IMPORTANT NOTE: PLEASE DO NOT UPDATE THE PHONES!

This lab is tested with Android-5.1 (which is installed on the Moto G phones). Things will probably work with Android-6 and Android-7 but we don’t know for certain, and we don’t want to spend time trying to roll them back to Android-5.1.

This is the final laboratory session in the Secure Programming course. Lab exercises are a key part of the delivery of this course. Your work in labs does not contribute towards the final assessment mark, but you are expected to take part to help your understanding of the material, which is tested in the final exam.

- **Guided introduction.** The lab will include a short introduction, giving some hints about the exercises. This will be delivered in groups, but please arrive on time to make sure you’ve seen it (or ask the demonstrators).

- **Working together is required.** We want to foster a supportive learning environment. Students who have prior knowledge or expertise are especially encouraged to work with others. Collaborating on the exercises may help you to think more deeply about the problems by discussing different aspects, as well as sharing existing knowledge. In this lab we are offering real phones for the exercise and we have a limited number available, so you will need to share!

- **Course staff will be on hand.** We will be here to discuss your progress and help with solving the problems. Detailed help will only be available during the timetabled labs.

- **Submit answers.** There are checkpoint questions in each exercise which you can answer to measure your progress. You may submit answers using the electronic submission mechanism or simply discuss them in labs. Submitting answers will allow us to give feedback. Marked submission will be attached to your marked coursework which will be returned at early April. See the last page of this handout for submission instructions.

As the work is intended to be completed in the lab and not take more of your time, there will be a short deadline for submissions. For this lab session, submissions will be accepted until **4pm, Monday 20th March**.
Mobile devices

At the start of the lab we will hand out mobile devices. The phones are freshly reset and you will need to set them up for development, by connecting them to the University of Edinburgh WiFi service. At the end of the lab we will collect the phones and they will be erased. **You should do a factory reset before returning the device to us to protect your own privacy and account details.**

1. Tap through all the initial setup. **Don’t sign into a Google account or give any personal information.** Turn off the collection of usage data. Connect to Wifi, we recommend using the University central or eduroam access points.

   **NOTE**

   If you are using central access point and see a big red lock on the login screen, you will need to open a browser and visit [www.ed.ac.uk](http://www.ed.ac.uk). The you will need to click **Advance → Allow proceeding to an insecure website.** Lastly, login with your DICE account to use the WiFi service.

2. Enable developer mode. Go into **Settings**, then **About phone** (it is at the bottom). Repeatedly tap the **Build number** until developer mode is unlocked (5-6 taps should be enough).

3. Enable USB debugging. Go into **Settings** and **Developer options**.

The DICE machines have been set up to allow you to connect these phones as USB devices to use for development. Your own phone may not be able to connect to the DICE machines — but you may of course use your own laptop or PC.

**Important note: Photo sharing**

This lab lets you upload shared photos which are publicly accessible on an external server. Please **do not** include any private information in photos and certainly not images which are in any way personal, indecent or containing obscenity. We suggest sticking to pictures of objects or (with permission) faces around the lab. The images may be deleted periodically and the server will be taken down after 20th March.

To see this handout online, visit [http://www.inf.ed.ac.uk/teaching/courses/sp/2016/labs/lab3/](http://www.inf.ed.ac.uk/teaching/courses/sp/2016/labs/lab3/)
Introduction

In this lab we’re going to look at privacy problems inside an Android app. The app takes the form of a photo sharing service: users can upload photos to a photo sharing site, http://infr11098.space via an app on their phones. Unfortunately the app has some privacy and security issues, which we’re going to fix.

Android Studio and Code

We’re giving you a script to install Android Studio and get things set up on the local machine. It is going to install some files to /tmp/$USER-secprog-lab3. Any changes to files in here won’t be saved on other machines so make sure you backup your work as you go!

Run the script:

bash /afs/inf.ed.ac.uk/group/teaching/module-sp/secprog_lab3_install.sh

When Android Studio eventually starts:

Note Step 1 – 4 may not appear because some of the dice machine has already been configured before

1. Click on “I do not have a previous version…” and “OK”.
2. Ignore and dismiss the messages about OpenJDK and IBus. Click “Next”.
4. Pick any color scheme you like.
5. If the page shows, “No Android SDK found.”, click “Next”.
6. Set the “Android SDK Location” to /tmp/username-sp-lab3/AndroidSdk and click “Next” (where username is your username). You can ignore the red warning saying “An existing Android SDK was detected. The setup will…”.
7. Click “Next”, and finally “Finish”.
8. Immediately cancel the install.
9. Click “Finish”.
10. Click “Open an existing Android Studio project” and type the path /tmp/username-sp-lab3/secprog_lab3_app_release (where username is your username).
11. When it asks you for the SDK installation path type /opt/android/AndroidSdk.
12. Wait some more. It should after a minute or two report that Gradle is building.
13. Wait for the indices to finish building. When its done you’ll be able to build the code and go.
0. Thinking about privacy and confidentiality

Consider the design of a photo-sharing application which allows people to upload photos in an unrestricted way to a public website.

Hints: consider the

- Privacy By Design process see https://www.ipc.on.ca/english/Privacy/Introduction-to-PbD
- The 2013 FTC report on Mobile Privacy Disclosures, at: https://www.ftc.gov/reports/mobile-privacy-disclosures-building-trust...
  ...through-transparency-federal-trade-commission
  (especially Appendix B).

Checkpoint 0. What ideas would you propose for a privacy policy for such an application, and how would you inform your users about it and enforce it in your code?

1. Metadata and privacy

Let’s look at the app that has been written so far. In Android Studio:

- You can find the source code in the Project tab (far left hand side).
- App then java then uk.ac.ed.inf/secureprogramming then MainActivity.
- You can build and run the code through the Run menu and Run ‘app’ (Shift F10) or the Debug ‘app’ (Shift F9) items.
- Set and remove breakpoints by clicking in the left hand code margin.

Try taking a few pictures on the app. You should find they get uploaded to http://infr11098.space. If you click on any of the pictures you will see all the metadata embedded in the photos.

Checkpoint 1. Where did the metadata come from? Is any of the data concerning from a privacy perspective?

Checkpoint 2. Pick a bit of metadata and describe how it could be used to attack, or deanonymise a user.

Task: Some of the metadata has come from the EXIF tags embedded in the image. Modify the app to strip the EXIF tags. To do this you can use the android.media.ExifInterface classes.


Inside the metadata there is also GPS information which says where the user was when they took the picture.

Checkpoint 3. How accurate is the data? Where do the coordinates say the pictures were taken?
2. Transport Security

By default the app uses HTTP to upload images. This can be dangerous as there is no encryption between the app and the server the images are uploaded to, so traffic could be intercepted. Luckily the server also offers upload via HTTPS.

You can fetch the servers certificate with the command:

```sh
openssl s_client -showcerts -servername infr11098.space -connect infr11098.space:443 < /dev/null
```

Your web browser can also display the certificate. Try clicking the lock icon in the address bar of Firefox. If you are using Chrome, Click Options → More Tools → Developer Options → Security Tag

Checkpoint 4. Describe the size and format of the key. What is the certificate’s chain of trust? For what sites is the key valid?

Checkpoint 5. Many browsers have the certificate authority’s certificates built in. In Firefox they can be found in the Certificates tab under Advanced in about:preferences. Find the CA certificates used to sign infr11098.space.

Task: Modify the app to use the HTTPS protocol if the server address requires it. If an error occurs you should stop and report the message to the user. You might like to look at the documentation for the HttpsURLConnection and HttpURLConnection.


Lookup the IP address of the infr11098.space site:

```sh
dig infr11098.space +noall +answer
```

Try using the IP address directly as the server address.

Checkpoint 6. Does the connection still work? If not why not?
3. Pinned Certificates

Sometimes HTTPS isn’t good enough. In a *man-in-the-middle* attack an attacker may set up a fake server that forwards any requests sent to the real site but logs the data going between the user and the server.

There are a couple of ways an attacker could implement this. A simple approach might be to buy a very similar domain, such as

http://infr11098.space

and hope the user doesn’t spot the difference. Phishing attacks have often used this technique (whether or not connecting through to the intended server). Alternatively, an attacker might set up a fake access point and redirect traffic from the real site to their fake ‘one.

**Checkpoint 7.** Think about the two attacks, and the work required for an attacker. What certificates would the attacker need for their deception to work with HTTPS? How might the user notice they were being attacked?

**Task:** Implement *certificate pinning* for the app and the server. To do this you will need to implement a custom TrustManager. You should check that the certificate is the one current one, and that the hostname is infr11098.space. See this article, and the following documentation.

- https://developer.android.com/training/articles/security-ssl.html#Pinning

To help you implement this try using the server https://infr11098.space:444/. It has a valid signed certificate on it, but this certificate should *not* be accepted by the app as it is not the pinned one.

**Note:** Static analysis tools like Mallodroid can detect some common errors with SSL certificate verification. You might like to have a go at getting it running. In 2014 researchers tried Mallodroid was run on over a million Android apps. Around 2% were found to have some SSL issues and many CVEs were issued. See:

- Mallodroid on Github: https://github.com/sfahl/mallodroid

The checking of certificates is a difficult matter... (Bonus)

Certificates often cost money to have them signed by a CA, although Let’s Encrypt and other initiatives are changing these for the lowest assurance category of certificate (which validates only ownership of an email address or server connected to a domain name).

A cheaper alternative, often used by thrifty organisations, is for the site owner to sign the certificate themselves and have users trust the site owner by checking the certificate manually or installing the issuer as a CA. This is particularly common in the internal networks of businesses.

Sometimes app developers use self-signed certificates so they can change the certificates used by their app and it’s servers quickly and at low-cost. Instead of forcing a user to install a new certificate they can embed their own certificates in the app. The code to implement this is similar to the code for certificate pinning. On updates to the app, they can provide a new set of certificates to check against, and revoke old certificates as required.

**Checkpoint 8.** Consider the security issues involved with implementing custom certificate managers and checking code, and the temptation to “just get things working” by implementing workarounds. There is a lot of bad advice around on blogs and so on which naive developers often find by searching, and then follow. Try to find some good and bad examples of certificate checking and explain why they do (or do not do) the right thing.
Submission instructions

Download a copy of the text file checkpoints.md from

http://www.inf.ed.ac.uk/teaching/courses/sp/2016/labs/lab3/checkpoints.md

and edit it to insert your answers. Submit it with the command:

submit sp lab3 checkpoints.md

We will give some feedback through Piazza.

Rui Li, Arthur Chan and David Aspinall, March 2017.
Thanks to Joseph Hallett for lab design and coding.