A **practical** approach to the mobile security **ecosystem** for Android

Stanford CS155

Chris Steipp
Security Partner, WhatsApp
https://www.linkedin.com/in/chrissteipp/

Meta

Agenda

# whoami

- I live in Portland with my family and rescue dog
- Security Partner for WhatsApp
- Formerly:
  - First Security Engineer at Wikimedia Foundation
  - Started AppSec program at Lyft
  - AppSec at Facebook

Who are you?

- Software Engineers or Computer/Data Scientists?
- InfoSec or Security Research Focus?
- Business or Product Design?

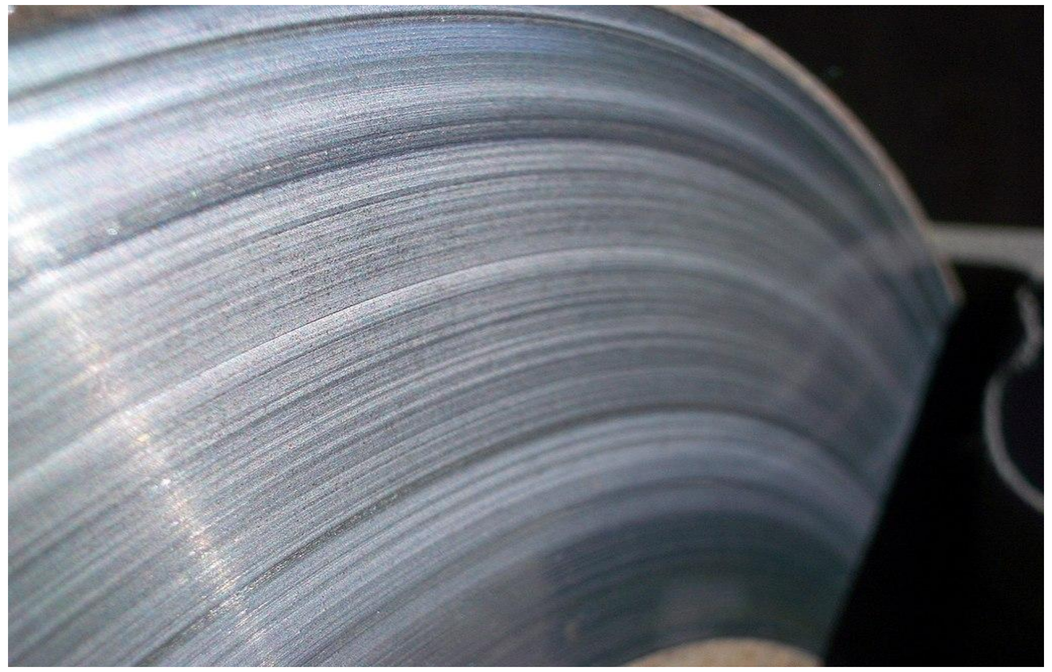
The elephant in the room...

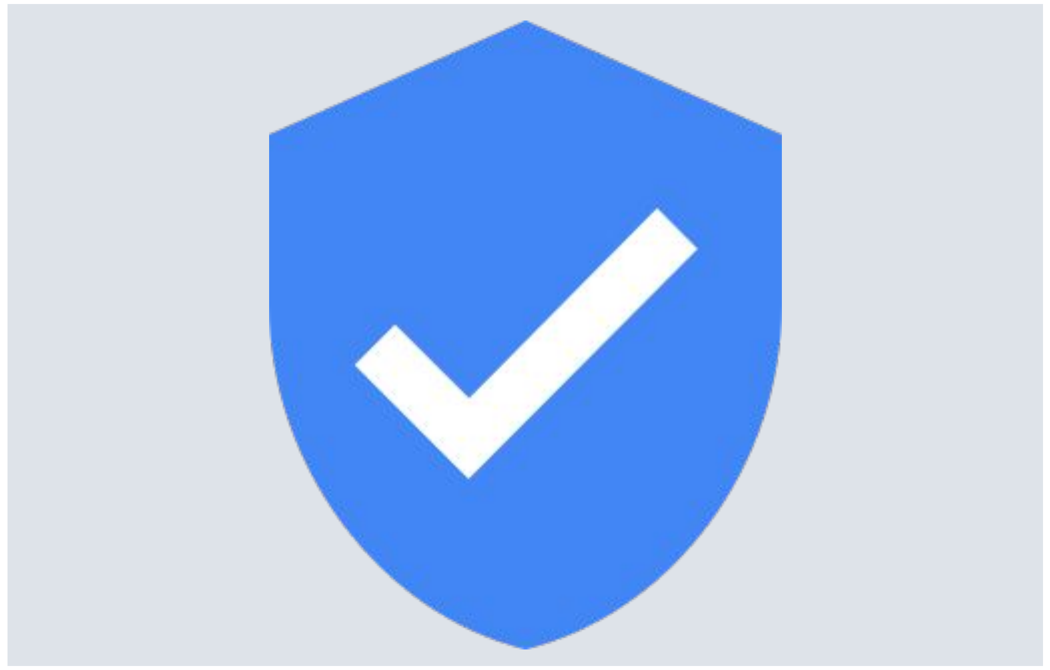
WhatsApp was exploited by NSO. Why should we listen to you?

02 Approach: Holistic Security

The security of android apps will depend as much on organizational culture and engineering practices, as it does on how you secure specific components.

Why do we care about Mobile App Security?





Organizational Security

- Security Culture
- Security of endpoints, network, build infra
- Compliance programs
- Detection programs

Secure Development

- Engineering practices that promote security
- Gates / Checkpoints / Paved Roads

Mobile App Security

- Specific practices and controls for Mobile App development

Organizational Security

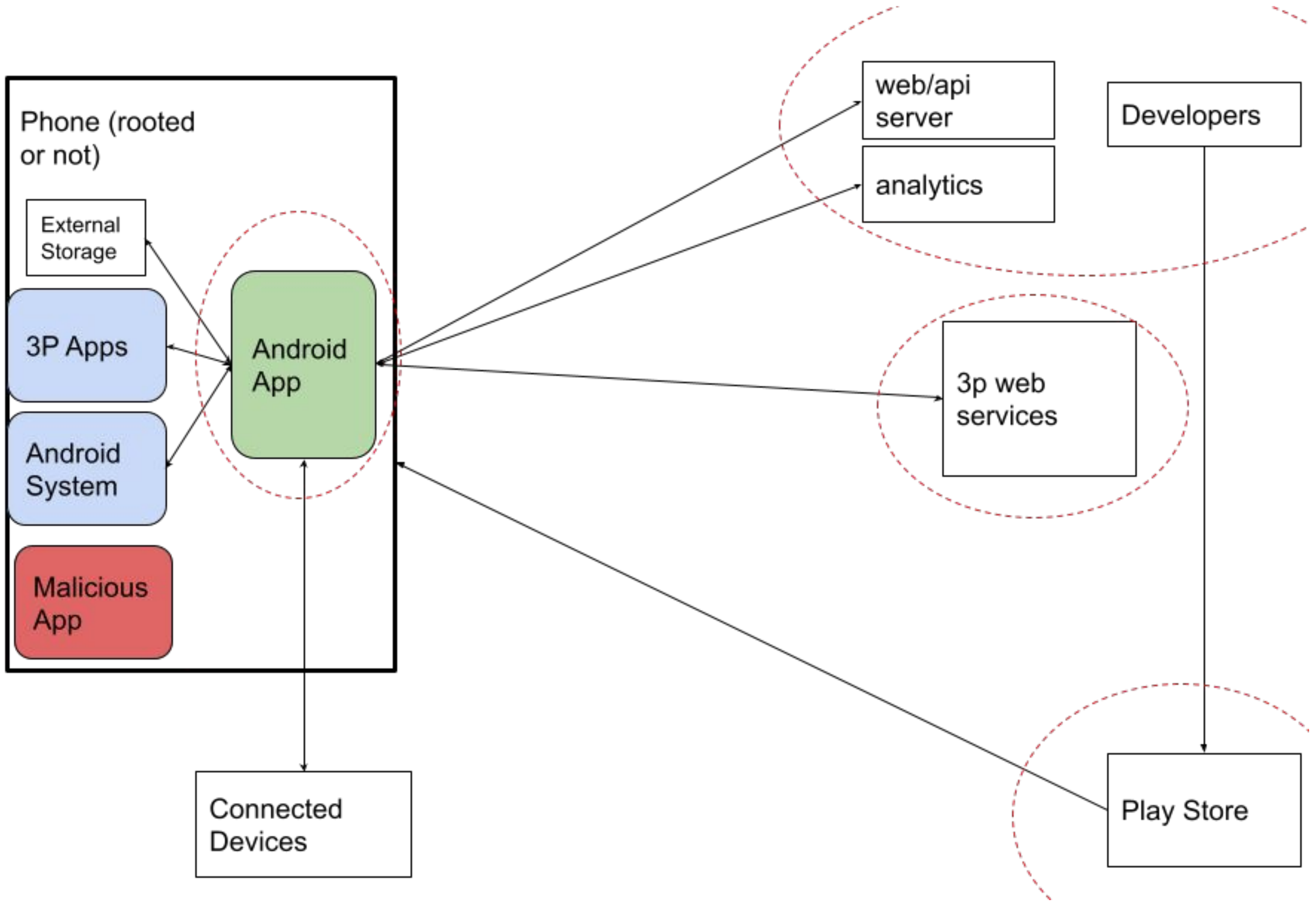
Secure Development

Mobile App Security

All good places for building controls too!

03 Risks

Threat Model of an Android App



Where to find lists of risks

- OWASP Top 10 (Mobile)
- Android Security Best Practices
- CWE
- Company Top 10

OWASP Top 10 for Mobile (2016)

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality

Android App security best practices

- Enforce secure communication
  - Intents
  - Re-authentication
  - traffic encryption
- Use WebView objects carefully
  - HTML channels
  - Javascript interface support
- Provide the right permissions
  - Intents
  - data sharing across apps
- Store data safely
  - Internal storage
  - external storage
  - shared files
  - validity of data
  - cache files
  - SharedPreferences
- Keep services and dependencies up-to-date
  - Google Play
  - app dependencies

Risks we’re going to look at in depth today

- Intents & IPC
- Web Views
- Native Code
- Authentication & AuthZ
- Logging Private Data
- Managing Features
- App Authenticity
- File Access

Intents & IPC

Android apps have many components, which need to talk to each other. Android makes accidental exporting and exporting to the wrong external apps easy.

Deeplink Open Redirects (“Android Nesting Intents”)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   // called with myapp://?callback=app%3A%2F%2Ffoo
   Intent incomingIntent = getIntent();
   Uri u = incomingIntent.getData();
   String redirect = u.getQueryParameter("callback");
   Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(redirect));
   startActivity(intent);
}
```

Solutions:

- Check callback
- Pending Intents
- Don’t use this pattern

Deeplink Open Redirects (“Android Nesting Intents”)

Shift Left:

- The problem is attacker controlled data being used to construct an intent, which is then called from a privileged context
- Static Analysis can be used for taint tracking

Mariana Trench:

- Argument(0) to a class inheriting from "Landroid/content/Intent;" <- Source
- Argument(0) to method with signature "Landroid/content/Intent;\\.parseUri:.*" <- Sink

Unauthorized Interception of Implicit Intents

```
Intent intent = new Intent();
intent.setAction("com.myapp.POST");
intent.putExtra(MyApp.MYAPP_MESSAGE_INTENT,
MyApp.getMessage());
context.sendBroadcast(intent);
```

Solutions:

- Explicit Intents
- Signature Permissions for Intents

Unauthorized Interception of Implicit Intents

Shift Left:

- We can setup functions returning “sensitive” data as sources, or annotations
- We can look for “features” of our data flows for explicit intents
    - Methods “setClass.*”, “setComponent”, or “setPackage” called in a class inheriting from Intent
- Alert on sensitive data -> intent launches that aren’t explicit

Web Views

Web views in the app have slightly different security properties than a full browser, and developers have many options for how they implement web views. Some options are more or less secure, depending on the use case.

Running Javascript, XSS, and Callbacks

```
webview.executeJavaScript(getJSForCallback());

webView.addJavascriptInterface()
```

Solutions:

- "Best practices" such as,
  - *If your application doesn't directly use JavaScript within a WebView, do not call setJavaScriptEnabled()*
  - *confirm that WebView objects display only trusted content*
- Security Reviews

Running Javascript, XSS, and Callbacks

Shift Left:

- Possible Static Analysis / taint tracking, but tracking across multiple languages is hard
- Frameworks
- Developer Training

Unreviewed code running in your app

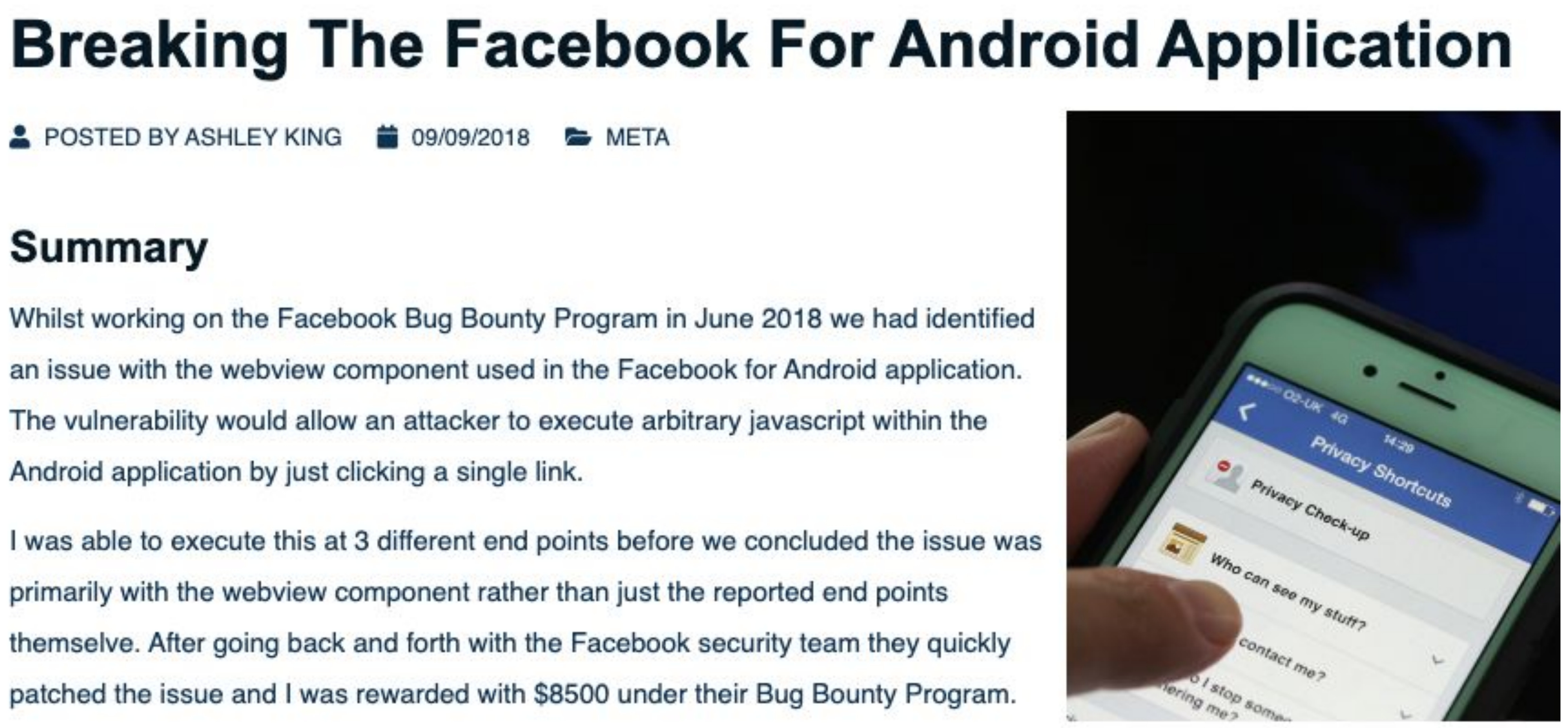
- Know your threat model
- Make sure other development teams in your org have consistent security standards
- Defensive coding for any callbacks

Intents + Webview for Profit

Problem:

- Intent included a url preview link
- Link wasn't properly sanitized
- Link was opened in a webview

```
adb shell am start -a
"android.intent.action.VIEW" -d
"fb://ig_lwicreate_instagram_account_full_sc
reen_ad_preview/?adPreviewUrl=javascript:c
onfirm('https://facebook.com/Ashley.King.U
K')"
```



Breaking The Facebook For Android Application

POSTED BY ASHLEY KING 09/09/2018 META

Summary

Whilst working on the Facebook Bug Bounty Program in June 2018 we had identified an issue with the webview component used in the Facebook for Android application. The vulnerability would allow an attacker to execute arbitrary javascript within the Android application by just clicking a single link.

I was able to execute this at 3 different end points before we concluded the issue was primarily with the webview component rather than just the reported end points themselve. After going back and forth with the Facebook security team they quickly patched the issue and I was rewarded with $8500 under their Bug Bounty Program.

Intents + Webview

- Benefits of a Bug Bounty programs
    - Plug: https://www.facebook.com/whitehat

Logging Private Data

When developing a product at scale, developers need visibility into errors. Your app will likely be logging app metrics, statistics, and error conditions.

- Android Vitals
- UncaughtExceptionHandler
- etc.

Logcat

Problem:

- Android <4.1 allowed apps to read log files of other apps
- Sending app logs to your server

Solutions:

- Don’t log anything sensitive

Logging Sensitive Data

Shift Left:

- Static Analysis for tracing sensitive data to logging sinks
- Runtime analysis:
    - Collect logs from integration testing & QA, look for test cases' sensitive data
    - Guided fuzzers, and grep for sensitive data in the logs (WA's FAUSTA)

Running Native Code

When developing a product at scale, developers need visibility into errors. Your app will likely be logging app metrics, statistics, and error conditions.

- Android Vitals
- UncaughtExceptionHandler
- etc.

"Out of the 58 in-the-wild 0-days for the year, 39, or 67% were memory corruption vulnerabilities. Memory corruption vulnerabilities have been the standard for attacking software for the last few decades and it's still how attackers are having success."
- Project Zero

Running Native Code

How do we do this?

- Android NDK

What are the risks?

- All the usual memory corruption risks. E.g., CVE-2019-3568, a buffer overflow in WhatsApp's VOIP packet handling

Running Native Code

Preventions at Scale:

- Anti-exploitation compiler flags
- Fuzz Everything*
- Migrate to to safe languages**

* - Setting up JNI objects in your test harnesses is non-trivial, usually manual work

** - Developer skillset/culture, build processes, dependency management, release process, static analysis tools, dependency vulnerability detection tools

Authentication & Authorization

“Poor or missing authentication schemes allow an adversary to anonymously execute functionality within the mobile app or backend server used by the mobile app.” - M4: Insecure Authentication

“Poor or missing authorization schemes allow an adversary to execute functionality they should not be entitled to using an authenticated but lower-privilege user of the mobile app. Authorization requirements are more vulnerable when making authorization decisions within the mobile device instead of through a remote server. This may be a requirement due to mobile requirements of offline usability.” - M6: Insecure Authorization

Multiple Users Same App

Problem:

- Device sharing is common in many communities
- Each user’s data is meant to be private
- We need to efficiently cache and store data, and segment the app’s sandbox.

Solutions:

- Understand your product requirements and user expectations.
- Local file encryption with per-user keys

Making PINs into Keys

Problem:

- “The form factor highly encourages short passwords that are often purely based on 4-digit PINs.” (OWASP, M4)
- Long, complex passwords are hard for users
- In WhatsApp, there are situations where users lose data if they lose their password, e.g. End-to-end Encrypted Backups.

Solutions:

- Fleet of HSMs that do key agreement based on a short password or PIN
  - HSM enforces brute forcing limits
  - HSM isn’t upgradable, so limits can’t be changed

File Access

Android has internal and external storage as well as cache files.

File Access

What can go wrong:

- WhatsApp CVE-2021-24027: Cached TLS sessions were readable by other apps

Solutions:

- Store private data within internal storage
- Check validity of data
- Store only non-sensitive data in cache files

Feature Management

Mobile engineering teams will often implement the ability to roll out features to only a cohort of users for A/B testing or artificially slowing rollout. The “hidden” features can be reverse engineered and enabled with tools like Frida.

Security teams often need to disable specific features without waiting for users to update their apps.

Feature Rollout

- User Education
- Code Obfuscation

Killswitches for Features

It’s far less painful to turn off a single feature than force-update many users.

Intelligent killswitches (platform-capability dependant, or operating on a regex of attacker controlled input) can be very useful.

App Authenticity

Your servers only see bytes on a wire. Developers are often surprised that attackers will reverse engineer the client protocol and communicate with your backend servers via a fake client. Fake clients don't have to send correct data, or handle data you give it appropriately.

Attestation

- Playstore attestation is useful, but not available on rooted devices and spoofable on older API versions.

04 Wrap Up and Q&A

Meta

Summary

- Your Android App's security is impacted by the entire ecosystem of your organization's security.
- This was a (very) quick overview of current threats and controls for securing Android apps. The specifics will change by the time you graduate. The threat model is slowly evolving. But hopefully I've communicated the principles so you can work these out in the future.

Image Credits

- (slide 3) Owner
- (slide 5) Basile Morin, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons
- (slide 8) Brakes: Richardmilgate (talk) (Uploads), Public domain, via Wikimedia Commons
- (slide 8) Doughnuts: Dave Crosby, CC BY-SA 2.0 <https://creativecommons.org/licenses/by-sa/2.0>, via Wikimedia Commons
- (slide 8) Checkmark: Google inc, Public domain, via Wikimedia Commons
- (slide 11) Auckland Museum, CC BY 4.0 <https://creativecommons.org/licenses/by/4.0>, via Wikimedia Commons