of AstroGuru, an Indian horoscope application that also offers paid fortune telling services. After decompiling the .dex files and

opening them up in jd-gui, I was able to find some user information stored in one of the class files. This included basic information such as the user's date of birth, zodiac sign, email address, and country of origin. This information was found very easily and is accessible to anyone who has the necessary software and root access. Although this information is not inherently sensitive or dangerous, such as credit card or social security information, this information can still be used for dubious purposes. For instance, information such as email addresses, dates of birth and zodiac signs can be very useful for targeted advertising. I was also able to find this snippet of code that appears to store some payment information in a separate JSON file. Again, this may not be inherently malicious as it is often necessary for companies to keep track of payment information, but it seems to be a bit too open for such sensitive data. Finally, I analyzed the APK files of Dynasty Heroes, a typical gacha style fantasy RPG game. A gacha game is essentially a free-to-play game that features micro-transactions and heavily incentivizes users to purchase them in order to gain an edge in the game. The first curious thing I noticed about the class files is that there is quite a bit of code written in Chinese. I wasn't initially aware that Dynasty Heroes was developed by a Chinese company, and I thought it was intriguing to see both English and Chinese code being used together for this application. Although it is not inherently suspicious, it is interesting that the code relating to payment information is coded in

```
if (!getIntent().hasExtra("newLaunch") || !getIntent().getBoole
  String str1 = this.c.getString("displayName", "");
  this.c.getString("userPhotoUrl", "");
  String str2 = this.c.getString("Zodiac", "Libra");
  String str3 = this.c.getString("DateOfBirth", "01/01/1990");
  String str4 = homeActivity.country;
  getIntent().putExtra("displayName", str1);
  getIntent().putExtra("zodiac", str2);
  getIntent().putExtra("dateOfBirth", str3);
  getIntent().putExtra("country", str4);
```

```
jSONObject2.put("language", LocaleHelper.language);
jSONObject1.put("action_data", jSONObject2);
jSONObject1.put("transaction_id", this.b);
HttpTasksExecutor.executeAsyncTask("/matchmaking/getmatc
return jSONObject;
catch (Exception exception) {
new ExceptionHandler();
ExceptionHandler.handleException(exception, (Context)this
GAEventTracker gAEventTracker1 = new GAEventTracker();
Application application1 = (Application)this.c.a.getAppli
stringBuilder2 = new StringBuilder();
stringBuilder2.append("Continue to Pay ");
stringBuilder2.append(this.c.a.getLocalClassName());
gAEventTracker1.trackGAEvent(application1, "ClickEvent",
this.a = 1;
return jSONObject;
```

Chinese. Perhaps the developers wanted to add an extra layer of ambiguity for sensitive functions and operations in their source code in order to mask suspicious activity, but this is unlikely. I also found some code that described a payload in another class file, which I found surprising, as payloads are often used in malicious files. It did not seem that this payload code was created to do anything particularly harmful, but I still found it interesting and decided to include it in this report. Perhaps the payload is used for advertising or data collection on the user, as is often the case with gacha games where the goal is to make as much money from microtransactions as possible. Other that those two interesting code snippets I found, I was unable to discover anything else malicious or harmful in the class files, such as unprotected user information.

```
public static void paymentDefault(PaymentParam paramPaymentParam) {
  癎躑選燴.肌緭().肌緭(paramPaymentParam);
}

public static void requestPermission(Context paramContext, String paramSt
  if (縵襜繄锱丟鄹湇棉.肌緭(paramContext, paramString)) {
    paramPermissionCallback.onPermissionGranted();
    return;
  }
  冇絿鞿芚蒼恾潑炥竺鉼趙滾.肌緭(paramContext).肌緭(paramString, paramPermis
}

public static void roleLevelUpgrade(int paramInt) {
  癎躑選燴.肌緭().肌緭(paramInt);
```

```
AutoValue_Event(@Nullable Integer paramInteger, T paramT,
  this.code = paramInteger;
  if (paramT == null)
    throw new NullPointerException("Null payload");
  this.payload = paramT;
  if (paramPriority == null)
    throw new NullPointerException("Null priority");
  this.priority = paramPriority;
}

public boolean equals(Object paramObject) {
  if (paramObject == this)
    return true;
  if (paramObject instanceof Event) {
    paramObject = paramObject;
    return (((this.code == null) ? (paramObject.getCode()
  }
  return false;
}

@Nullable
public Integer getCode() {
  return this.code;
}

public T getPayload() {
  return this.payload;
}
```