Broken Fingers: On the Usage of the Fingerprint API in Android

Antonio Bianchi	University of California, Santa Barbara
Yanick Fratantonio	EURECOM
Aravind Machiry	University of California, Santa Barbara
Christopher Kruegel	University of California, Santa Barbara
Giovanni Vigna	University of California, Santa Barbara
Simon Pak Ho Chung	Georgia Institute of Technology
Wenke Lee	Georgia Institute of Technology

Network and Distributed System Security Symposium (NDSS)

February 19th, 2018

Authentication Schemas in Mobile Apps

Username/Password authentication is problematic, especially on mobile

- Inserting long passwords
- Remembering passwords → Password reuse

We want safer and more usable solutions

- Google Sign-In
- Smart Lock
- …
- **Fingerprint**

Universal 2 Factor (U2F)

Authentication Schemas in Mobile Apps

Can we have the same on mobile devices?

YES

In theory, using the fingerprint API

However, many apps use it incorrectly

Hardware-Protected Authentication

Modern devices have hardware capabilities to implement U2F
→ Their proper usage ***could*** defend even against powerful **"root"** attackers

ARM TrustZone → Trusted Execution Environment (TEE)

- Securely stores and uses cryptographic keys
- The keys are stored inside TrustZone (key non-exportability)
- The keys are locked (cannot be used without a fingerprint touch)

Fingerprint reader sensor

- It communicates directly with TrustZone
- Touching the sensor with registered fingerprints unlocks a key

Contributions

Systematic study
How is the fingerprint API used by Android apps?
How different usages can be exploited?

Automatic detection
Static-analysis tool to automatically detect how apps use the fingerprint API

Propose improvements
Identify weaknesses of the current API, propose improvements

Scope and Threat Model

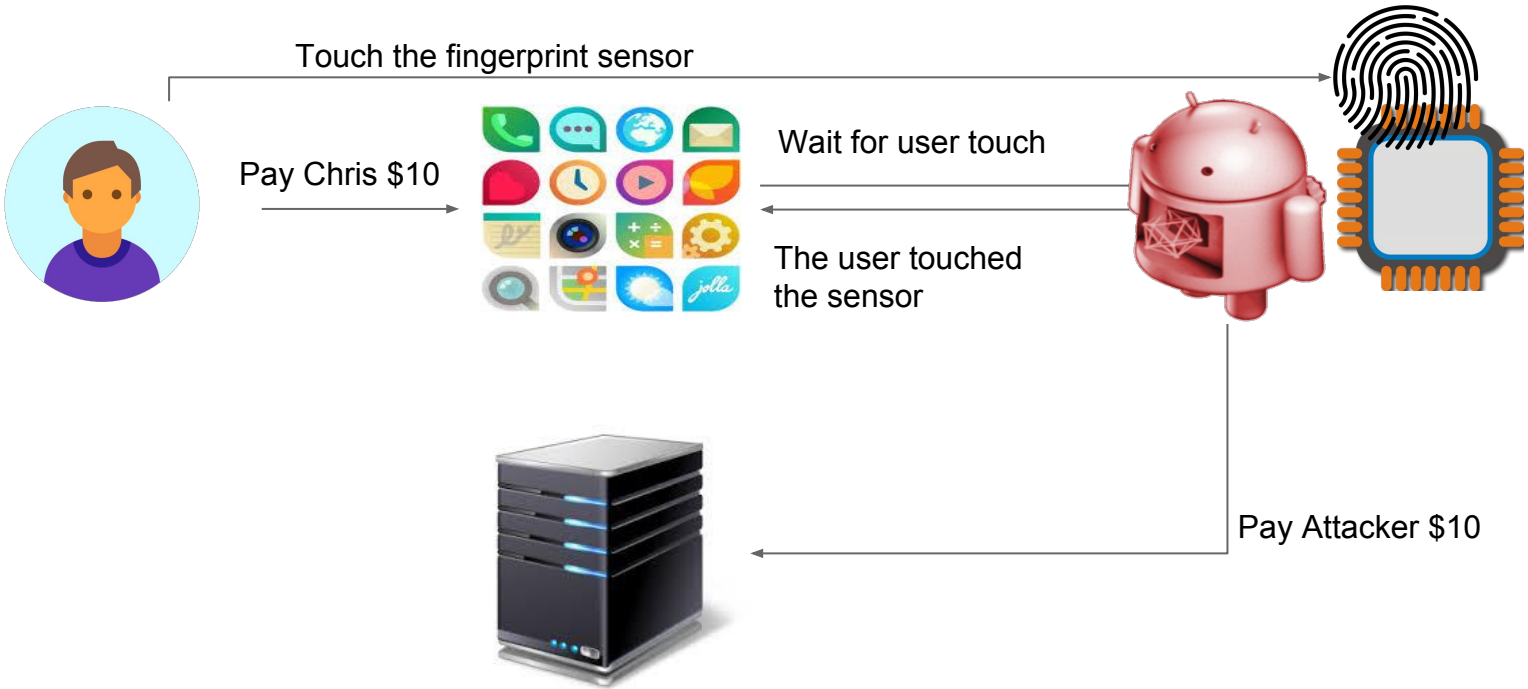
We focus on Google's implementation/devices

- Nexus, Pixel

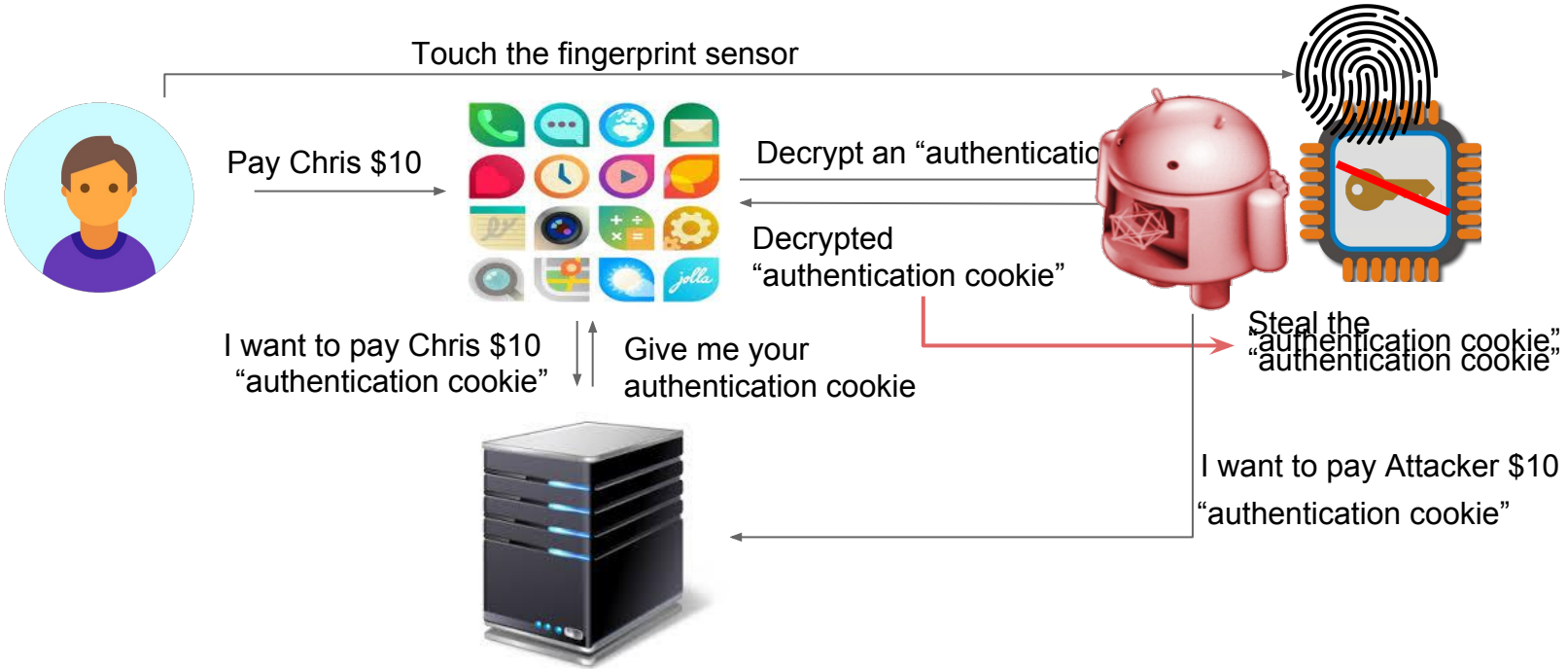
"Physical layer" attacks are out of scope

Assuming TrustZone code is not compromised

Bad Usage: ***Weak***



Not-Ideal Usage: ***Decryption***



Best Usage: ***Sign***

Attack Summary

Assuming an attacker has root

Weak

Complete bypass

Decryption

Complete bypass after the “authentication cookie” is decrypted once

Sign

Safest (confused deputy is still possible)

Static Analysis

Static analysis → Detect how apps use the fingerprint API

- Weak/Decryption/Sign

The analysis is based on

- Call-graph reconstruction
- Data-flow analysis

APK → IR (Soot) → Feature Extraction → Classification

API Details

Functionality	API	Features
Key Generation	`KeyGenParameterSpec$Builder`	*DecryptionKey* *SigningKey*
Key Locking	`setKeyAuthenticationRequired`	*LockedKey* *UnlockedKey*
Key Unlocking	`authenticate(<key>, ...)`	*Null* *NotNull*
Callback	`onAuthenticationSucceded` **Weak**	*NoCrypto* *Constant* *Decryption* *Sign*

Results

501 apps (out of 30,459) can potentially use the fingerprint API (declare the USE_FINGERPRINT permission)

Classified as follow

Errors	***Not Used***	***Weak***	***Decryption***	***Sign***
5 (1.00%)	72 (14.37%)	269 (53.59%)	146 (29.14%)	9 (1.80%)

80% (16/20) ***should*** have used cryptographic checks

Results

Errors	*Not Used*	*Weak*	*Decryption*	*Sign*
5 (1.00%)	72 (14.37%)	269 (53.59%)	146 (29.14%)	9 (1.80%)

Verification

- On a subset of 39 apps
- Dynamically (simulating an attacker)
- Reverse engineering

Accuracy

- 2 misclassifications (~5%)

Case Study – Google Play Store

The Android “Market” app from Google

Configurable to require fingerprint touch to approve purchases

Weak implementation

- No cryptographic checks

Against guidelines from Google itself

- Guidelines suggest to use ***Sign*** for *“authenticating online transactions”*

Case Study – Unlocking Unlocked Keys

A cryptographic key is unlocked by the fingerprint only if the *setUserAuthenticationRequired* API is called

Otherwise, the key is usable without the user touching the sensor

We found 15 apps (4 manually verified) that

- Use the fingerprint API to unlock a cryptographic key
- “Forget” to lock it in the first place!

Current API Weaknesses

The current API has some intrinsic weaknesses (even assuming **Sign** usage)

No Secure UI

The user has no reliable way to know what is *signed* by touching the sensor

TrustZone ***could*** be used to implement Secure UI

Current API Weaknesses

If an attacker has root when the public/private key pair is generated:

- the attacker can send to the remote backend a public key for which the attacker knows the corresponding private key

Key Attestation mitigates this issue

- Verify that the provided key has been generated by TrustZone
- Not commonly used
  - No app using it in our dataset from Feb 2017

Questions?

Antonio Bianchi
antoniob@cs.ucsb.edu