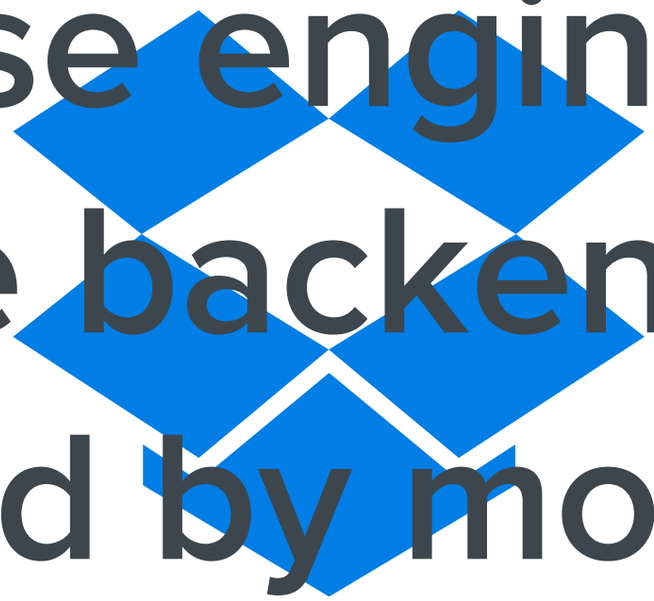


Reverse engineering private backend APIs used by mobile applications this title is too long...

Chris Varenhorst

chris@dropbox.com, @varenc

REPBAUbMA

Chris Varenhorst

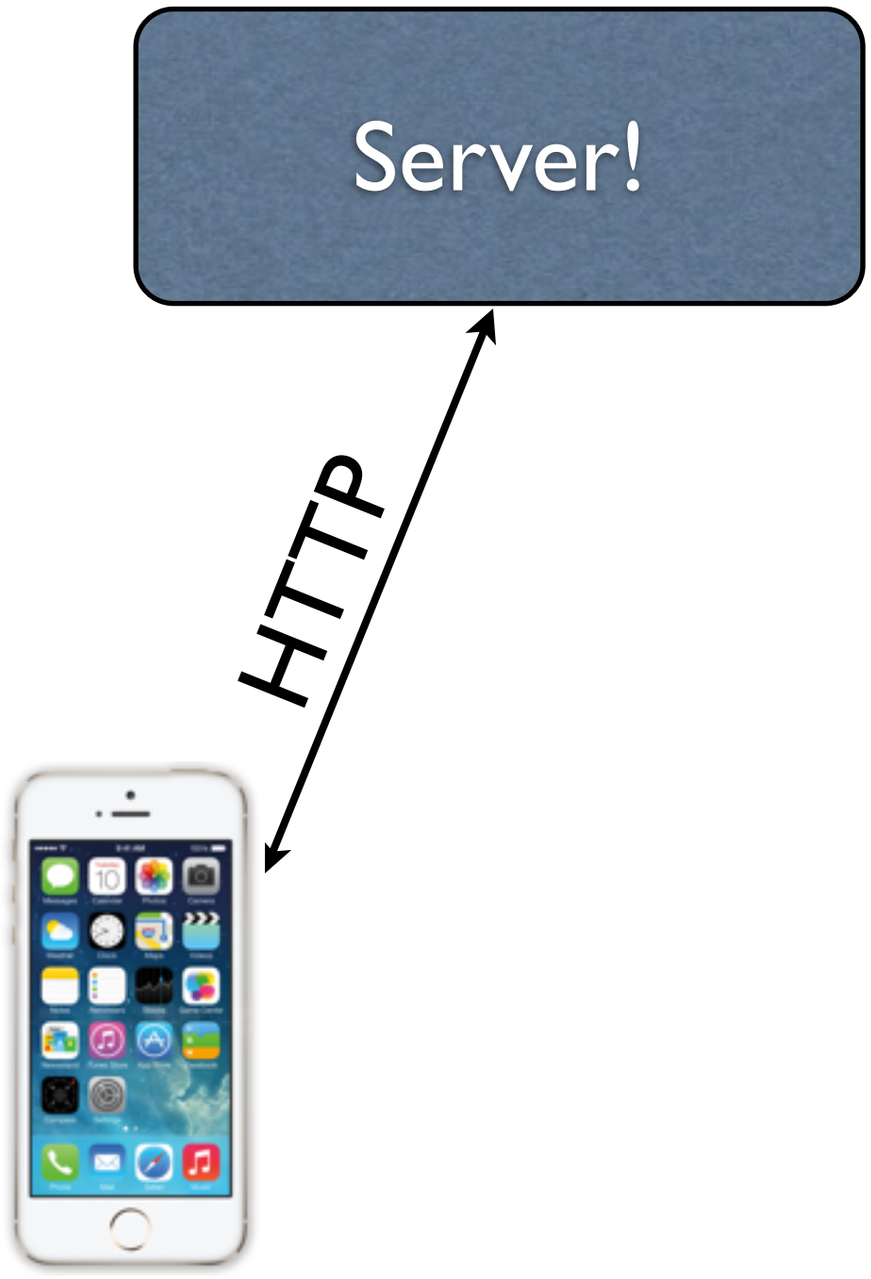
chris@dropbox.com, @varenc

Today's Talk

- Backend APIs used by mobile apps
- Common design problems
- Demo! Reversing by observation (iPhone)
- Demo! Reversing by decompiling (Android)
- Designing APIs right/mitigation techniques
- QnA

Backend APIs

- How the app talks to the server
- In just about every popular mobile application
- Usually HTTP[S]



Common design flaws

- Trusting the client too much...
  - Not enforcing access control
  - No abuse mitigation strategy
  - Example: Snapchat contacts lookup

Common design flaws

- Unintentionally exposing too much data.
- Example: <undisclosed ride sharing company>

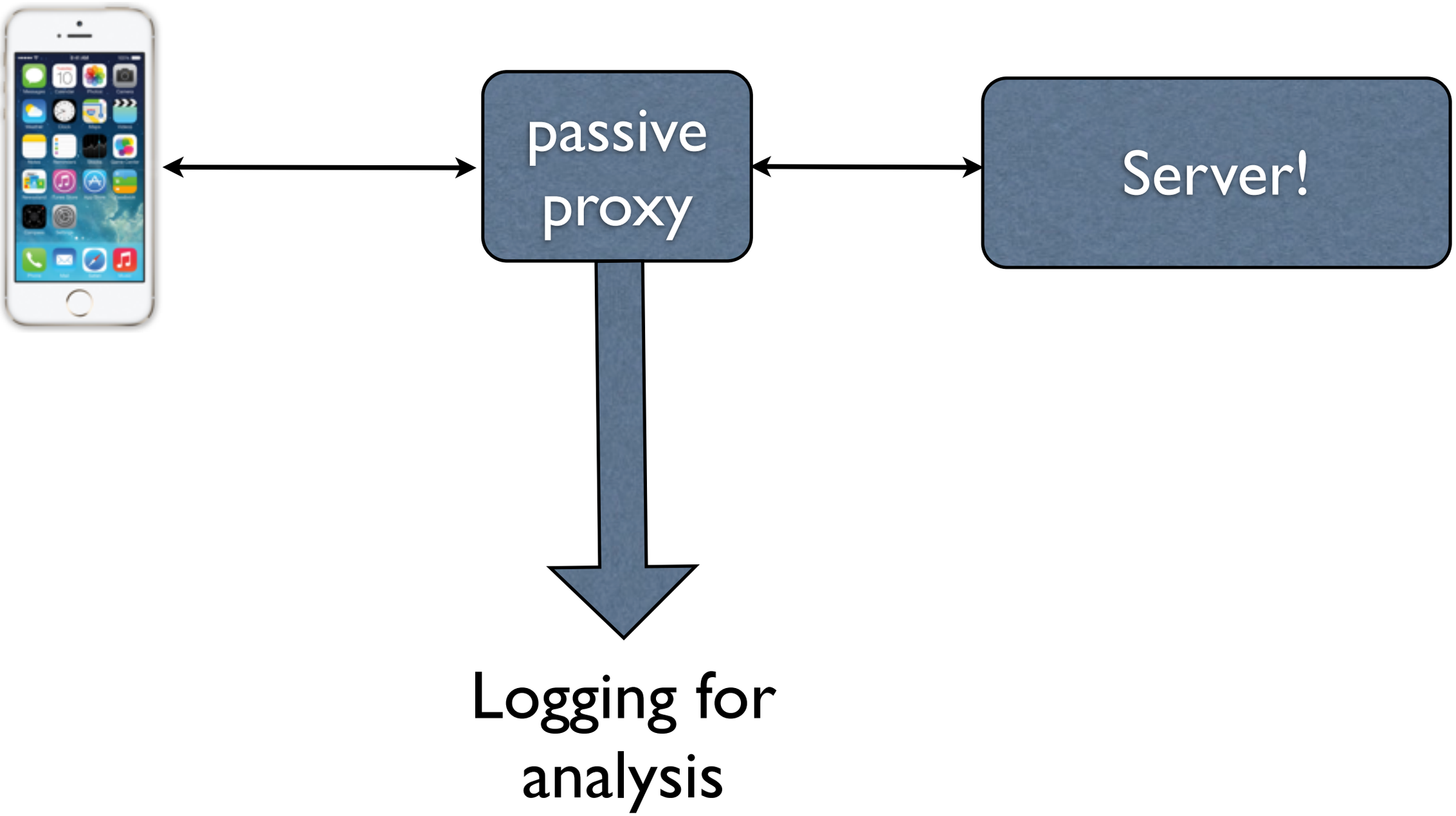
Reverse engineering...

- Understand the backend API
- Impersonate legitimate clients.
- Sometime just snooping itself is interesting

Challenging

- Don’t have the source code
- HTTPS - can’t intercept the data
- Extra layer of encryption or signing
- Certificate pinning
- Obfuscation of code (or API)
- Non-HTTP[S]

Easysauce: HTTP Interception



What about HTTPS?

- HTTPS (TLS/SSL + HTTP) was built to protect from this
- Relies on trusted Certificate Authorities
- But we can our device trust anything...

Easysauce: HTTPS Interception

FAKE cert

ACTIVE proxy

REAL cert

Server!

install fake Certificate Authority

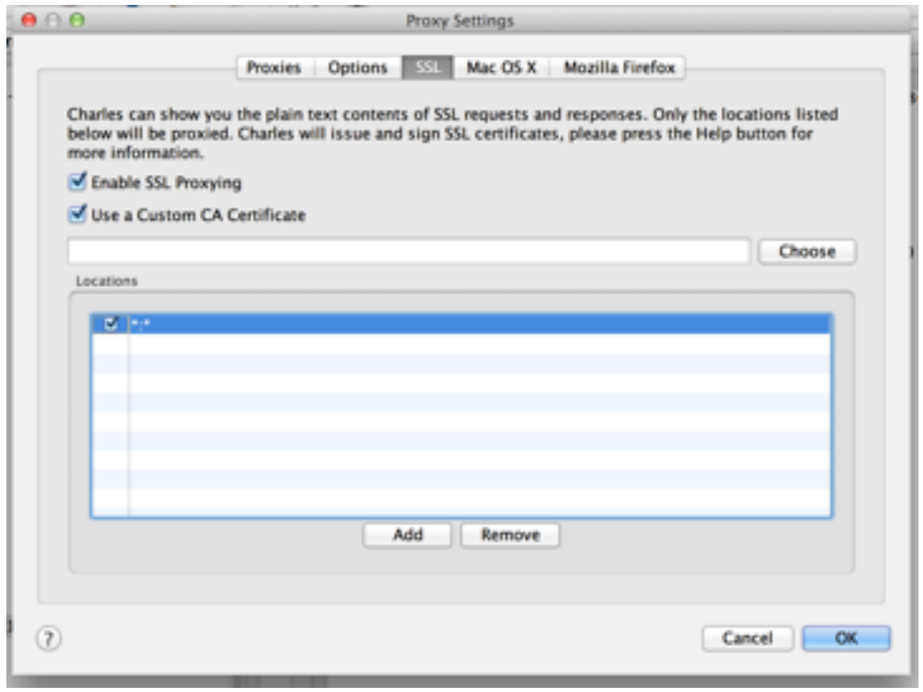
Logging for analysis

The tools

- My HTTP[S] proxy of choice: Charles
- One iPhone (much harder on Android)
- One “fake” Certificate Authority

Observing calls (iPhone)

- Setup Charles and install its root CA on your iPhone Guide #1, Guide #2 (BE CAREFUL)
- Enable SSL interception
- Add device IP to access control
- Finally, input the proxy in your iPhone. Guide.



Demo Time

Inspecting apps directly

- That was fun! but can only take us so far...
- Sometimes you'll need to dive into the application itself
- Easiest on Android
  - JVM code, easily rooted

Lazy strategy

- Insight: groking dalvik JVM code is hard. Logging is easy.
  - Decompile apk application
  - Find something interesting
  - Add logging calls
  - Recompile, install, and run.
  - Watch the logs

Setup

- Optional: Install genymotion emulator
  - Download android image
  - Follow this guide to install google play (this is somewhat naughty)
  - Or use a real device in debug mode
- Install Android SDK
  - Will use the adb tool extensively

Setup continued...

- Install apktool
  - Amazing tool for decompiling APKs
  - Outputs JVM code in smali format
  - Allows you to recompile as well

Demo Time

Inspecting apps directly

- Checkout this guide. Key points
  - Use `adb` to pull the APK off the device
  - use `./apktool` to decompile
  - insert logging code
  - recompile/sign/align apk
  - push apk to device
  - run and monitor logs with `adb -d logcat`

Inserting logging code into smali

- find `.locals` at top of method and add 1
- create a new constant for the logging tag
  - `const-string v<NUM>, "LOGGING DATA"`
- log inserting
  - `invoke-static {v<NUM>, <register to snoop>}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I`

Mitigation techniques

- Big idea: plan for your API to be reverse engineered.
- Use an already public facing API
- Have systems to mitigate abuse
- Delay feedback

When you have to obfuscate

- Just one piece of defense in depth
- Use certificate pining
- Hide revealing strings
- On Android:
  - Call out to native libraries
  - ProGuard for obfuscation
- Use device specific identifiers

QnA

chris@dropbox.com

@varenc