All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

Common Android Bugs Pt. 2

Path Traversal

Applications often download files to their private data directory. If this is controllable by an attacker and not properly sanitized, path traversal vulnerabilities can occur.

```
FileWriter writer = new FileWriter(
    Context.getApplicationInfo().dataDir + "/downloads/" + filename
);
```

Preventing this bug is straightforward:

1. Avoid using external input for direct path construction at all costs
   a) Consider hashing file contents to produce filenames
2. When you need specific filenames, remove any slashes or backslashes to prevent path manipulation

Zip Path Traversal

Zip files contain a table full of entries mapping file path to contents

- something.txt → bytes 0x100-0x1000
- subdirectory/another.txt → bytes 0x1000-0x2000
- …

Unzipping just takes a destination path and concatenates the supplied path from the entry, and decompresses the data there.

A malicious zip file can contain entries with path traversal in the names, though:

- ../../../../../etc/passwd → bytes 0x100-0x1000
- …

Naive unzipping code will simply write the files where the zip entries tell it to write, triggering the bug.

The *evilarc* tool will automate construction of such archives for you.

In your own code, it’s best to detect any instance of ../ or ..\ in zip file entries and bail out.

Attempting to remove the path traversal is likely to lead to additional bugs and no legitimate archive will contain these.

Embedded Secrets

When you decompile an app, you’ll often see:

- Symmetric crypto keys
- Private keys
- HMAC keys

Sometimes these pose a real risk to the security of an application, but they’re often perfectly fine. How can you tell the difference?

Likewise, you may see access keys for third-party services. When you see these, determine what service is in use and find all the documentation you can.
Figure out what access can be gained and whether or not it’s required for the application to function.

Once you’ve collected all of this, you need to ask one question:
*What can an attacker **do** with this information?*
Can you access personal data? **Bug.**
Can you use the application as intended? Not a bug.
This question tells you everything you need to know.

Developers need to tread softly when it comes to embedded secrets. Assume that anyone and everyone will see them and understand what can be done.
Does it absolutely need to be present?
Can you reduce the access the secret provides?

Oauth Implicit Grants

Implicit grants are a concept in Oauth to support single-page applications. The browser makes a request like:
https://site/auth?clientid=...&redirect_url=com.myapp.oauth&response_type=token&scope=all
The token response type means that after auth, it will redirect to:
com.myapp.oauth#access_token=<authorization>&…

Given that the implicit grant provides plenty of options for interception, this should not be used by developers any longer. Instead, use the standard Oauth authorization flow, via *response_type=code.*
Also consider using platform-integrated authorization systems, like Google Sign-In and Facebook Login.

Oauth Redirect Hijacking

As we saw, *redirect_uri* controls the eventual recipient of the authorization code/token. If this can be controlled by an attacker and isn't properly validated, this can be compromised by a malicious application.
Looking for routes to control portions of the Oauth authorization query string can often lead to discovering these bugs.

Developers should hard-code the *redirect_uri* parameter whenever possible, as well as validating it against known-good values on the server side, if at all possible.
As before, platform-integrated authentication systems shift burden away from your application and should be considered as well.

GPSRP

The Google Play Security Rewards Program provides good bounties for the issues covered in this video (PDF). If you're interested in honing your skills and building up your experience, this is an excellent route: https://hackerone.com/googleplay