Securing Android Applications

Dario Incalza

OWASP

The Open Web Application Security Project

$ whoami

- Pre-sales & Security Engineer @ GuardSquare
- Pentesting mobile applications
- Securing mobile applications
- keybase.io/h4oxer
- @h4oxer
- www.darioincalza.be

Outline

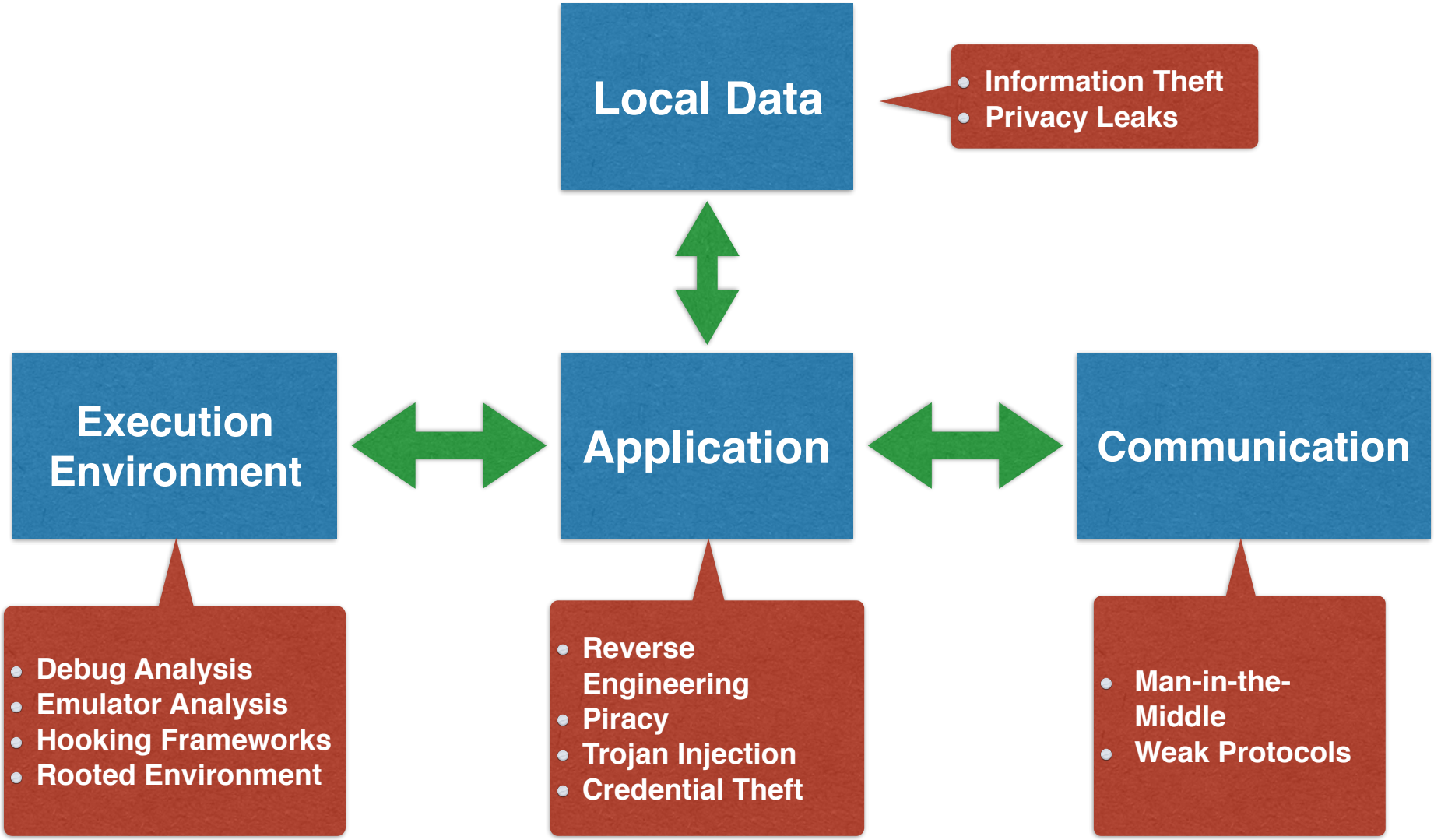
- Android Application 101
- Attack Surfaces Android Application
- Securing Android Applications
  - Cryptography
  - Code Protection
  - Secure Communications
  - Secure Execution Environment

Android Application 101

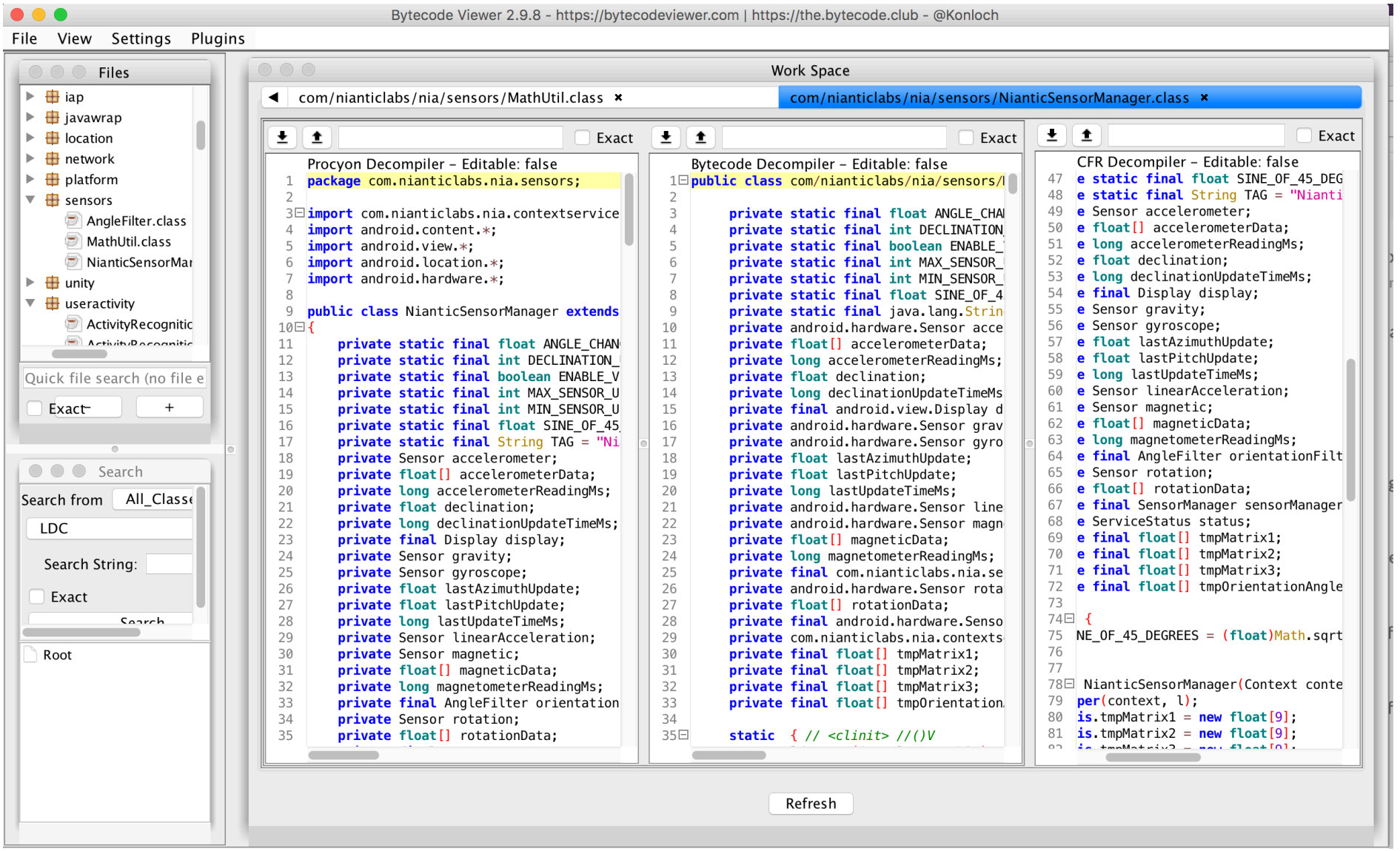
- Java or C/C++
- .apk file == zip file
- Easy to disassemble
- Recompiled upon installation

Attack Surfaces

Attack Surfaces



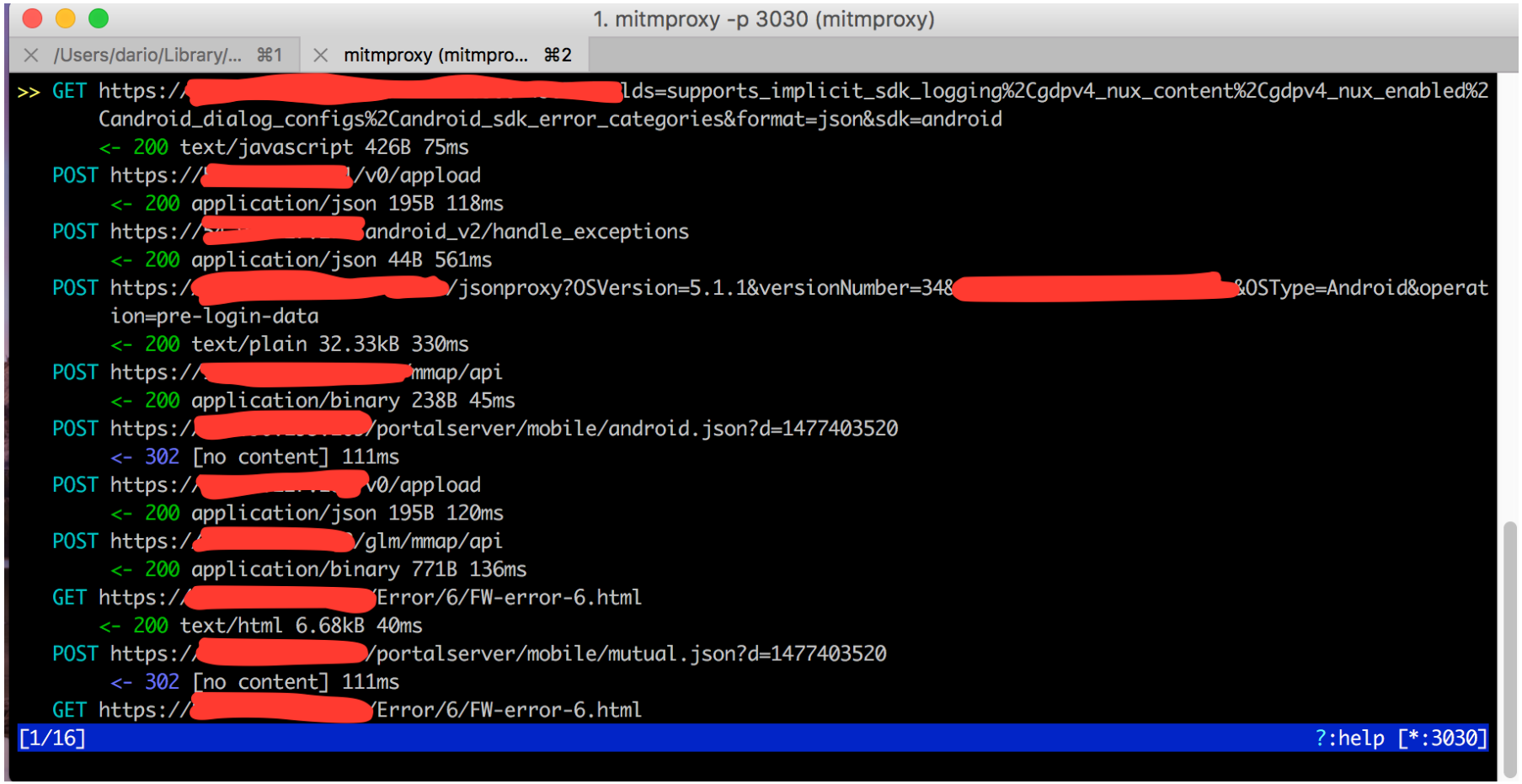
Bytecodeviewer



APKTool

```
dario@cryptox:~/REWorkspace/pokego » apktool d pokemongo.apk -o apktool_unzipped
I: Using Apktool 2.2.1 on pokemongo.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/dario/Library/apktool/framework/1.apk
I: Regular manifest package...
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...
dario@cryptox:~/REWorkspace/pokego »
```

mitmproxy


1. mitmproxy -p 3030 (mitmproxy)
/Users/dario/Library/... ⌘1
mitmproxy (mitmpro... ⌘2
>> GET https://[redacted]lds=supports_implicit_sdk_logging%2Cgdpv4_nux_content%2Cgdpv4_nux_enabled%2Candroid_dialog_configs%2Candroid_sdk_error_categories&format=json&sdk=android
<- 200 text/javascript 426B 75ms
POST https://[redacted]/v0/appload
<- 200 application/json 195B 118ms
POST https://[redacted]android_v2/handle_exceptions
<- 200 application/json 44B 561ms
POST https://[redacted]/jsonproxy?OSVersion=5.1.1&versionNumber=348[redacted]&OSType=Android&operation=pre-login-data
<- 200 text/plain 32.33kB 330ms
POST https://[redacted]mmap/api
<- 200 application/binary 238B 45ms
POST https://[redacted]/portalserver/mobile/android.json?d=1477403520
<- 302 [no content] 111ms
POST https://[redacted]v0/appload
<- 200 application/json 195B 120ms
POST https://[redacted]/glm/mmap/api
<- 200 application/binary 771B 136ms
GET https://[redacted]Error/6/FW-error-6.html
<- 200 text/html 6.68kB 40ms
POST https://[redacted]/portalserver/mobile/mutual.json?d=1477403520
<- 302 [no content] 111ms
GET https://[redacted]Error/6/FW-error-6.html
[1/16]
?:help [*:3030]

xPosed Framework

- Enables Java and native hooking
- Manipulates zygote process on Android
- Injects XposedBridge.jar in every app
- Implement hooking modules
- No need to modify APKs

xPosed Hooking Module

```
    findAndHookMethod("com.example.BankApp", "signTransaction",
new XC_MethodHook()
{
    protected void beforeHookedMethod(MethodHookParam param)
    {
          //execute code before method call
    }


    protected void afterHookedMethod(MethodHookParam param)
    {
         //execute code after method call
    }
}
```

Securing Android Applications

Securing Android Applications

- Use secure best coding practices
- Protect, obfuscate and encrypt your application code
- Harden your communication
- Take into account the execution environment


GREAT QUESTION
GLAD YOU ASKED

Cryptography

Problems

- How to store sensitive information on the device?
- How to securely generate crypto keys?
- How to manage crypto keys?
- What if the user enables FDE?

Crypto 101

- **Symmetric Crypto** = one key for encryption/decryption
  - AES, 3DES, Blowfish, many more
- **Public-key Crypto** = private and public key
  - Encrypt with private key, decrypt with public key = digital signatures
  - Encrypt with public key, decrypt with private key = confidentiality
  - RSA, ElGamal, ECC, many more

Securely Generate a PBK

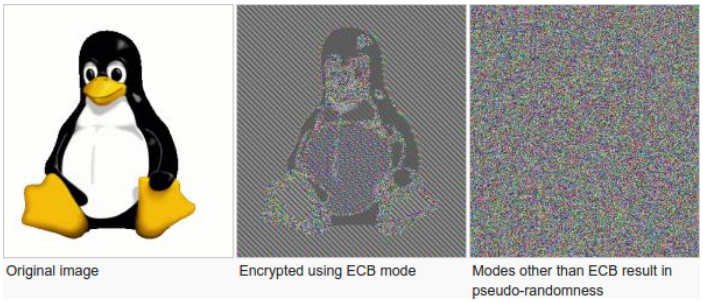
```
public byte[] getEncryptionKey(char[] strongPassword){

    int iterationCount = 10000;
    int keyLength = 256;
    int saltLength = keyLength / 8;
    SecureRandom random = new SecureRandom();
    byte[] salt = new byte[saltLength];
    random.nextBytes(salt);

    KeySpec keySpec = new PBEKeySpec(strongPassword, salt,
            iterationCount, keyLength);

    SecretKeyFactory keyFactory = SecretKeyFactory
            .getInstance("PBKDF2WithHmacSHA1");

    return keyFactory.generateSecret(keySpec).getEncoded();

}
```

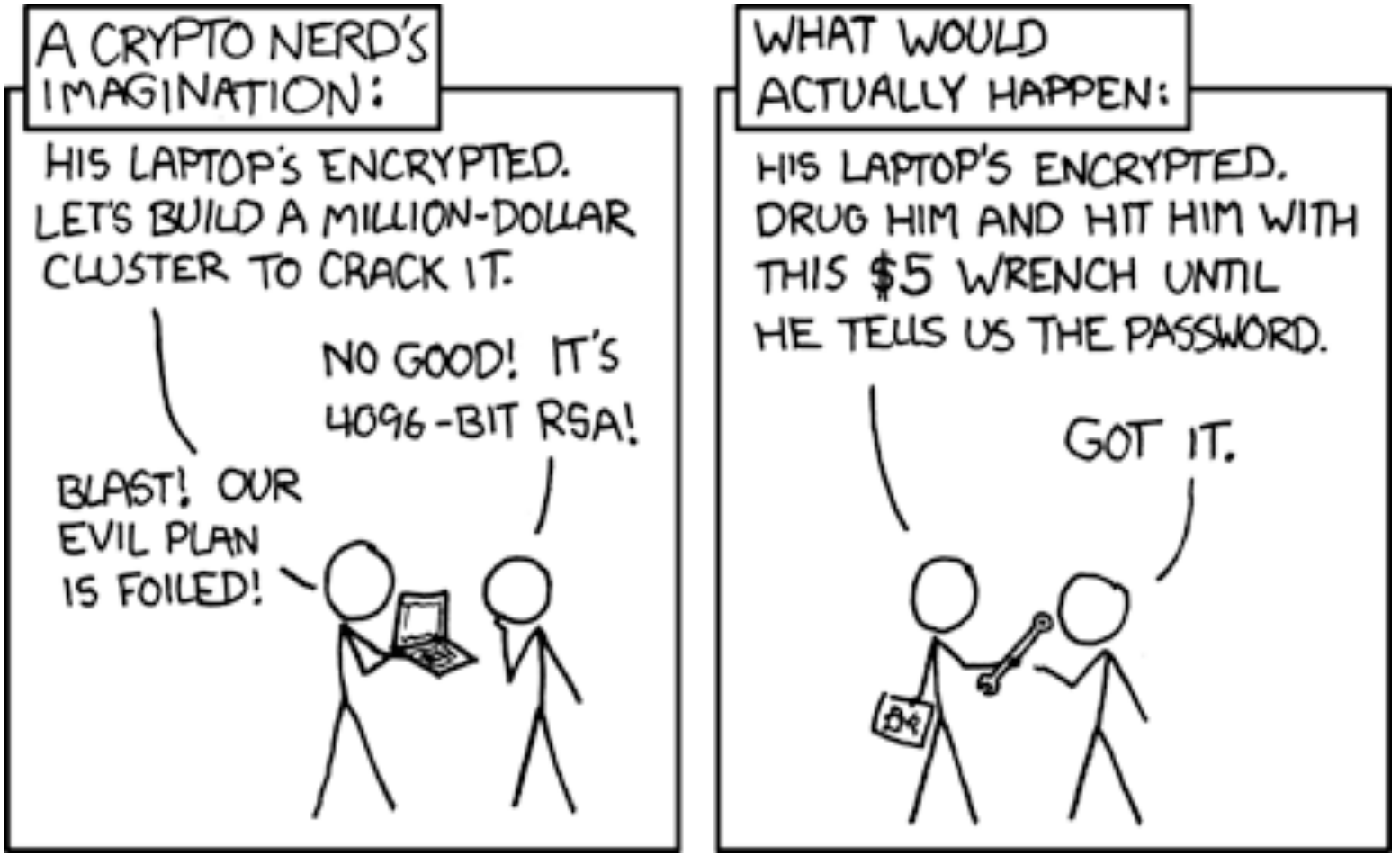
Securely Manage Keys

1. Ask user for password, do not store keys, use PBKDF2
2. Generate Keys and store in KeyStore
   - Vulnerable on rooted devices (hard)
3. Generate Keys and store in SharedPreferences
   - Vulnerable on rooted devices (easy)
4. Use hardcoded key in application
   - One key, reverse engineering, key leaked, big problem
5. Store Generated Key in /sdcard/
   - Readable by all apps, stop.

DONT'S

- Hardcoded Crypto Keys
- Save Crypto Keys in /sdcard/
- Log sensitive information
- Use AES in ECB mode
- Use DES, MD5, it's broken/weak
- Implement DIY crypto
- String objects for sensitive information
- Not fixing the SecureRandom vulnerability < JB




A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Code Protection

Problems

- How to make reverse engineering harder?
- How to protect your code against extraction?
- How to protect API keys?
- How to hide cryptographic operations?

Code Protection

- Name obfuscation
- String encryption
- Class encryption
- Resources, asset and native library encryption
- Control flow and arithmetic obfuscation
- Hide calls through reflection

For Example …

```
 public String encryptSensitiveMessage()
 {
   String nuclearLaunchCode = "abc123";
   String encryptionKey     = “secretkey";

   return CryptoEngine.encrypt(nuclearLaunchCode,
encryptionKey);
 }
```

Layer 1 - API Call Hiding

```
  public String encryptSensitiveMessage()
  {

      String nuclearLaunchCode = "abc123";
      String encryptionKey     = "secretkey";
      Class  clazz             = Class.forName("CryptoEngine");

      Method meth = clazz.getMethod(“encrypt”, String.class,
String.class);

      return (String) meth.invoke(null, nuclearLaunchCode, encryptionKey);
  }
```

Layer 2 - String Obfuscation

```
 public String encryptSensitiveMessage()
 {

   String nuclearLaunchCode = Base64.decode("YWJjMTIz");
   String encryptionKey     = Base64.decode("c2VjcmV0a2V5");
   Class  clazz             =
Class.forName(Base64.decode("Q3J5cHRvRW5naW5l"));
   Method meth              =
clazz.getMethod(Base64.decode("ZW5jcnlwdA=="),
                                 String.class,String.class);

   return (String) meth.invoke(null,nuclearLaunchCode,encryptionKey);
 }
```

Layer 3 - Name Obfuscation

```
    public String a()
    {

        String a      = e.f("YWJjMTIz");
        String b      = e.f(“c2VjcmV0a2V5");

        Class  c      =
Class.forName(e.f(“Q3J5cHRvRW5naW5l"));

        Method d      = c.getMethod(e.f(“ZW5jcnlwdA=="),
String.class, String.class);

        return (String) d.invoke(null, a, b);
    }
```

ProGuard

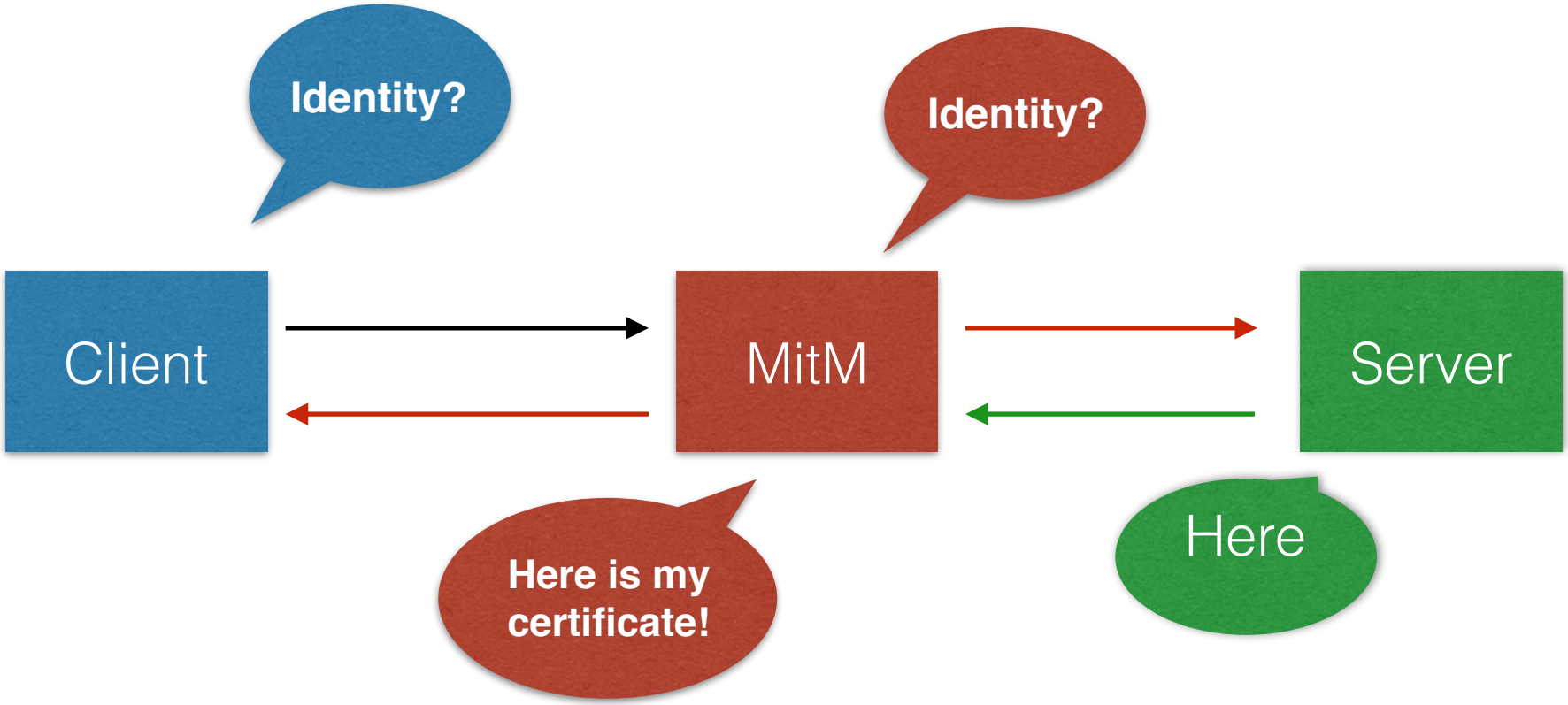
- Open source
- Optimization & shrinking
- Name obfuscation
- Default in the Android SDK

Securing Communications

SSL 101

- A certificate = cryptographically signed identification information
- Certificates are issued by Certificate Authorities (CAs)
- Your Android device trusts a number of CAs
- SSL validation = check if certificate of server is issued by trusted CA

Problem


Identity?
Identity?
Client
MitM
Server
Here is my certificate!
Here

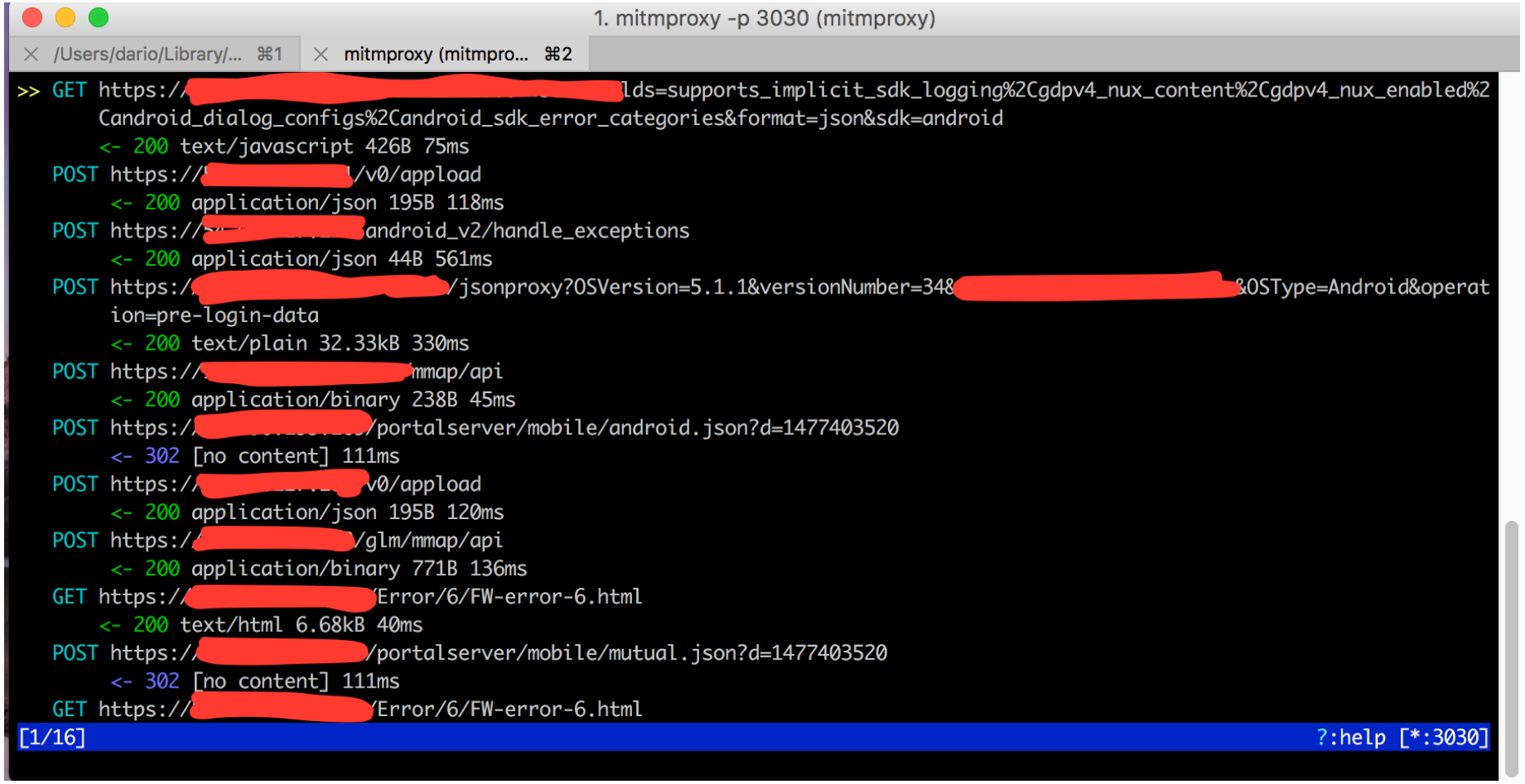
Problem

```
$ emulator -avd Nexus_5X_API_22 -http-proxy
http://localhost:3030
```

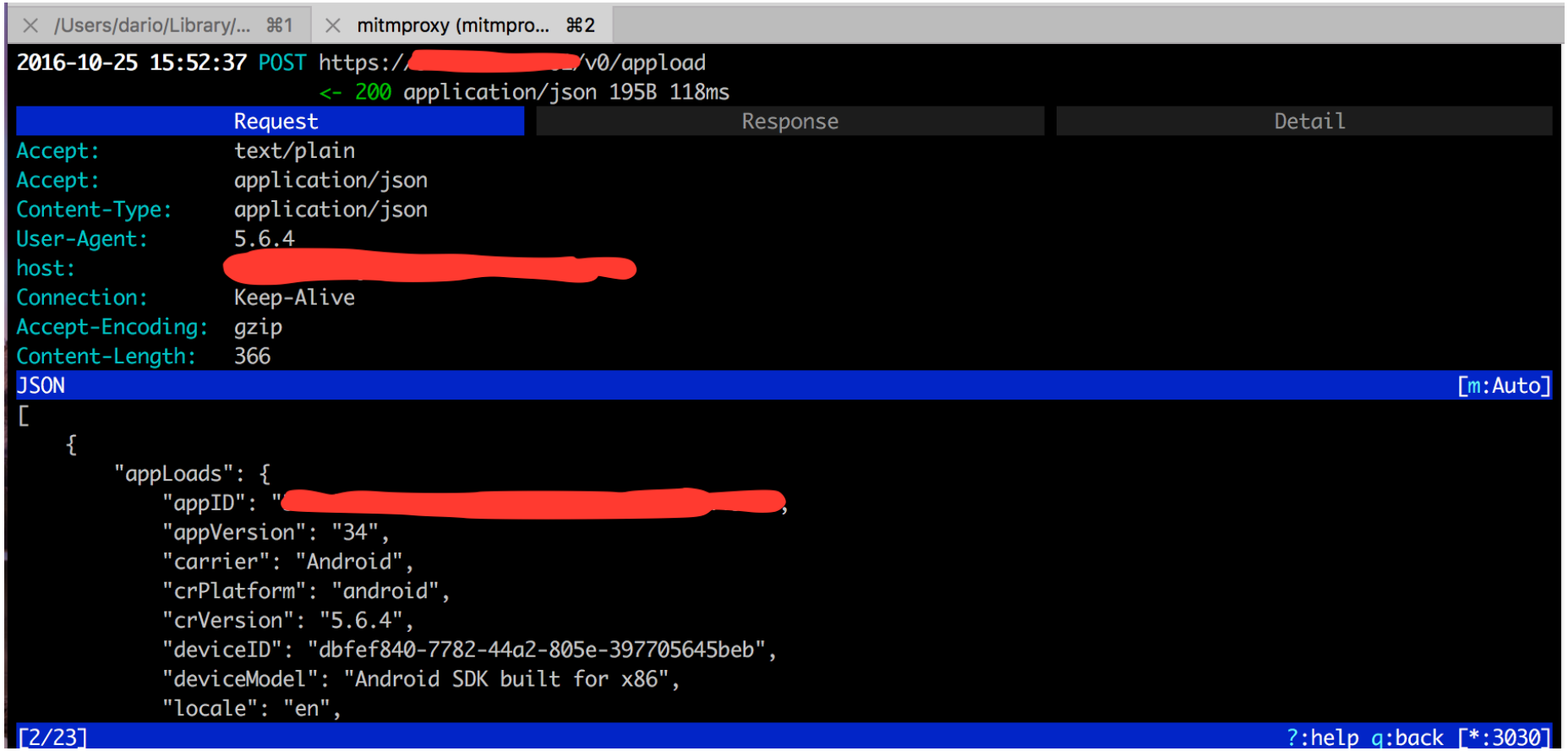
```
$ mitmproxy -p 3030
```

- Used for API debugging
- Used for API analysis
- Used for MiTM attacks

Problem


1. mitmproxy -p 3030 (mitmproxy)
/Users/dario/Library/... ⌘1
mitmproxy (mitmpro... ⌘2
>> GET https://[redacted]lds=supports_implicit_sdk_logging%2Cgdpv4_nux_content%2Cgdpv4_nux_enabled%2Candroid_dialog_configs%2Candroid_sdk_error_categories&format=json&sdk=android
<- 200 text/javascript 426B 75ms
POST https://[redacted]/v0/appload
<- 200 application/json 195B 118ms
POST https://[redacted]android_v2/handle_exceptions
<- 200 application/json 44B 561ms
POST https://[redacted]/jsonproxy?OSVersion=5.1.1&versionNumber=348[redacted]&OSType=Android&operation=pre-login-data
<- 200 text/plain 32.33kB 330ms
POST https://[redacted]mmap/api
<- 200 application/binary 238B 45ms
POST https://[redacted]/portalserver/mobile/android.json?d=1477403520
<- 302 [no content] 111ms
POST https://[redacted]v0/appload
<- 200 application/json 195B 120ms
POST https://[redacted]/glm/mmap/api
<- 200 application/binary 771B 136ms
GET https://[redacted]Error/6/FW-error-6.html
<- 200 text/html 6.68kB 40ms
POST https://[redacted]/portalserver/mobile/mutual.json?d=1477403520
<- 302 [no content] 111ms
GET https://[redacted]Error/6/FW-error-6.html
[1/16]
?:help [*:3030]

Problem


/Users/dario/Library/... ⌘1
mitmproxy (mitmpro... ⌘2
2016-10-25 15:52:37 POST https://[redacted]/v0/appload
<- 200 application/json 195B 118ms
Request
Response
Detail
Accept: text/plain
Accept: application/json
Content-Type: application/json
User-Agent: 5.6.4
host: [redacted]
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 366
JSON
[m:Auto]
[
{
"appLoads": {
"appID": "[redacted]",
"appVersion": "34",
"carrier": "Android",
"crPlatform": "android",
"crVersion": "5.6.4",
"deviceID": "dbfef840-7782-44a2-805e-397705645beb",
"deviceModel": "Android SDK built for x86",
"locale": "en",
[2/23]
?:help q:back [*:3030]

MiTM Attack

- Attacker needs to get a trusted certificate
  - Hacked CAs: DigiNotar (2011) & Comodo (2011)
- Or install his own certificate as trusted
  - < Android 7.0 : By default all installed certs are trusted for an app
  - Android 7.0 : only system installed certs are trusted
- Traffic can be read/altered by MitM

Mitigate MiTM

- SSL or Certificate Pinning within app
  - Option 1: pin on public keys
  - Option 2: provide your own trust store or certs
- Android 7.0+ has native support
  - network_security_config.xml

Secure Execution Environment

Problems

- Static code protection leads to dynamic attacks
- Rooted devices
- Three main attack techniques
  - Dynamic code injection a.k.a hooking
  - Attaching debuggers
  - Memory dumping

Dynamic Code Injection

- Tools: XPosed, Frida
- Requires rooted device
- Places hooks
  - E.g., before encryption calls, after decryption calls

Debuggers

- Tools: Java Debug Bridge (JDB), Gnu Project Debugger (GDB)
- Inspect code execution, paths, variables
- In Android alter AndroidManifest.xml > debuggable=true

Memory Dumping

- Tools: Linux Memory Extractor (LiME)
- Advanced security tools offer code encryption
- Code available in memory
- Dumping memory == getting unencrypted code

cat /proc/pid/maps

```
a3562000-a392d000 r--p 00000000 fd:20 35328      /data/data/example.com.classloading/app_outdex/code.dex
a392d000-a3bff000 r-xp 003cb000 fd:20 35328      /data/data/example.com.classloading/app_outdex/code.dex
a3bff000-a3c00000 rw-p 0069d000 fd:20 35328      /data/data/example.com.classloading/app_outdex/code.dex
```

Securing Your Environment

- Application can scan its environment
  - Should it run on a rooted device?
  - Should it run on an emulator - which is rooted by default?
- Detect dynamic code injection

SafetyNet API

- Get Google's opinion on the device status
- Response is JSON Web Signature (JWS)
- Developer needs to review response and verify signature
- `SafetyNetApi.attest()`

SafetyNet API

- SafetyNet looks at various device attributes (by @ikoz)
  - Installed packages
  - SU Files
  - Settings (adb enabled, lock screen enabled, …)
  - SE Linux state
  - Device admin blacklist
  - …

SafetyNet API

- Advantages
  - Google knows a lot
  - Updated remotely
  - Takes a lot into consideration

SafetyNet API

- Disadvantage
  - You only get a binary answer: compatible/ incompatible
  - Google Play Services dependency
  - Network requests take time
  - Developer needs to verify JWS

Conclusion

Conclusion

- Implement strong coding practices and strong cryptography
- Protect code statically through various layers that protect code and each other
- Harden the communications
- Scan, detect and protect against insecure execution environments

Q/A

References

- https://nelenkov.blogspot.be/2012/04/using-password-based-encryption-on.html
- https://android-developers.blogspot.nl/2013/08/some-securerandom-thoughts.html
- https://koz.io/inside-safetynet/
- Android Hacker's Handbook
- Android Security Internals
- www.guardsquare.com