An Introduction to Android Application Security Testing

Nikolay Elenkov

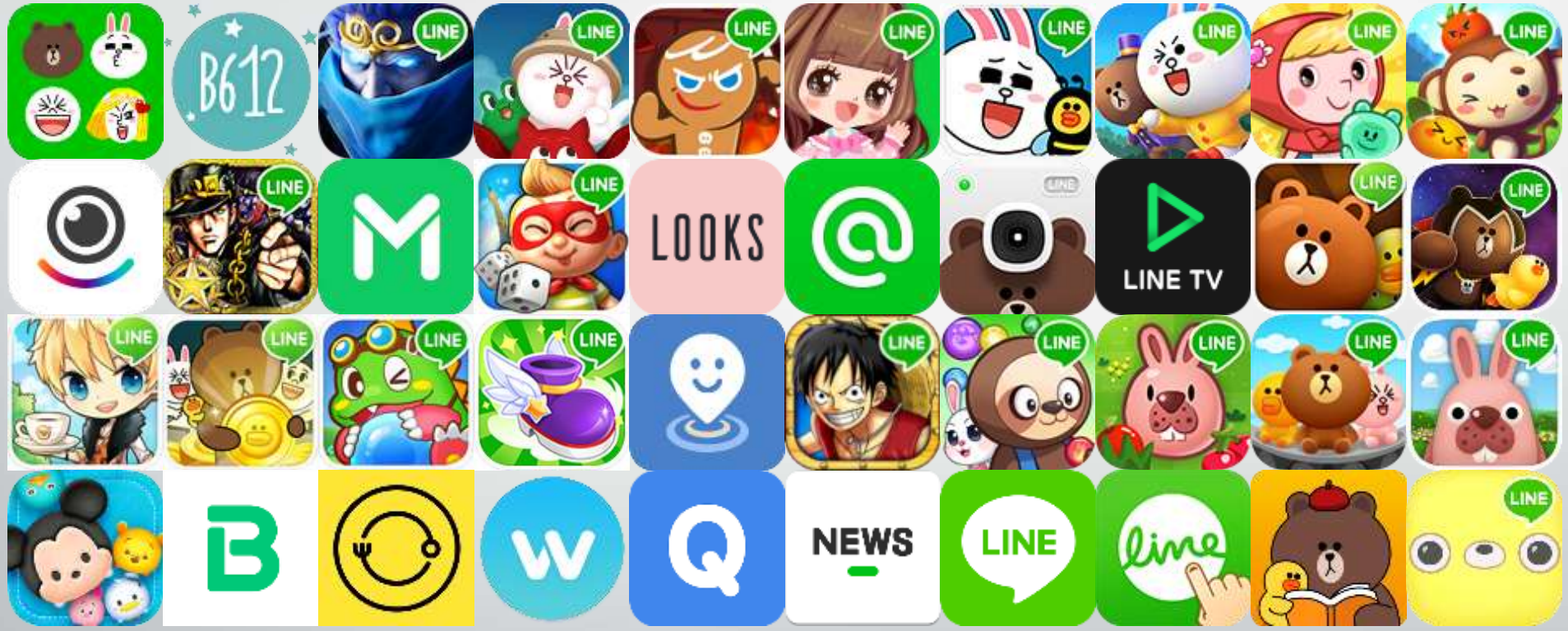
Android Security Symposium, March 2017

Vienna

Agenda

- Static analysis
- Traffic analysis
- Obtaining app data
- Dynamic analysis/hooking
- Common vulnerabilities

LINE apps



Static analysis overview

- Unpack APK file
  - `unzip`
  - `apktool` -> decodes manifest/resources, disassembles Dalvik
- Disassemble/decompile Java (`classes.dex`)
  - `apktool/baksmali` -> smali
  - jadx/JEB -> Java
- Change/repack/resign if needed
- Disassemble/decompile native libraries
  - IDA Pro
  - Hopper

Unpack APK

```
$ apktool d app.apk
I: Using Apktool 2.2.0 on app.apk
…
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

```
./AndroidManifest.xml
./assets/server.p12
./lib/armeabi/libnative.so
./res/anim/abc_fade_in.xml
./res/values/strings.xml
./smali/…/MyActivity.smali
./smali/…/MyApplication.smali
…
```


https://github.com/OWASP/OMTG-Hacking-Playground

Repack APK

```
$ apktool b
I: Using Apktool 2.2.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
$ zipalign -v 4 app.apk app-a.apk
$ jarsigner -keystore test.ks app-a.apk sign
jar signed.
```

```
./AndroidManifest.xml
./assets/server.p12
./lib/armeabi/libnative.so
./res/anim/abc_fade_in.xml
./res/values/strings.xml
./smali/…/MyActivity.smali
./smali/…/MyApplication.smali
…
```


https://github.com/OWASP/OMTG-Hacking-Playground

Examine `AndroidManifest.xml`

```
<manifest package="com.example.app">
  <uses-permission a:name="WRITE_EXTERNAL_STORAGE"/>
  <uses-permission a:name="INTERNET"/>
  <application a:allowBackup="true"
     a:name=".MyApplication" >
   <activity a:name=".MyActivity">
     <intent-filter>
       <action android:name="MAIN"/>
       <category android:name="LAUNCHER"/>
     </intent-filter>
   </activity>
   <service a:exported="false"
            a:name="org.acra.sender.SenderService"
            a:process=":acra"/>
</manifest>
```

- Package name: `com.example.app`
- Permissions
  - can write to external storage
  - can access Internet
- Backup is enabled
- Main activity: `MyActivity`
- Other activities (not shown)
- Has one service
- Uses ACRA for error reporting
- Multi-process app

Smali vs Java

```
const-string v11, "RSA/ECB/PKCS1Padding"

const-string v12, "AndroidOpenSSL"
invoke-static {v11, v12},
Ljavax/crypto/Cipher;-
>getInstance(Ljava/lang/String;Ljava/lang
/String;)Ljavax/crypto/Cipher;

move-result-object v7

const/4 v11, 0x2

invoke-virtual {v7, v11, v8},
Ljavax/crypto/Cipher;-
>init(ILjava/security/Key;)V
```

```
public void decryptString(String alias) {
  try {
    PrivateKeyEntry privateKeyEntry = (PrivateKeyEntry) this.keyStore.getEntry(alias, null);
    RSAPrivateKey privateKey = (RSAPrivateKey) privateKeyEntry.getPrivateKey();
    Log.e(this.TAG, String.valueOf(privateKeyEntry.getPrivateKey().getEncoded()));
    Cipher output = Cipher.getInstance("RSA/ECB/PKCS1Padding", "AndroidOpenSSL");
    output.init(2, privateKey);
```


https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

Native code

- Not really harder to decompile compared to Java
- Not a good way to hide ‘secrets’
- Reversing complex code could be tricky
  - C++
  - templates
- Could be both optimized and obfuscated
  - no symbols
  - syscalls by number
  - Obfuscator-LLVM
  - packing

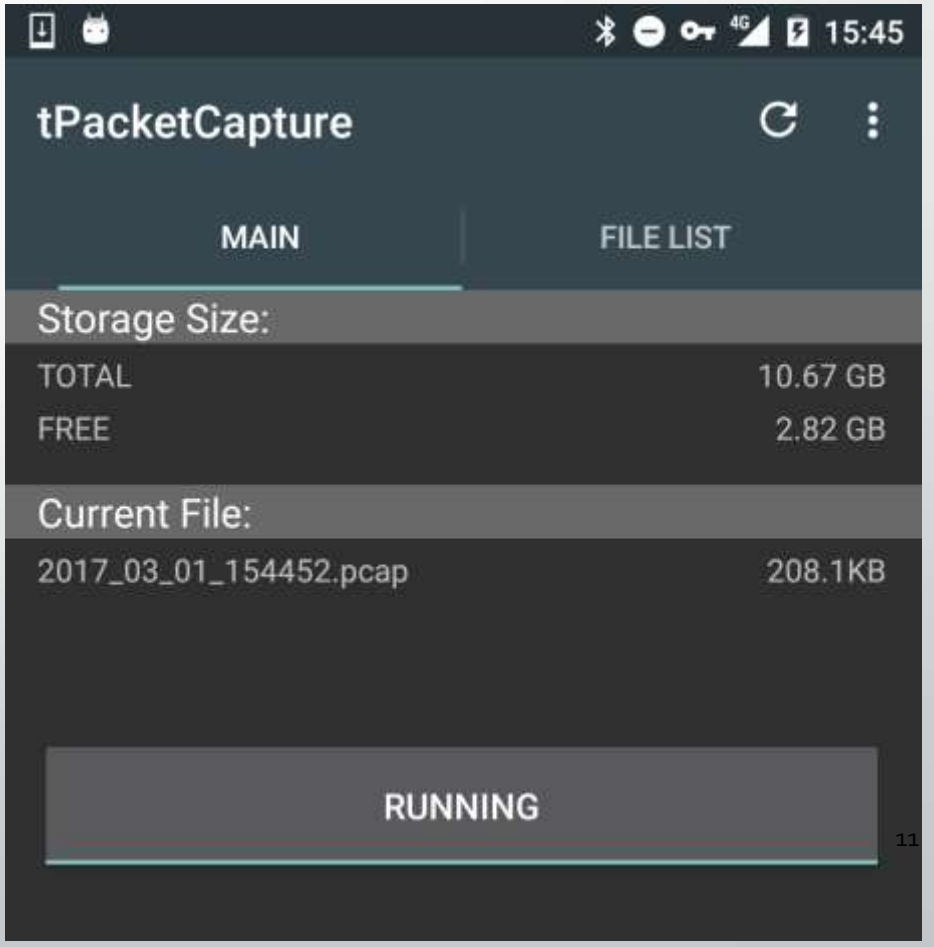
```
var_1C= -0x1C
var_18= -0x18
var_14= -0x14
var_10= -0x10
var_C= -0xC

STMFD   SP!, {R11,LR}
SUB     SP, SP, #0x18
MOV     R2, R1
MOV     R3, R0
STR     R0, [SP,#0x20+var_C]
STR     R1, [SP,#0x20+var_10]
LDR     R0, [SP,#0x20+var_C]
LDR     R1, [R0]
LDR     R1, [R1,#0x29C]
LDR     R12, =(aS3cr3tstring - 0xEE8)
ADD     R12, PC, R12    ; "S3cr3tString!!!"
STR     R1, [SP,#0x20+var_14]
MOV     R1, R12
LDR     R12, [SP,#0x20+var_14]
STR     R2, [SP,#0x20+var_18]
STR     R3, [SP,#0x20+var_1C]
BLX     R12
ADD     SP, SP, #0x18
LDMFD   SP!, {R11,PC}
```

Traffic analysis -- HTTP

- HTTP proxy is usually sufficient
  - Burp Proxy/Charles Proxy/Fiddler/mitmproxy
  - not for HTTP/2 or SPDY…
- Need to perform MiTM to decrypt SSL traffic
  - install CA certificate in user store
  - ignored in Android 7.0, if `targetSdkVersion>=23`
- Certificate pinning could get in the way
  - JustTrustMe and friends (requires root and Xposed)
- Some apps ignore OS proxy settings
  - use Proxy Droid to setup local transparent proxy (needs root)
  - reverse tethering
    - WiFi AP on laptop, route target traffic through transparent proxy

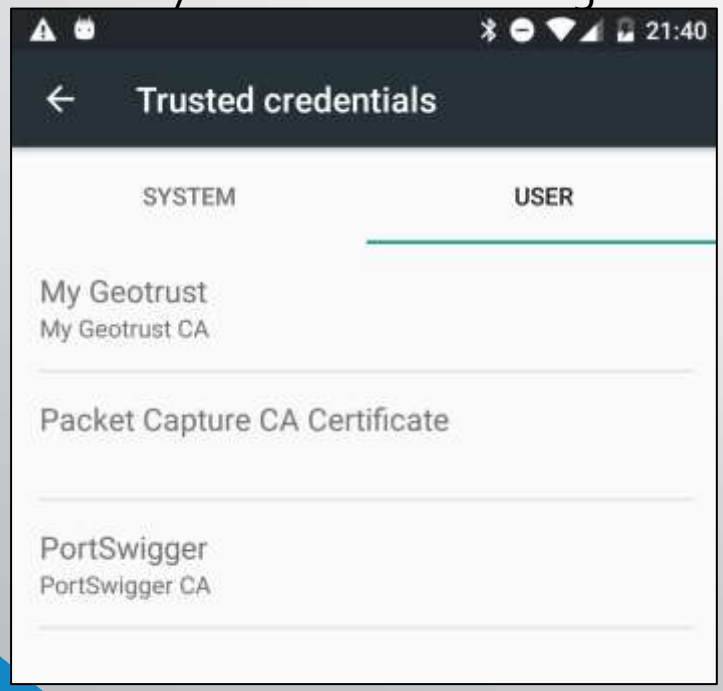
Traffic analysis -- other

- Wireshark for low-level or non-HTTP protocols
  - can decrypt SSL with RSA private key (no-FS cipher suites only)
  - Multiple plugins
- `socat` + Burp/mitmproxy for HTTP-like protocols (SIP)
- `tcpdump` on Android device to capture 3G/LTE traffic
- Android VPN that dumps traffic in pcap format
  - no root required
  - works for both WiFi/3G

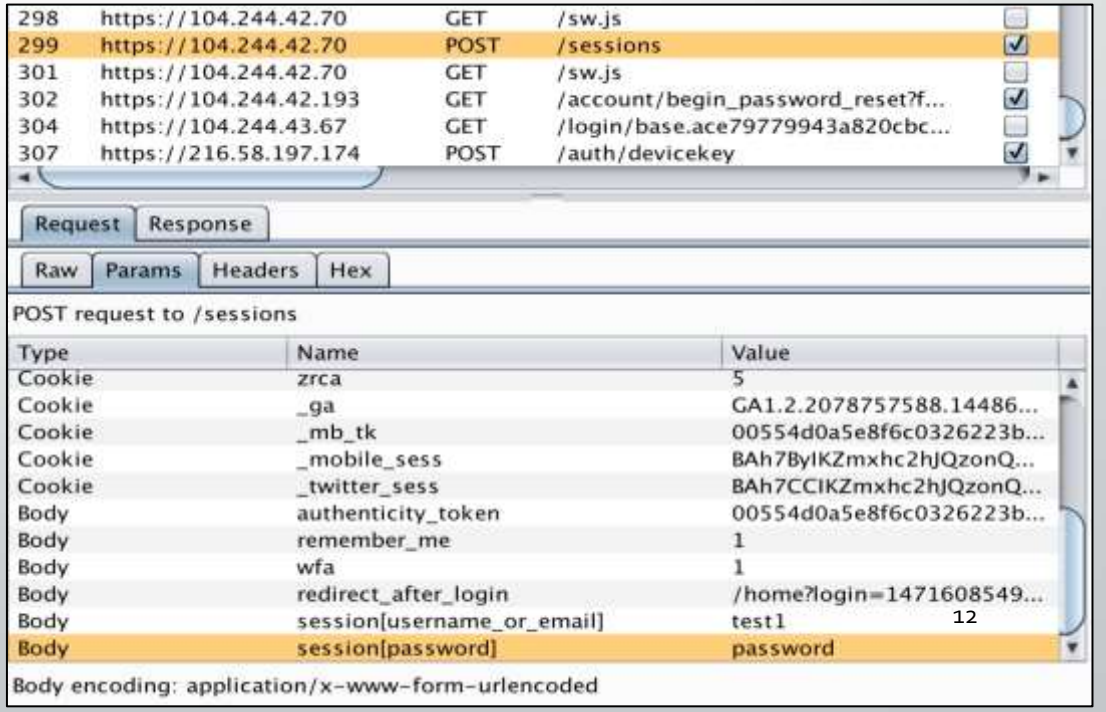


SSL MiTM setup

- Download CA certificate:
  - `http://localhost:8080`
- Security > Install from storage



- Redirect traffic to proxy
  - WiFi settings or Proxy Droid or rev. tethering



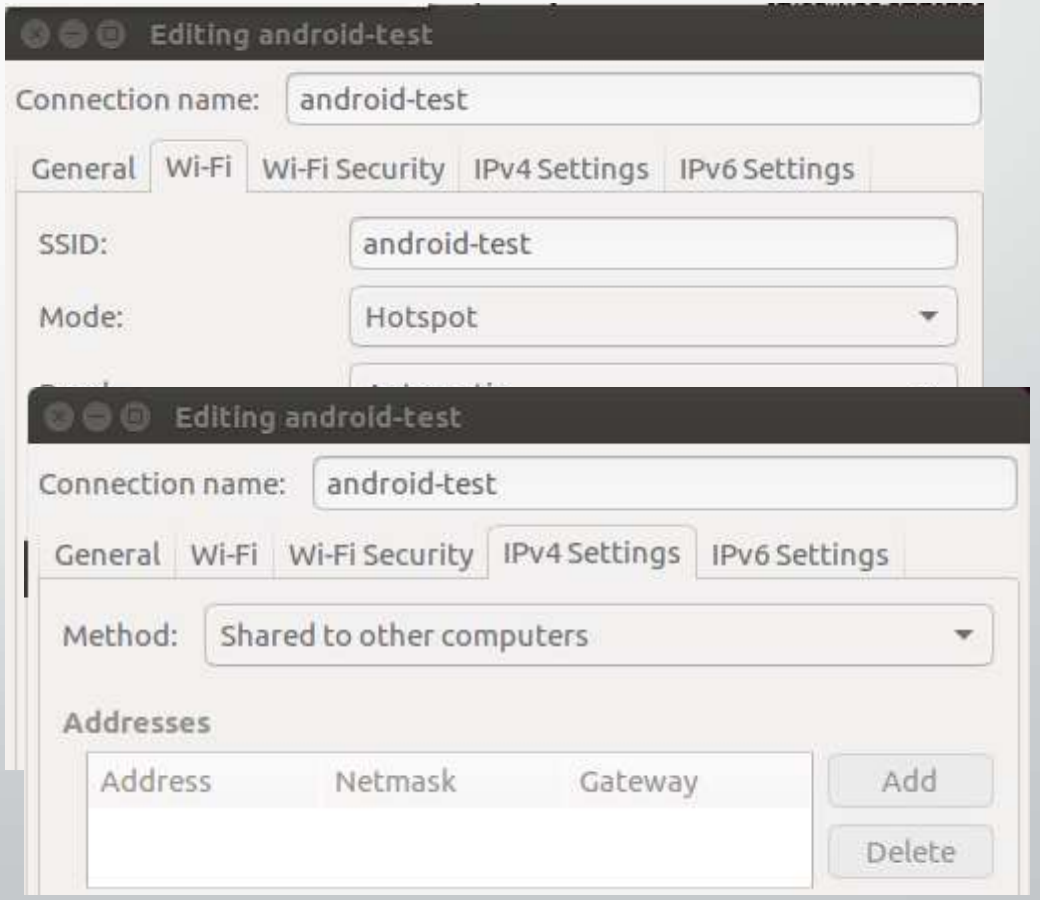
Reverse tethering (Mac)

System Preferences → Sharing → Internet Sharing: On

```
$ sudo sysctl -w net.inet.ip.forwarding=1
$ cat /etc/pf.anchors/forwarding
rdr pass on bridge100 inet proto tcp from any to any port 80 -> 127.0.0.1 port 8080
rdr pass on bridge100 inet proto tcp from any to any port 443 -> 127.0.0.1 port 8080
$ cat /etc/pf.conf
rdr-anchor "forwarding"
load anchor "forwarding" from "/etc/pf.anchors/proxy.rules”

$ sudo pfctl -evf pf.conf
```

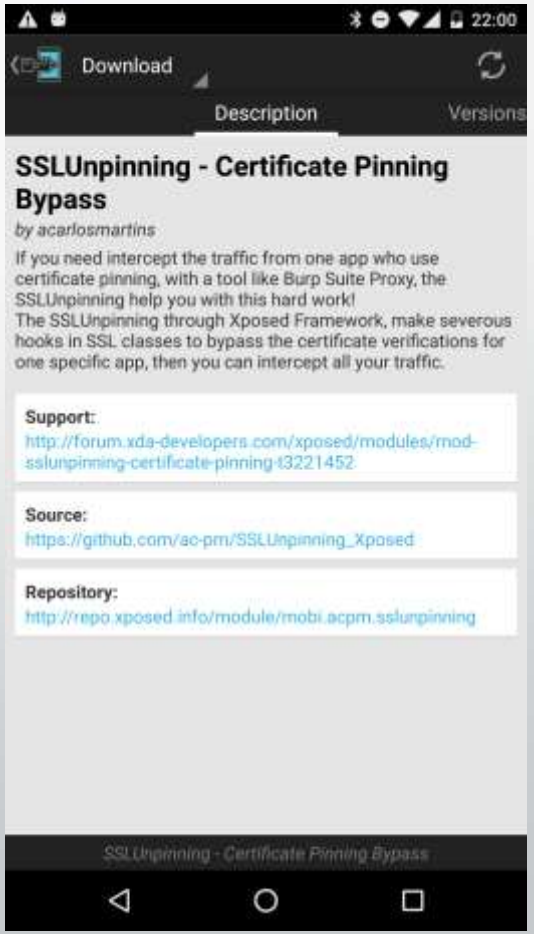
Reverse tethering (Linux)

```
$ iptables -t nat -A PREROUTING -i wlan0 ¥
  -p tcp --dport 80  -j REDIRECT --to-port 8080
$ iptables -t nat -A PREROUTING -i wlan0 ¥
   -p tcp --dport 443 -j REDIRECT --to-port
8080
```



Defeating certificate pinning

- Pinning fixes ('pins') trusted certificate(s) for a particular site
  - supported by platform in Android 7.0+
- Two main methods
  - custom trust store (`.bks` file in assets)
  - check public key hash(es) in code
- To disable pinning
  - remove pinning code and repack
  - hook OS certificate validation (requires root)
    - JustTrustMe
    - SSLUnpinning



Declarative Network Security Config in 7.x

```
<network-security-config>
  <domain-config>
   <domain>example.com</domain>
   <pin-set expiration="2018-01-01">
     <pin digest="SHA-256">...</pin>
   </pin-set>
  </domain-config>
</network-security-config>
```

- Custom trusted CAs
- CAs for debugging
- Disallow cleartext traffic
- Certificate pinning
  - Only trust certain issuers for a given domain
  - Can set expiration time

Network Security Config Implementation

- `Provider` + `TrustManager` + config from XML resource
- `frameworks/base/core/java/android/security/net/config`
  - `NetworkSecurityConfigProvider`
  - `NetworkSecurityConfig`
  - `NetworkSecurityTrustManager`
  - `RootTrustManager`
- Provider installed early in `ActivityThread`

```
NetworkSecurityConfig.getDefaultBuilder():

if (targetSdkVersion <= Build.VERSION_CODES.M) {
   // User certificate store, …
  builder.addCertificatesEntryRef(
    new CertificatesEntryRef(
       UserCertificateSource.getInstance(),
        false));
}
```

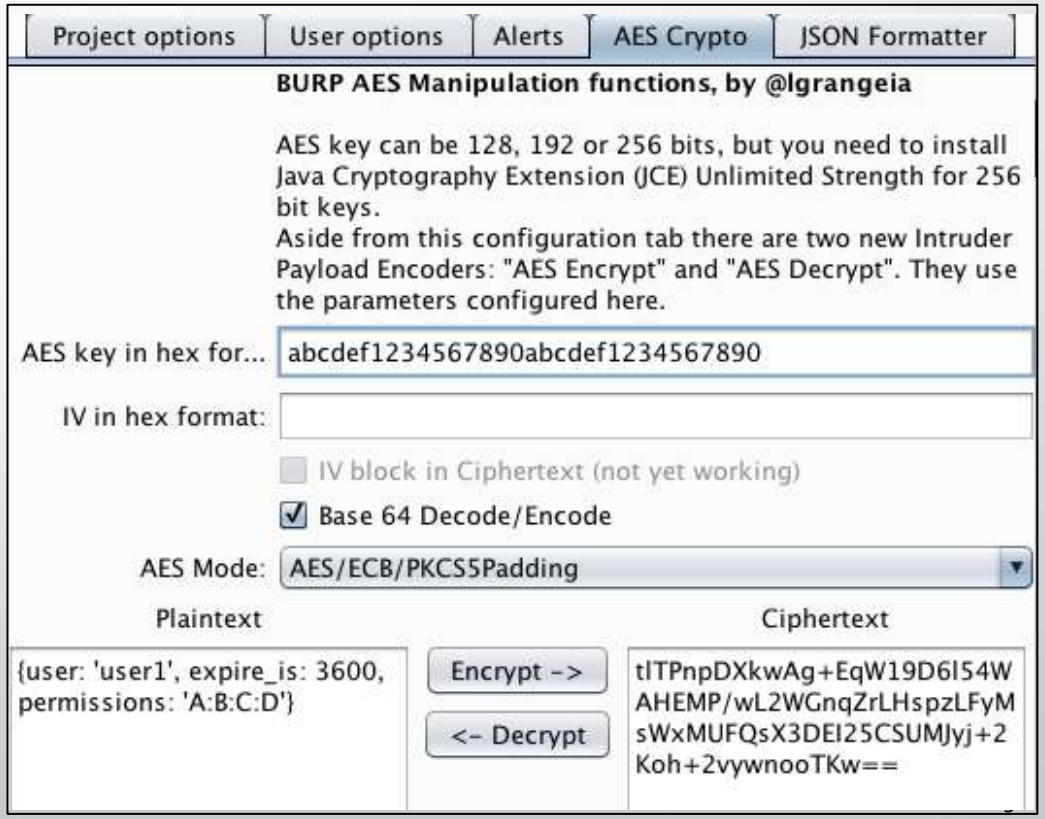
Disabling Network Security Config

- Repack APK and target API < 23
  - will use certs in User trust store
  - (same as pre-Nougat)
- Hook `RootTrustManager`, etc.
  - inject own CA
  - disable trust checks
  - requires root + Frida, etc.

```
Java.perform(function() {
  var rootTm =Java.use("android.security.net.config.RootTrustManager");

rootTm.checkServerTrusted.overload("[Ljava.security.cert.X509Certificat
e;", "java.lang.String", "java.net.Socket").implementation =
function(certs, authType, socket) {
        send("cert subject: [" + certs[0].getSubjectDN().getName() +
"]");
        send("authType: [" + authType + "]");
        var soAddr = socket.getRemoteSocketAddress().toString();
        send("address: [" + soAddr + "]");
    };
});
```

Protocol Analysis

- Wireshark for exploration
  - custom HTTP ports (e.g., TLS on port 80)
- Burp for analysis/manipulation
  - Capture/replay
  - Match and replace
  - Many plugins available
  - Java-based (can use Python too)
  - Can extract serialization code from app and use in plugin (Thrift, Protobuf, etc.)
- mitmproxy if you like Python



Obtaining data via backup

- Backup is on by default (unless disabled in manifest), no root required

```
$ adb backup org.nick.kanjirecognizer
Now unlock your device and confirm the backup operation...
$ java -jar abe-all.jar unpack backup.ab backup.tar password
21893120 bytes written to backup.tar.
OR (for unencypted devices)
$ dd if=backup.ab bs=24 skip=1|openssl zlib -d > backup.tar
$ tar xvf backup.tar
apps/org.nick.kanjirecognizer/_manifest
apps/org.nick.kanjirecognizer/r/app_webview
…
apps/org.nick.kanjirecognizer/r/app_webview/databases
apps/org.nick.kanjirecognizer/r/app_webview/databases/Databases.db-journal
apps/org.nick.kanjirecognizer/r/app_webview/databases/Databases.db
```

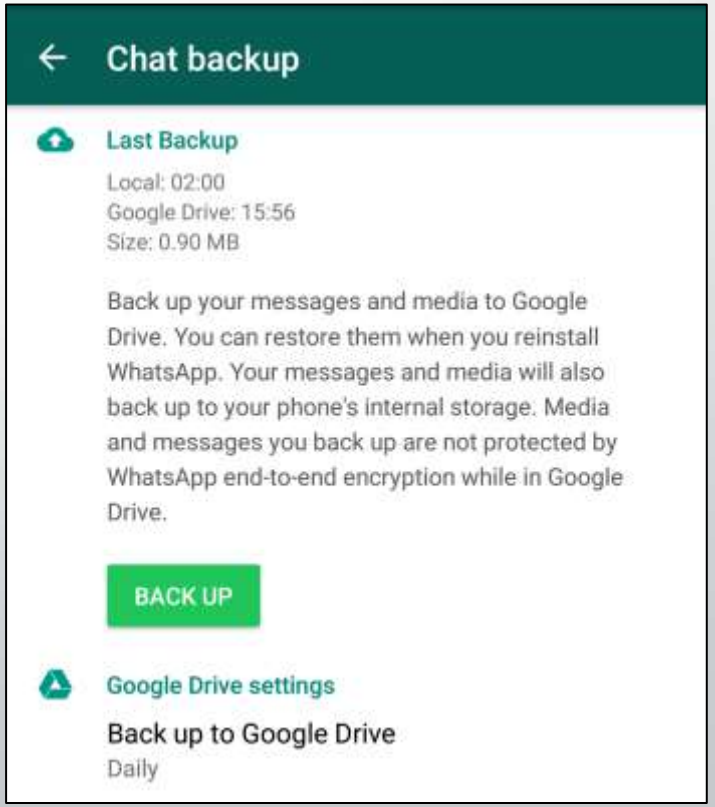
Copying data directly

- Requires rooted device for sandboxed files
- Sandboxed app data is in `/data/data/<package name>/`
- Shared app data is in `/sdcard/Android/data/<package name>/files`

```
$ adb shell
$ su
# cd /data/data
# tar cvf /sdcard/kr-data.tar org.nick.kanjirecognizer/
# tar cvf /sdcard/kr-sdata.tar /sdcard/Android/data/org.nick.kanjirecognizer/files/

(on desktop)
$ adb pull /sdcard/kr-data.tar
$ adb pull /sdcard/kr-sdata.tar
```

Cloud storage

- Common cloud data stores
  - Google Drive app folder
    - integrated in Play Services, has libs
    - hidden, `drive.appdata` scope
  - Firebase Storage
  - Dropbox, etc.
  - Custom server-side storage



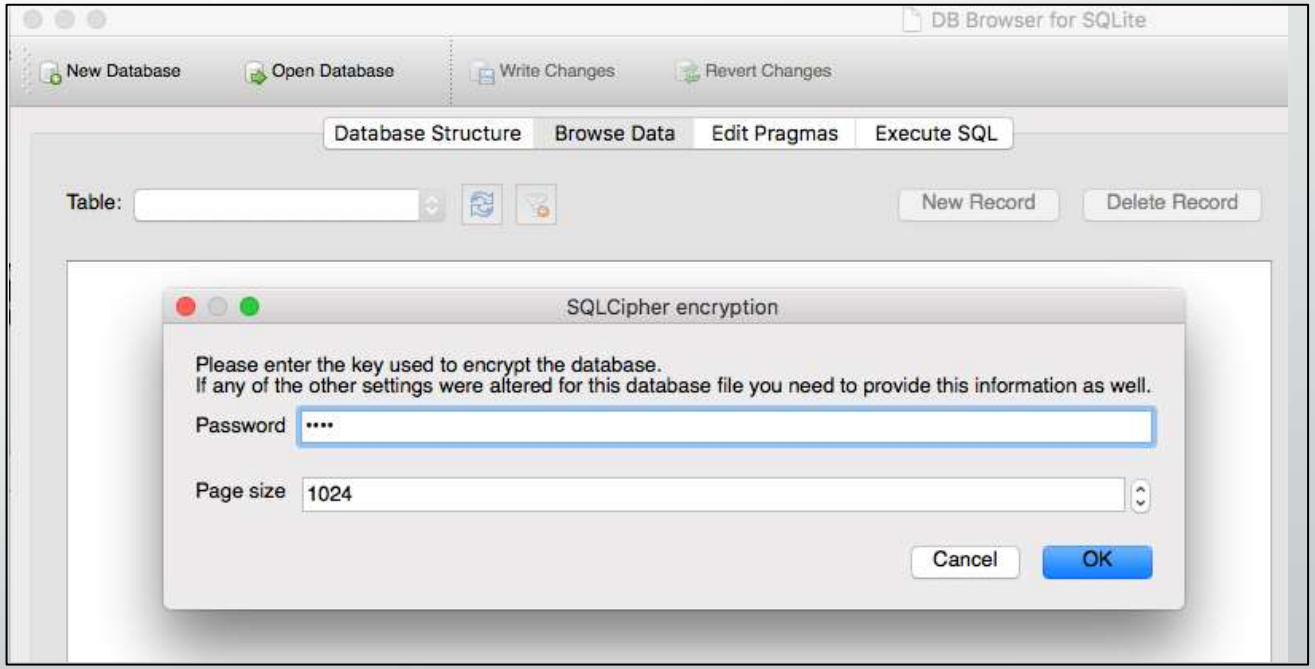
Obtaining cloud data

- Capture or extract auth tokens
  - from `AccountManager` DB
  - pretend to be Play Services…
- Call HTTP APIs with token(s)
- Watch our for API limits

```
def get_master_token(email, password):
    url = 'https://android.clients.google.com/auth'
    d = {}
    d['Email'] = email
    d['Passwd'] = password
    d['app'] = 'com.google.android.gms'
    d['client_sig'] = GMS_SIG
    d['parentAndroidId'] = deviceID
    hdrs = {…}
    r = requests.post(url, headers=hdrs, data=d)
    token = r.text.split('¥n')[2].split('=')[1]
```

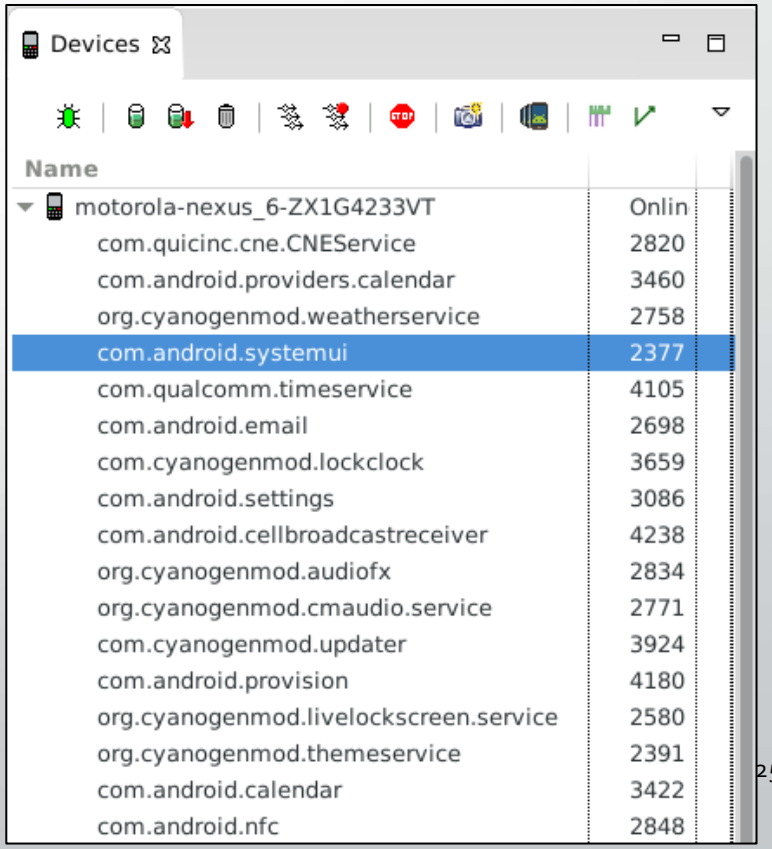
Types of app data

- Databases – `databases/`
  - SQLite, SQLCipher (encrypted)
  - may be accessible via content providers
- Shared preferences – `shared_prefs/`
  - XML
- Files – `files/`
  - app-specific format
  - noSQL -- realm DB, etc.
- WebView files – `app_webview/`, `cache/`
  - cache
  - cookies



Dynamic analysis -- debugging

- Can only attach to apps with `debuggable` attr
  - repack as debuggable if not
- Use custom engineering build (ROM)
  - can attach to any process (`ro.debuggable=1, ro.secure=0`)
- Execution tracing
  - JDWP/DDMS
  - `strace/ftrace`
- Stepping through decompiled code
  - smalidea
  - JEB
- Native debugging
  - `gdb` from NDK
  - IDA Pro



Dynamic analysis -- hooking

- Change app behavior w/o repackaging
  - Modify method parameters
  - Capture return values (crypto keys, etc.)
- Requires root and/or replacing OS components
- Can mod(ify) OS or app behavior
  - Disable system protections/sandbox
  - Disable app integrity checks
- Java – Xposed, Frida
- Native – Frida, ADBI

Hooking -- Xposed

- Replaces Dalvik/ART to allow for hooks
- Hooks are in Java, inside a helper APK
- Requires restart to apply changes
- Often used in OS mods
- Doesn't currently work on Android 7.x

```
findAndHookMethod("android.webkit.WebViewClient",
    lpparam.classLoader,
    "onReceivedError", WebView.class,
    int.class, String.class, String.class,
    new XC_MethodReplacement() {
    @Override
    protected Object replaceHookedMethod(
       MethodHookParam param) throws Throwable {
         return null;
    }
});
```


https://github.com/Fuzion24/JustTrustMe

Hooking -- Frida

- Works on iOS and Android
- Works on 64-bit and Android 7.x
- Python API/tools + JS script
- Can update hooks without restart
- Needs agent process on device
- Harder to hook before startup
  - spawn gating + event handling

```
Java.perform(function() {
   var secretKey = Java.use("javax.crypto.spec.SecretKeySpec");
   var crypto = Java.use("org.nick.androidpbe.Crypto");
   crypto.encrypt.overload("java.lang.String",
     "javax.crypto.SecretKey", "[B").implementation =
       function(plaintext, key, salt) {
         send("plaintext: [" + plaintext + "]");
         var sc = Java.cast(key, secretKey);
         send("key: [" + sc.getEncoded()+ "]");
         var ret = this.encrypt.overload("java.lang.String",
           "javax.crypto.SecretKey",
            "[B").call(this, plaintext, key, salt);
         return ret;
   };
});
```

Testing game security

- Mobile games are not really Android apps…
  - Unity3D -- .NET (Mono)
    - or IL2CPP → compile to native code
  - Cocos2d-x – C++
- Tools
  - ILSpy
  - .NET Reflector
  - Hopper
  - IDA Pro

- Threat model – cheating
  - attacker == user
  - local – memory cheats
  - network – replay, packet modification
  - game bots
  - item purchase bypass
- Mitigations
  - binary packing/obfuscation
  - monitoring, abuse/bot detection
  - protocol obfuscation/encryption

Common vulnerabilities

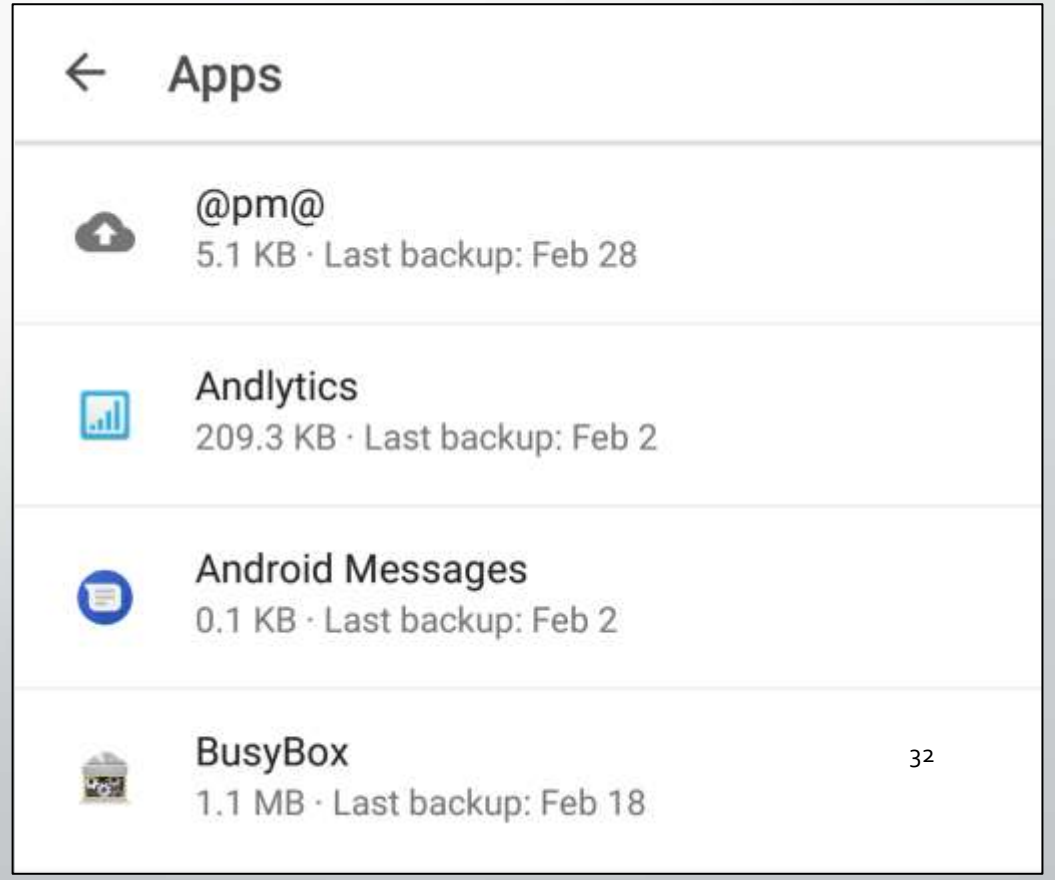
- Unsecured Android components
- Insecure storage
- Information leakage
- Insecure communication
- Weak crypto
- Broken authentication/session management
- Insecure WebViews
- Lack of OS bug mitigations
- Lack of code protection
  - (obfuscation/root detection/tamper detection, etc.)

Unsecured Android components

- Exported components
  - services/content providers/activities
  - has `<intent-filter>` == exported
  - content providers exported by default for API level < 17
- Unsecured custom permissions
  - permission level != *signatur*e -- insecure
  - *dangerous* permissions require user interaction if targeting Android 6.0+
- *How to check*
  - examine `AndroidManifest.xml`
  - Watch out for `<permission protectionLevel="normal|dangerous">`
  - try to access suspicious components from other app/ADB shell
    - `am start –a /am broadcast -a/am startservice/service call <service> <code> [params]`

Insecure storage

- Sensitive data stored without encryption
  - credentials, cryptographic keys, keystores, auth tokens
  - in DBs/shared prefs/files/account manager
- External storage (SD card)
  - Accessible to all applications with `READ_EXTERNAL_STORAGE` permission
- Internal storage
  - normally protected by OS sandbox
  - may leak via backup/data export (local or cloud), compromised OS or `WORLD_READABLE` files
- *How to check*
  - check for world readable files on internal storage
  - obtain all app data
  - search for/match potentially sensitive data



Information leakage

- Sensitive data and/or app state into logcat
  - can be obtained by other apps on older Android
- Sensitive data in side-channels (third-party services)
  - ad networks
  - analytics
- Broadcasting sensitive data
- *How to check*
  - monitor/capture logcat and grep for sensitive sata/IDs
    - `adb logcat -d -v time > logcat.txt`
  - examine all traffic (including to third-party hosts)
  - examine third party libraries (ad/tracking/analytics SDKs)
  - watch out for unsecured/system-wide broadcasts/IPC

Insecure communication – plain text

- Credentials/tokens over plaintext connection (HTTP)
- Extension using untrusted data downloaded via HTTP
  - ZIP files – path traversal, ZIP bombs
  - Executable code (plugins, DEX files)
  - JavaScript
- Home-made transport encryption
- *How to check*
  - examine app all traffic (including from third-party hosts)
  - check origin of all downloaded data
  - check level of local validation (if any)/manipulate server responses

Insecure communication – broken SSL

- **No verification of server certificate**
- No hostname verification
- Broken/old ciphersuites (DES, RC4, MD5, SHA-1 based)
- Vulnerable SSL implementation (old OpenSSL)
- *How to check*
  - perform MiTM (without installing CA certificate on device)
  - grep for custom `X509TrustManager|HostnameVerifier`
  - check linked OpenSSL version (if used)
  - check for custom OpenSSL `verify_callback`
  - use nogotofail

Weak cryptography – usual suspects

- Home-made 'encryption' (Base64, XOR-based, etc.)
- Old/broken algorithms (DES, MD5, SHA-1)
- Weak RNGs, fixed seeds (`java.util.Random`, `RandomStringUtils`, `setSeed()`)
- Hard-coded keys
- Insecure block modes (ECB, fixed IVs, CBC without HMAC)
- Hash instead of HMAC (length extension)
- Weak/home-made KDFs (SHA1PRNG-based, `srand(time(NULL))`)
- Home-made crypto protocols (authentication, key exchange, etc.)
- Vulnerable crypto libraries

Weak cryptography – how to check

- Identify (most) crypto-related code
  - `grep -riI 'javax.crypto' src/`
  - `grep -riI 'X509' src/`
  - `grep -riI 'SSL_' native-src/`
- Decode anything that looks like Base64 strings
  - sometimes plaintext
  - lots of hard-coded secrets
- Look for patterns in tokens/session IDs/encrypted files
- Examine crypto protocols for flaws

Broken authentication/session management

- Hard-coded API keys/secrets
- Short/repeated/easily guessable session IDs
- Insufficient API authorization
- Direct object reference
- *How to check*
  - monitor and manipulate/replay traffic
    - remove cookies/tokens and resend
  - collect session IDs/tokens and look for patterns/repetition
  - build alternative API clients, send unexpected data, call APIs out of order

Insecure WebViews

- Too much native functionality exposed via `@JavascriptInterface`
  - lack of input validation
- Ignores SSL errors: calls `SslErrorHandler.proceed()`
- Allows access to local files
- Unchecked (custom) URL schemes: intents with scheme/host specified
- XSS in server application
- *How to check*
  - monitor and manipulate traffic, inject script tags
  - inject/debug JavaScript (remote debugging in Chrome)
  - `$ grep onReceivedSslError() |setAllowFileAccess()| setCookie()|shouldOverrideUrlLoading()`

Lacking OS bug mitigations

- Relevant if supporting older Android versions
  - check `minSdkVersion` and `targetSdkVersion`
- Broken `SecureRandom` (Android 4.1 – 4.3)
  - https://android-developers.blogspot.jp/2013/08/some-securerandom-thoughts.html
- `HttpsURLConnection` uses SSLv3 on Android < 6.0
  - OkHttp retries with SSLv3 even when using TLS-only context (like most browsers)
  - Need to forcibly remove SSLv3 in socket factory
- Vulnerable OpenSSL (e.g., CVE-2014-0224)
  - Updatable OpenSSL-based GMS provider
  - https://developer.android.com/training/articles/security-gms-provider.html

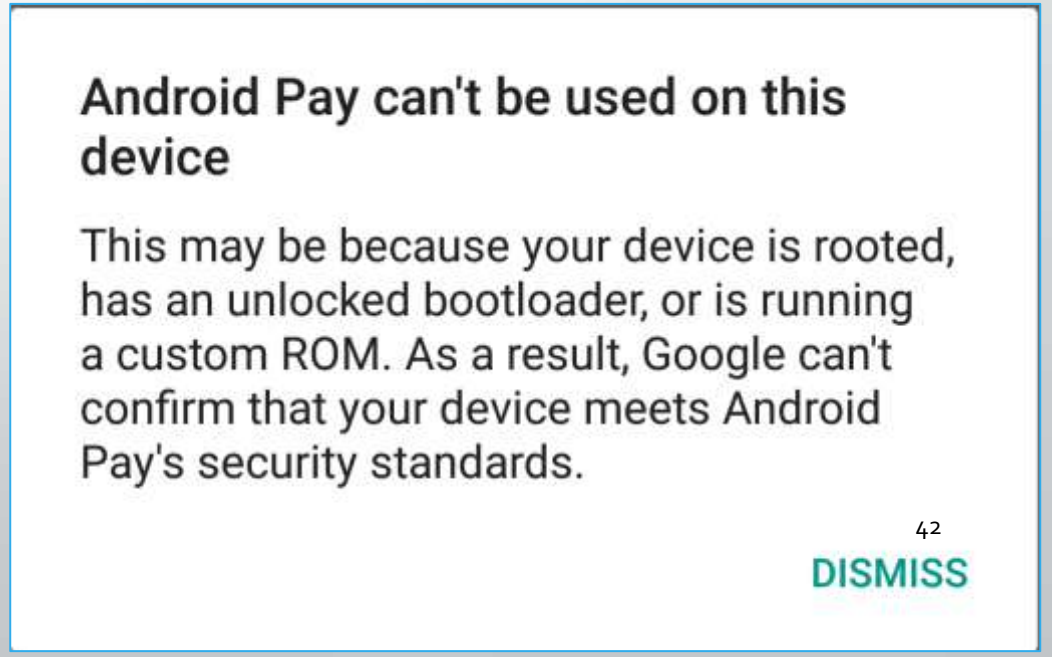
Lack of code protection

- No obfuscation
  - dex files
  - native libs
- No tamper detection
  - hash of `classes.dex`, hash of signing certificate
- *How to check*
  - decompile, change smali code, repackage and run
  - run on developer, rooted, custom ROM device

No OS integrity check

- Rooting, etc. effectively break the OS security model
- Common mitigation -- 'root' detection
  - check for SuperSU package, `su` binary, other SUID binaries
  - not 100% reliable, usually insufficient
- Other indicators that OS security may be weakened
  - signed with `test-keys`
  - code injection in `dalvik-cache/`
  - rogue device admin apps
  - unknown CA certificates in User trust store
  - SELinux, dm-verity disabled
  - core system properties modified
  - OS-wide proxy or VPN installed

- Google SafetyNet -- checks if running on certified device
  - checks for multiple indicators



Tools and more

- https://ibotpeaches.github.io/Apktool
- https://github.com/skylot/jadx
- https://github.com/JesusFreke/smali/wiki/smalidea
- https://github.com/google/enjarify
- https://portswigger.net/burp/proxy.html
- https://mitmproxy.org/
- https://github.com/rovo89/Xposed
- https://github.com/Fuzion24/JustTrustMe
- https://github.com/nelenkov/android-backup-extractor
- https://github.com/AndroBugs
- https://github.com/frida/frida
- https://github.com/crmulliner/adbi
- https://github.com/CalebFenton/simplify
- https://github.com/rednaga/APKiD
- https://github.com/google/nogotofail
- https://github.com/scottyab/rootbeer
- https://developer.android.com/training/safetynet/index.html
- https://github.com/poliva/random-scripts/tree/master/android
- https://github.com/MobSF
- https://github.com/ucsb-seclab/agrigento

Resources

- OWASP Mobile
  - https://github.com/OWASP/owasp-masvs
  - https://github.com/OWASP/owasp-mstg/
- https://www.nowsecure.com/resources/secure-mobile-development/
- https://github.com/doridori/Android-Security-Reference
- http://www.droidsec.org/wiki/
- https://www.jssec.org/dl/android_securecoding_en.pdf
- https://www.enisa.europa.eu/publications/smartphone-secure-development-guidelines-2016

Questions?

- Twitter: @kapitanpetko
- Blog: https://nelenkov.blogspot.com