Lab #9: Mobile Reversing
CSE 3801 : Introduction to Cyber Operations
Team: Tyler Dionne

Challenge 1:
Provided files: flag_100.apk
For this challenge my first thought was to 'strings' the file and see if the flag was present.
My thoughts were correct. Using the following command:

```
$ strings flag_100.apk | grep flag
```

The flag was revealed.

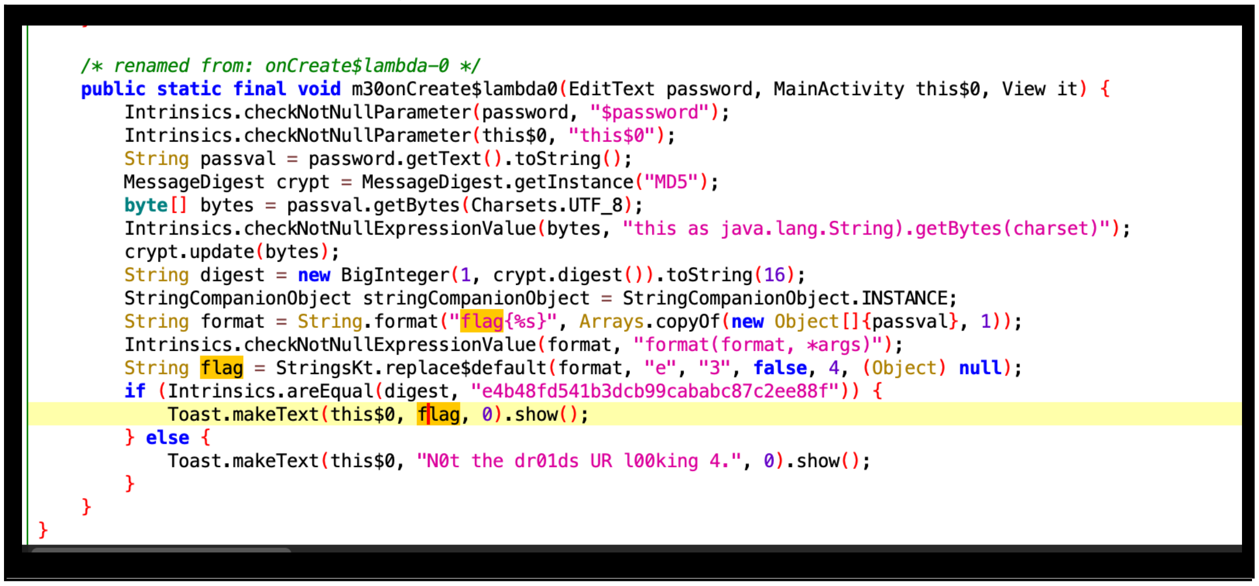
Challenge 2:
Provided files: flag_200.apk
For this challenge my first thought was to again try to 'strings' the file for the flag. After this did not work my next thought was to use foremost on the .apk file. Using the following command:

```
$ foremost flag_200.apk
```

Then navigating to the 'output' folder generated by this command I clicked on the 'png' folder and the flag was present on the first image in the folder.

Challenge 3:
Provided files: flag_300.apk
Assuming that neither of the previous methods would work for this challenge I had to think of a different approach.
The title of this problem was 'basecheck' which led me to the idea of searching through the .apk file for base-64 encoded strings that may be the flag.
Using the site 'dcode.fr' I used the base 64 encoding tool to encode the string flag which gave me the result 'ZmxhZw=='.
I then loaded the .apk file into jadx-gui where I did a search for this string. Although this entire string was not present I continued to backspace each character one by one until I found a match.
I found a match with 'ZmxhZ'.
Upon navigating to the match I found the string 'ZmxhZ3t0aDNyM19yX24wX3MzY3IzdHNfMl9zdWNjM3NzfQ==' which after using the base 64 decoder on dcode.fr I was able to recover the flag.

Challenge 4:
Provided files: flag_300.apk
For this challenge I was forced to find a different approach than the ones used in the three other problems. My first thought was to load the .apk file into jadx-gui and search for ‘flag{‘. This search resulted in an interesting block of code. See screenshot below:



```
    /* renamed from: onCreate$lambda-0 */
    public static final void m30onCreate$lambda0(EditText password, MainActivity this$0, View it) {
        Intrinsics.checkNotNullParameter(password, "$password");
        Intrinsics.checkNotNullParameter(this$0, "this$0");
        String passval = password.getText().toString();
        MessageDigest crypt = MessageDigest.getInstance("MD5");
        byte[] bytes = passval.getBytes(Charsets.UTF_8);
        Intrinsics.checkNotNullExpressionValue(bytes, "this as java.lang.String).getBytes(charset)");
        crypt.update(bytes);
        String digest = new BigInteger(1, crypt.digest()).toString(16);
        StringCompanionObject stringCompanionObject = StringCompanionObject.INSTANCE;
        String format = String.format("flag{%s}", Arrays.copyOf(new Object[]{passval}, 1));
        Intrinsics.checkNotNullExpressionValue(format, "format(format, *args)");
        String flag = StringsKt.replace$default(format, "e", "3", false, 4, (Object) null);
        if (Intrinsics.areEqual(digest, "e4b48fd541b3dcb99cababc87c2ee88f")) {
            Toast.makeText(this$0, flag, 0).show();
        } else {
            Toast.makeText(this$0, "N0t the dr01ds UR l00king 4.", 0).show();
        }
    }
}
```

This code shows that when the correct password is entered, the flag is displayed.
The next step was to find the correct password. The code shows that if I enter the password that corresponds with the MD5 hash ‘e4b48fd541b3dcb99cababc87c2ee88f’ then the flag will be displayed.
To do this I stored the hash in a file named hash1.txt and used johntheripper to crack the hash. Using the following command:

```
$ john --format=raw-md5 --wordlist=johnwordlist.lst hash1.txt
```

I found that the correct password was ‘elephant’.
I then downloaded Android Studio to emulate the application and upon entering the password ‘elephant’ and pressing the button labeled ‘dont press the button’ I was able to retrieve the flag.