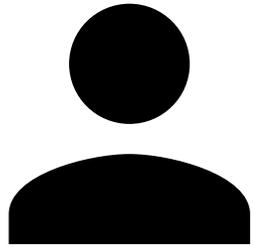


Assignment 1

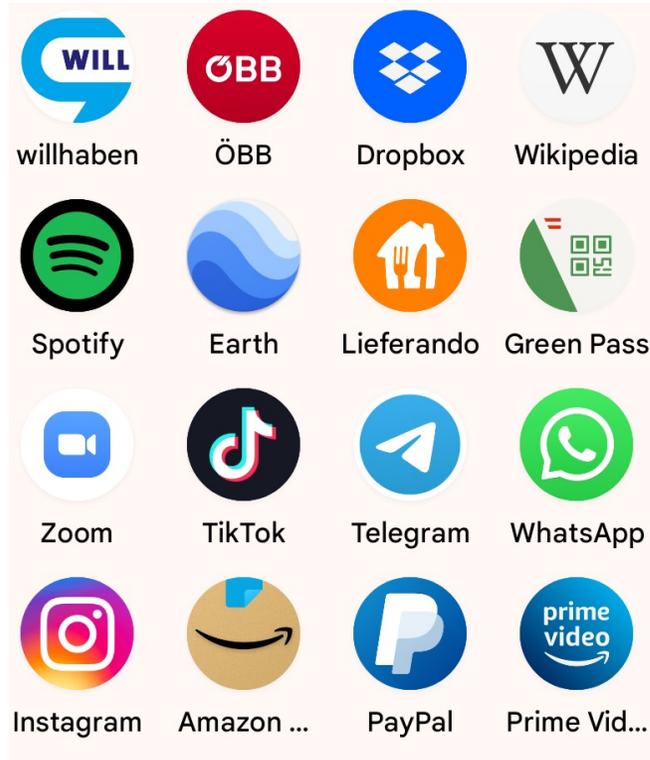
Mobile Security 2023

Florian Draschbacher
florian.draschbacher@iaik.tugraz.at

Some slides based on material by **Johannes Feichtner**

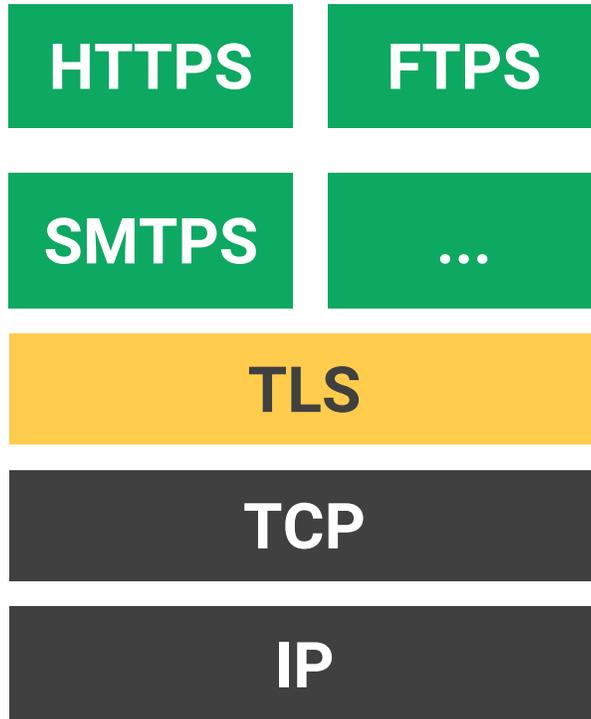


...wants
privacy



- *Am I talking to who I think I do?*
- *Does anyone tamper with my data?*
- *Who else can see my conversation?*

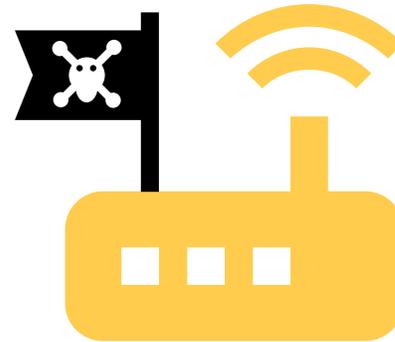
Recap: Transport Layer Security



Problem: „Secure Identity“

Problem: Key Exchange

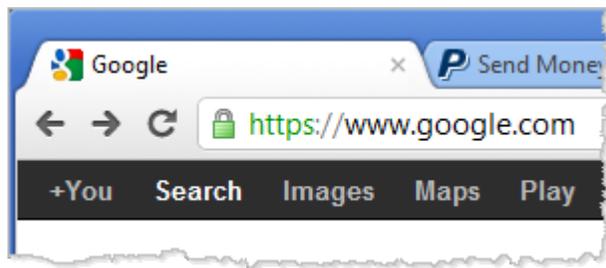
Recap: Man-in-the-middle



Active attacker

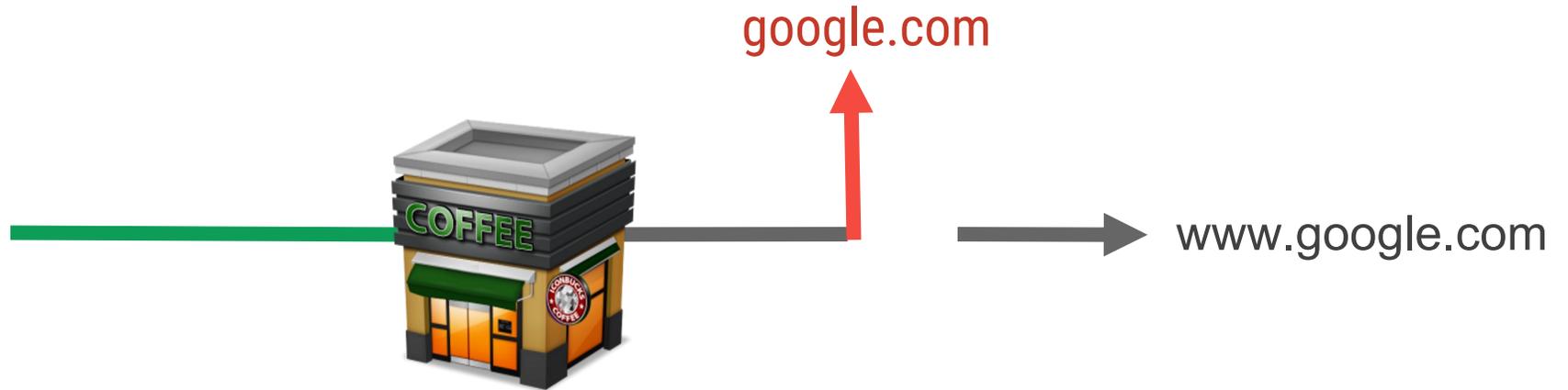
Secretly relay (and possibly modify) traffic between client and server

Picture: [Google](#) / [Apache 2.0](#)



Client

Ideally does not notice anything (from an attacker's perspective)



Picture: [blaugrana-tez](#) / [CC BY-NC-ND](#)

Practical Defenses

- **Validate server certificate chain**
 - From server certificate to device-installed CA
 - Baseline of TLS security
 - Some developers disable validation for supporting self-signed certificates
 - Very bad idea!
- **Implement certificate pinning**
 - Hard-code the expected hash of the server certificate
 - Prevents attacks that
 - Involve state actors, malicious or compromised CAs
 - Involve users who installed additional CA certs to their device

TLS on Android

- **SSLSocket** class for establishing secure TLS or SSL connection
- **Validating certificate chain: TrustManager**
 - Default: Trust any CA installed on device
 - Custom implementations may perform any validation logic (or none at all)
- **Ensuring certificate hostname matches server hostname: HostnameVerifier**
 - Has to be invoked by code above SSLSocket
 - Developer's responsibility!

HTTPS on Android

- **Use Android's `HttpsURLConnection` class**
 - By default: `SecureTrustManager` and `HostnameVerifier` (Details depend on Android version)
 - Possibility to use custom `TrustManager` and `HostnameVerifier`
- **Use a third-party library such as `OkHttp` (built on top of `SSLSocket`)**
 - Usually secure custom `TrustManager` and `HostnameVerifier`
 - Support self-signed certificates, certificate pinning, ...
- **Implement a custom HTTP stack on top of `SSLSocket`**
 - Secure system-default `TrustManager`
 - `HostnameVerifier` up to developer!

Situation Pre-Android 7

Q: “Does someone know how to accept a self-signed certificate on Android?
A code sample would be perfect.”

A: “Use the AcceptAllTrustManager”.

Q: “All I need to do is download some basic text-based and image files from
a web server that has a self-signed SSL certificate...getting the SSL to
work is a nightmare...”

A: “I found two great examples of how to accept self-signed SSL
certificates, one each for `HttpsURLConnection` and `HttpClient`.”

[Source: Stackoverflow]

Applications

- Can overwrite certificate validation routines (system default: correct check)
- Self-signed certificates → used to require custom TrustManager
- Used to have to implement pinning on their own if wanted

Network Security Configuration (Android 7)

- XML-based system for configuring self-signed certificates and pinning
- These use cases no longer require custom validation code
- Default NSC: Don't trust user-installed CA certificates

However

- Even the NSC can be misconfigured
 - Trust user-installed CAs
- Some applications still use custom TrustManagers or HostnameVerifiers
 - Overrides the NSC system altogether

Your Task

Task 1

Analyse a set of min. 3 applications

- Find out if they are susceptible to MITM
- If any sensitive data is transmitted
- Android recommended, iOS possible as well, but more complex

Roadmap

1. Select and install arbitrary apps on your phone
2. Get used to the topic of MITM / Pinning and learn an attack tool
3. Probe the chosen apps and summarize your results

Grading of Task 1: Your result report

Major impact on grade: Task 2 **but** positive finish only if you solve Task 1 **and** 2!

Task 1 – Detailed Steps (for each of the 3+ apps)

1. Try to intercept app's traffic using proxy server
2. If any HTTP connections or insecure HTTPS
 - Document this fact, go to step 6
3. If you use iOS and your device is jailed:
 - Find another app, go to 1
4. Decompile app to find out how pinning is implemented
 - HTTP library, NSC, custom TrustManager?
5. Android: Modify NSC to trust user-installed CAs
 - Recompile, resign, reinstall the app
6. Analyse the intercepted server communication
 - Sensitive data? Hard-coded secrets? Analytics?
7. Document all findings (screenshots + descriptions)

More details on
assignment website

On the dark side...

MITM attack tools

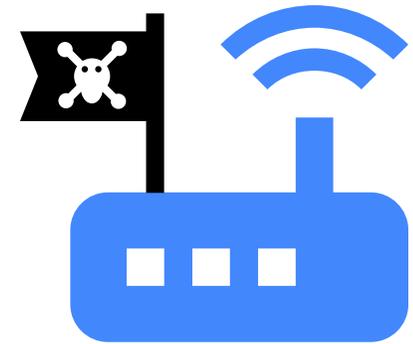
- mitmproxy.org

Decompiling and modifying Android apps

- JADX
- Apktool
- Uber-APK-Signer

Decompiling and analyzing iOS apps

- Ghidra
- Hopper



Picture: [Google](#) / [Apache 2.0](#)

Submission

- **Submit until 17.04.2023:**
 - Report in PDF format
 - List of analysed apps and versions
- **Describe how** you analysed each of the applications
 - Text, screenshots, excerpts from dumps etc.
 - Provide reasoning for your approach
- **Describe** your findings
 - Is any sensitive data leaked?
 - Is HTTP authorization used? Are the credentials hard-coded?
 - Does the app collect analytics?
 - Any other interesting findings?

Submission cont.

Submit **until 17.04.2023:**

- ZIP file with PDF and any supplementary materials (dumps, etc)
- Email to mobilesec@iaik.tugraz.at
- If your ZIP file is too large, upload it to
 - <https://seafile.iaik.tugraz.at/u/d/3019662fd41f41bb8240/>
 - Still send me an email, referencing uploaded file

Reminder: Task 2

- Select a topic for assignment 2 until **31.03.2023**
- Plenty of topics to chose from on website
 - Or suggest your own!
- Groups of up to 3 people
 - But also possible to work on your own
- Send an email to mobilesec@iaik.tugraz.at about group members and topic

Questions?

Short tutorial today after the lecture