Assessing and Improving Mobile Application Security

Carlton Northern

Michael Peck

March 2017

Approved for Public Release; Distribution Unlimited. Case Numbers 17-0968 and 16-4772.

About Us

- **The MITRE Corporation**
  – Not-for-profit organization
  – Operates federally funded research and development centers (FFRDCs)

- **Carlton Northern**
  – Chief Engineer at MITRE focusing on mobile security solutions

- **Michael Peck**
  – Security Engineer at MITRE primarily focusing on mobile security

Outline

- **Analyzing the Effectiveness of Mobile Application Vetting Tools**
  - Mobile Application Security Architecture
  - Analysis Criteria
  - Vulnerable and Malicious Mobile Apps for Testing
  - Analysis Results
  - Outcomes

- **Improving Android Application Security**
  - Contributions to the Android Open Source Project
    - Android app developer tools
    - Android platform security architecture

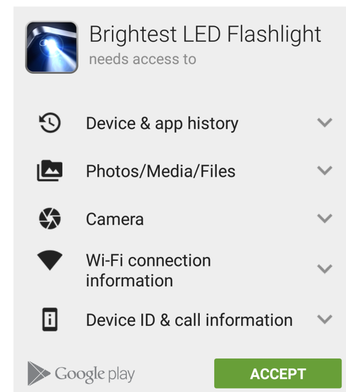
Overview: Analyzing Mobile App Vetting Tools

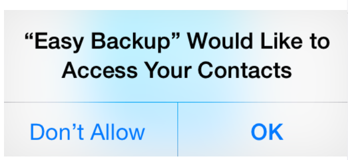
- **Goal: Analyze feasibility for enterprises to apply automated tools to determine whether apps are safe to use on mobile devices**
  - Ability for tools to identify security vulnerabilities
  - Ability for tools to identify potentially malicious or privacy violating behaviors
  - Integration of tools with Enterprise Mobility Management systems
  - Capability for reputation analysis of apps and app developers

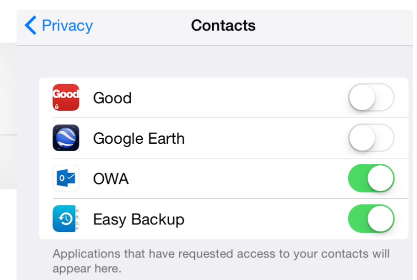
- **Methodology:**
  - Understand current Android and iOS app security architecture and initiatives
  - Formulate analysis criteria
  - Develop test apps that demonstrate vulnerable and malicious behavior that map to the criteria
  - Assess a number of commercial and free mobile app vetting solutions by scanning the test apps

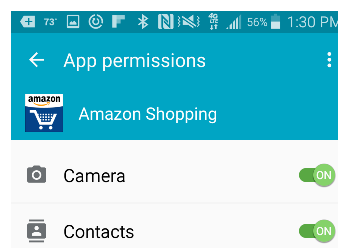
Mobile Application Security Architecture

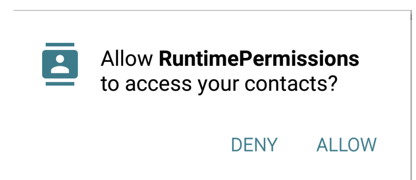
Numerous mitigations are inherently provided by the security architecture of mobile devices











- **Apps are sandboxed from each other and underlying system**
- **Apps must request and obtain permission to access sensitive resources**

Mobile Application Security Architecture Enhancements

- **Continuing to evolve in response to common app vulnerabilities**
  - Will discuss later
- **Evolving in response to emerging threats, malicious behaviors**
  - Changes to Android Device Admin API in response to ransomware (Android 6)
  - Remove ability for Android apps to see MAC addresses, other processes (Android 6 / 7)
  - Runtime permission requests (Android 6)
  - Apple iOS restrictions on installing non-App Store apps (iOS 9)
- **Additional information**
  - Google I/O 2016: What's new in Android security (M and N) video
    - https://www.youtube.com/watch?v=XZzLjllizYs
  - Apple WWDC 2016: What's New in Security video
    - https://developer.apple.com/videos/play/wwdc2016/706/

Security architecture enhancements can be leveraged during app security testing

Analysis Criteria

- **National Information Assurance Partnership's (NIAP) Protection Profile (PP) for Application Software**
  - Security criteria requirements for software applications (mobile, desktop, server) common criteria evaluation.
  - Security requirements focusing on encryption, access to platform resources (information repositories and hardware), use of PII, configuration and anti-exploitation
  - https://www.niap-ccevs.org/profile/Info.cfm?id=394

- **Our criteria is based on the NIAP PP for Application Software**
  - 16 out of the PP's 25 mandatory security functional requirements
  - 6 requirements may not be automatable and 3 not necessary on Android and iOS

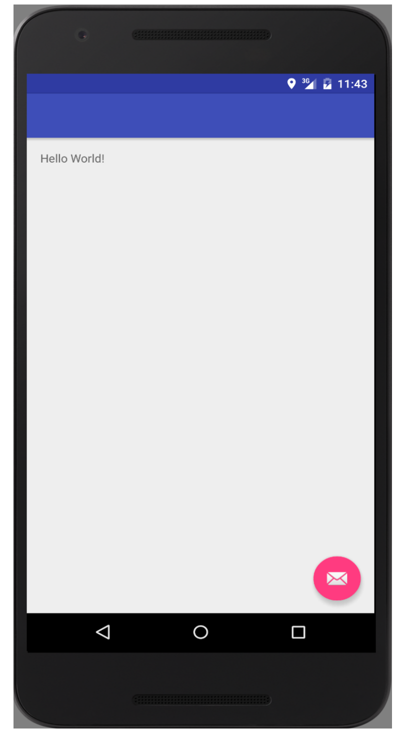
- **Ability to identify security vulnerabilities, e.g.:**
  - Cryptographic issues (e.g. randomness, key storage)
  - Insecure data storage
  - Insecure network communication
  - Memory mappings
  - Third-party library issues
  - Inter-process communication issues

Analysis Criteria (cont'd)

- **Ability to identify potentially inappropriate behaviors, e.g.:**
  - Access to hardware resources and sensitive repositories
  - Dynamic code execution
  - Report all network communication
  - App includes well-known device exploit code
  - iOS URL scheme hijacking
  - Requests Android Device Administration access
- **Security of the app vetting system itself**
  - Ability to resist analysis environment detection by malicious apps
  - Doesn't reveal information about other apps under analysis in multi-tenant environment
- **Reporting capabilities**
  - Supported output formats
  - APIs
  - Integration with EMM/MDM systems

Android Vulnerable and Malicious Test Apps

- **UploadDataApp**
  - Grabs lots of sensitive data and sends to remote server
  - Uses both HTTP and HTTPS with cert validation disabled
  - Etc.
- **CustomClassLoader**
  - Modified sample app from Google
  - Downloads and executes .DEX and .SO files
  - Downloads and stores files insecurely
- **DeviceAdminReceiver**
  - Google sample app that activates Device Admin
- **App with older version of OpenSSL embedded**

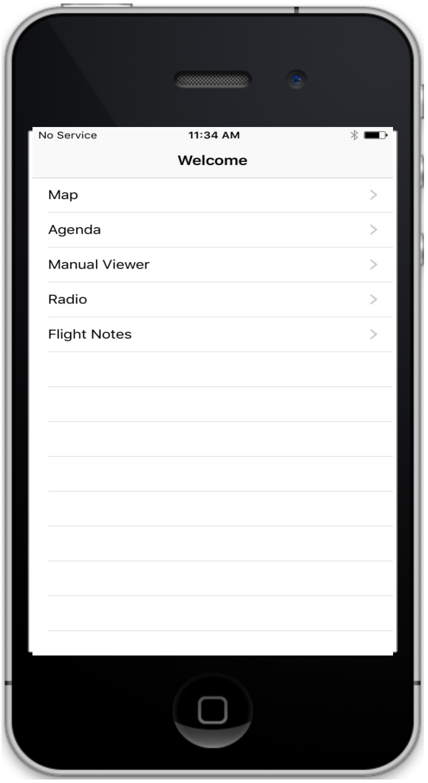


Available soon at https://mitre.github.io/vulnerable-mobile-apps/

MITRE

iOS Vulnerable and Malicious Test App

- **AcmeAirlines**
  - Insecure network communications
  - Insecure storage
  - Collects data and send to remote server
  - Dynamic code execution with JSPatch*
  - URI scheme hijacking
  - Time-bomb exploit
  - Etc.



*https://www.fireeye.com/blog/threat-research/2016/04/rollout_or_not_the.html

Vendors / Products Included in Evaluation

- **Selection Criteria:**
  - Gartner's *Application Security Testing Magic Quadrant 2015*
  - Gartner's *Critical Capabilities for Application Security Testing 2015 – Mobile App Testing*
  - Inclusion of NIAP Protection Profile for Application Software requirement checks
  - Also include free tools that are easy to obtain / integrate

- **Tools:**
  - Android Lint (Included in Android Studio and Android SDK)
  - 8 other commercial products

Android Test Results

Assessment Criteria	Android Lint	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7	Product 8
3A Static IV for Encryption									
3B Cleartext Password File Storage									
3C Insecure Internal File Storage									
Insecure External File Storage									
3D Report Network Destinations and Ports									
Sensitive Data Cleartext									
Certificate Checking & Hostname Verify									
3E Embedded Default Credentials									
3F Memory Mapping Explicit Locations									
3G Memory Mapping Write and Execute									
3H Latest OS Anti-exploitation									
3J Executable Code Storage									
3K Stack-based Buffer Overflow Protection									
3L Identify 3rd Party Libraries									
3M Other Crypto Issues									
3N Inter-app Communication Security Issues									
4A Device Resource Permissions									
4B Sensor Access									
Sensitive Information Acess									
4D Dynamic Code Execution									
4E Use of Private/Unsupported APIs									
4F Obfuscation Detection									
4G Identify Known Malicious Code									
4H Device Administrator Access									
5A Detect Analysis Environment									
5B Multi-tenant Concerns									
6A Output formats									
6B Provide Evidence of Findings									
6C Enterprise Integration capabilities									

MITRE

iOS Test Results

Assessment Criteria	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7	Product 8
3A Static IV for Encryption							
3B Cleartext Password File Storage							
3C nsecure Internal File Storage							
3D Report Network Destinations and Ports							
Sensitive Data Cleartext							
Certificate Checking & Hostname Verify							
3E Embedded Default Credentials							
3H Latest OS Anti-exploitation							
3L Identify 3rd Party Libraries							
3M Other Crypto Issues							
3N / inter-app Communication Security							
4A Device Resource Permissions							
4B Sensor Access							
Sensitive Information Acess							
4D Dynamic Code Execution							
4E Use of Private/Unsupported APIs							
4F Obfuscation Detection							
5A Detect Analysis Environment							
5B Multi-tenant Concerns							
6A Output formats							
6B Provide Evidence of Findings							
6C Enterprise Integration capabilities							
7A Unsanitized Input							
7B Code Coverage							

Overall Results

- **Feasible to detect and identify many common security vulnerabilities**
- **Best solutions performed a combination of static and dynamic analysis**
  - Both are required to get a full picture of app properties and actual runtime behavior
- **Identifying vulnerabilities vs. identifying malicious behavior are different use cases – many vendors focus on one or the other**
- **Detecting malicious behavior is a much harder problem**
  - Easy for malicious app to detect presence of analysis environment
    - e.g. Presence of Xposed Framework (Android), Cydia (iOS)
  - Malicious apps can dynamically download and execute harmful code at runtime (including iOS)
  - Recommend continued investigation into reputation analysis capabilities
- **Vendors are starting to incorporate the NIAP App PP requirements into their analysis and reports**

Outcomes

- **Repeatable criteria, process, and example apps suitable for testing effectiveness of app vetting tools**
- **Results of applying criteria to leading industry tools**
- **Provided feedback to help vendors improve their products**
- **Feedback to NIAP on streamlining the Protection Profile for Application Software**
  - Decrease time and cost of app evaluations
  - Rely on device security architecture where possible
  - Prefer requirements/tests that are automatable

Overview: Improving Android Application Security

- **Users and enterprises want assurance that applications installed on their mobile devices can be safely used**
  - But individually assessing application security can be time consuming and expensive

- **We will describe efforts to improve confidence in Android app security**
  - Integrate security checks into the app development process to help developers follow best practices and avoid common mistakes
  - Build upon Android's platform security architecture to:
    - decrease likelihood of code weaknesses
    - prevent exploitation of vulnerabilities

- **Contributions made to the Android Open Source Project**
  - **Open to external contributors**
  - **Android app developer tools**
  - **Android platform security architecture (SELinux policies)**

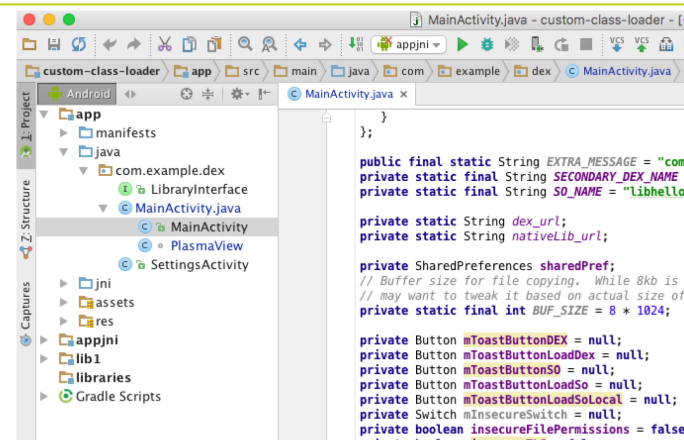
MITRE

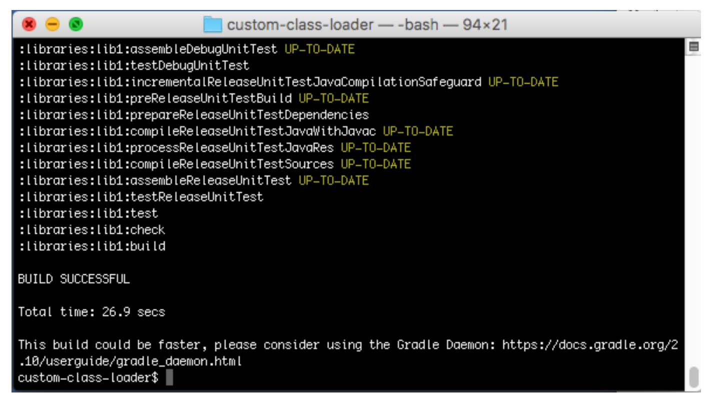
Mobile Application Development Tools

- **Android Studio and the Android Software Development Kit (SDK) are commonly used by app developers**

- **Android Lint – Part of the Android Open Source Project**
  - Static analysis tool integrated into Android Studio and the Android SDK

- **Android Lint (and similar tools) can:**
  - Alert developers to security weaknesses early in their development lifecycle when they are easiest and cheapest to fix
  - Encourage developers to comply with best practices





Android Platform Security Architecture

- **Android's security architecture provides inherent protection against exploitation of many common app vulnerabilities, and protection from malicious actions by apps**

- **Apps are sandboxed from each other and from underlying system**
- **App developer must declare properties up-front in manifest**
  - Apps must request and obtain permission to access sensitive resources
- **Not a complete solution – but important to understand its benefits and take into account when assessing app security**

Examples of Common Mobile App Security Issues

- Insecure Network Communication (OWASP Mobile Top Ten: M3)
- Insecure Data Storage (OWASP Mobile Top Ten: M2)
- Insecure Dynamic Code Execution

(Not intended to be a comprehensive list)

Insecure Network Communication

- **Problem: Network communication weaknesses are regularly found in mobile apps, and they can be easy to exploit because mobile devices are often used on unprotected networks (e.g. public Wi-Fi)**
  - Plaintext network communication
  - Java TrustManager overridden with insecure version that skips certificate validation
  - Java HostnameVerifier overridden with insecure version

Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security

Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith
Distributed Computing & Security Group
Leibniz University of Hannover
Hannover, Germany
{fahl,harbach,muders,smith}@dcsec.uni-hannover.de

Lars Baumgärtner, Bernd Freisleben
Department of Math. & Computer Science
Philipps University of Marburg
Marburg, Germany
{lbaumgaertner, freisleb}@informatik.uni-marburg.de

FireEye | Customer Stories | Blogs
Products | Solutions | Mandiant Consulting | Current Threats | Partners | Su
Home › FireEye Blogs › Threat Research › August 2014 Threat Research Blog Posts ›
SSL Vulnerabilities: Who listens when Android appl...

SSL VULNERABILITIES: WHO LISTENS WHEN ANDROID APPLICATIONS TALK?

August 20, 2014 | by Adrian Mettler, Yulong Zhang, Vishwanath Raman | Mobile Threats, Threat Research

RSAConference2015
San Francisco | April 20-24 | Moscone Center
SESSION ID: HTA-T08

How We Discovered Thousands of Vulnerable Android Apps in 1 Day

Joji Montelibano
Vulnerability Analysis Technical Manager
CERT
@certcc

Will Dormann
Vulnerability Analyst
CERT
@wdormann

The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software

Martin Georgiev, The University of Texas at Austin
Subodh Iyengar, Stanford University
Suman Jana, The University of Texas at Austin
Rishita Anubhai, Stanford University
Dan Boneh, Stanford University
Vitaly Shmatikov, The University of Texas at Austin

Security Concerns in Android mHealth Apps

Dongjing He, Muhammad Naveed, Carl A. Gunter, Klara Nahrstedt
Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL

ER: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps

David Sounthiraraj, Justin Sahs, Garret Greenwood, Zhiqiang Lin, Latifur Khan
Department of Computer Science, The University of Texas at Dallas
{david.sounthiraraj, justin.sahs, garrett.greenwood, zhiqiang.lin, lkhan}@utdallas.edu

MITRE

Insecure Network Communication Solutions: App Development Process

- **We contributed new checks to Android Lint**
  - Detect insecure TrustManager
  - Detect insecure HostnameVerifier
  - Detect insecure SSLCertificateSocketFactory

```
// from http://blog.denevell.org/android-trust-all-ssl-certificates.html
TrustManager[] trustAllCerts = new TrustManager[] {
    new X509TrustManager() {
        @Override
        public void checkClientTrusted(X509Certificate[] certs, String authType) { }

        @Override
        public void checkServerTrusted(X509Certificate[] certs, String authType) { }
```

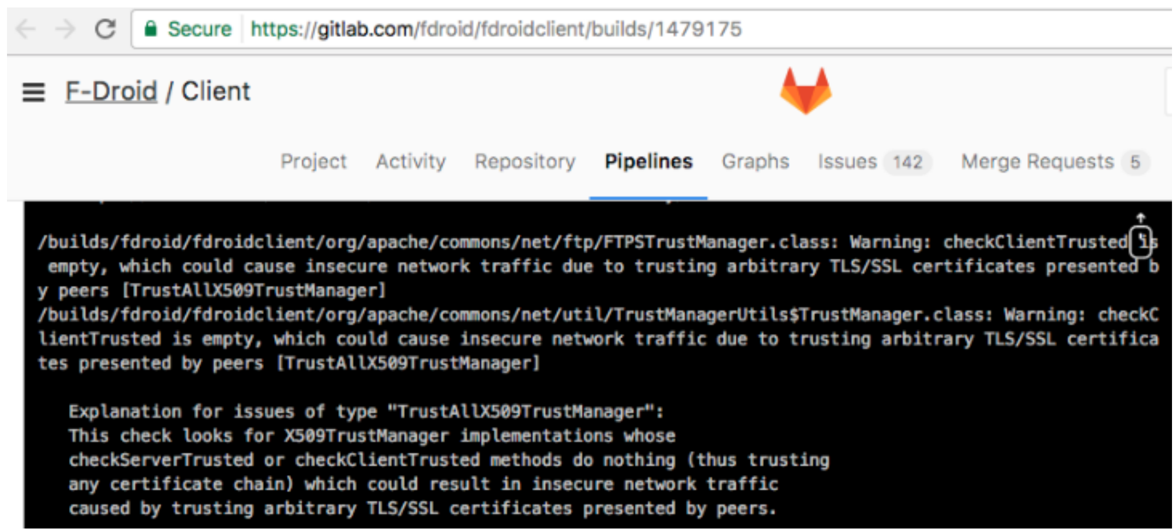
checkServerTrusted is empty, which could cause insecure network traffic due to trusting arbitrary TLS/SSL certificates presented by peers more... (⌘F1)

MITRE

Insecure Network Communication Solutions: App Development Process

The lint checks are being used by real projects:

Used in Continuous Integration (CI) builds



Insecure Network Communication Solutions: App Development Process

Good news: Developers are using Android Lint and its security checks

Bad news: Stack Overflow advice may evolve to work around the checks

stackoverflow.com/questions/39352818/retrofit-2-unable-to-resolve-host

```
public class CustomX509TrustManager implements X509TrustManager {

    @SuppressLint("TrustAllX509TrustManager")
    @Override
    public void checkClientTrusted(X509Certificate[] chain, String a
    }

    @SuppressLint("TrustAllX509TrustManager")
    @Override
    public void checkServerTrusted(X509Certificate[] chain, String a
    }
```

DO NOT DO THIS!!!!

(without a really good reason)

MITRE

Insecure Network Communication Solutions: App Development Process: Google Play Enforcement

https://developer.android.com/google/play/asi.html

ers DESIGN DEVELOP DISTRIBUTE Search

App Security Improvement Program

GitHub, Inc. [US] https://github.com/aws/aws-sdk-android/issues/160

Play Store Security Alert on X509TrustManager #160

Closed jullylau opened this issue on Jun 23, 2016 · 1 comment

Secure https://support.google.com/faqs/answer/6346016

How to fix apps containing an unsafe implementation of TrustManager

GitHub, Inc. [US] https://github.com/aws/aws-sdk-android/blob/master/CHANGELOG.md#release-2218-06022016

Release 2.2.18 (06/02/2016)

Bug Fixes

- **AWS Core Runtime Library:** Removed testing implementation for `X509TrustManager`, for more information see.

MITRE

Insecure Network Communication Solutions: Platform Security Architecture

- **Android 7 and up: Network Security Configuration**
  - App developer declares app's network security properties in an XML file
  - Reduces need for app developer to muck with security sensitive code
  - Policies can be easily examined by app stores, security assessors
  - Caution: Policies may not be enforced by third-party networking libraries

- **iOS 9 and up: App Transport Security (ATS)**
  - Enforces encrypted communication by apps and compliance with best practices
    - Enabled by default, apps must explicitly opt-out
  - Apple may enforce App Transport Security as a condition for App Store publication
    - Planned for January 2017, but delayed
  - 80% of top 50 iOS apps opt-out from HTTPS requirement (NowSecure – 08/2016)
  - 97% of top 200 iOS apps opt-out from at least one aspect of ATS (Appthority – 12/2016)

Insecure Data Storage

- **Problem: Android's default file permissions prevent apps from reading or writing files belonging to other apps. However, apps may set their files world readable or world writable, either inadvertently or due to a desire for data sharing with other apps.**

NowSecure™

The Poweramp Music Player app uses lax file permissions for preference files and some of its executable code.

```
root@hammerhead:/data/data/com.maxmpz.audioplayer/files # ls -l
-rw-rw-rw- u0_a96    u0_a96    1187312 2015-07-30 13:43 libaudioplayer_native.so
-rw-rw-rw- u0_a96    u0_a96     690168 2015-07-30 13:43 libpampffmpeg.so

root@hammerhead:/data/data/com.maxmpz.audioplayer/shared_prefs # ls -l
-rw-rw-rw- u0_a96    u0_a96        372 2015-07-30 12:43 PlayerService.xml
-rw-rw--- u0_a96    u0_a96        130 2015-07-29 16:32 _has_set_default_values.xml
-rw-rw-rw- u0_a96    u0_a96       8508 2015-07-29 16:32 com.maxmpz.audioplayer_preferences.xml
-rw-rw-rw- u0_a96    u0_a96       1103 2015-07-29 16:32 eq.xml
-rw-rw--- u0_a96    u0_a96        101 2015-07-29 16:32 l.xml
```

[Updated] Exclusive: Vulnerability In Skype For Android Is Exposing Your Name, Phone Number, Chat Logs, And A Lot More

Justin Case
Apr 14, 2011

G+5 f 22

```
# ls -l /data/data/com.skype.merlin_mecha/files/jcaseap
-rw-rw-rw- app_152  app_152    331776 2011-04-13 00:08 main.db
-rw-rw-rw- app_152  app_152    119528 2011-04-13 00:08 main.db-journal
-rw-rw-rw- app_152  app_152     40960 2011-04-11 14:05 keyval.db
-rw-rw-rw- app_152  app_152      3522 2011-04-12 23:39 config.xml
drwxrwxrwx app_152  app_152           2011-04-11 14:05 voicemail
```


MITRE

Insecure Data Storage Solutions: App Development Process

- Android Lint already included checks to identify use of MODE_WORLD_READABLE and MODE_WORLD_WRITEABLE

```
public void loadDex(Activity ctx, File nativepath) {

    File optimizedDexOutputPath;
    if (insecureFilePermissions)
      optimizedDexOutputPath = getDir("outdex", Context.MODE_WORLD_READABLE|Context.MODE_WORLD_WRITEABLE);
```

Using MODE_WORLD_READABLE when creating files can be risky, review carefully less... (⌘F1)

There are cases where it is appropriate for an application to write world readable files, but these should be reviewed carefully to ensure that they contain no private data that is leaked to other applications.

'MODE_WORLD_READABLE' is deprecated less... (⌘F1)

This inspection reports where deprecated code is used in the specified inspection scope.

- We expanded the cases covered by the existing checks, and added new checks for setReadable and setWritable

```
if (insecureFilePermissions) {
    nativePath.setReadable(true, false);
    nativePath.setWritable(true, false);
```

Setting file permissions to world-writable can be risky, review carefully less... (⌘F1)

Setting files world-writable is very dangerous, and likely to cause security holes in applications. It is strongly discouraged; instead, applications should use more formal mechanisms for interactions such as ContentProvider, BroadcastReceiver, and Service.

MITRE

Insecure Data Storage Solutions: Platform Security Architecture

- **Security improvements in Android 7**
  - For apps that target compatibility with Android 7 and up (targetSdkVersion >= 24)
  - App data directories are now mode 0700 by default (-rwx------)
    - Blocks access by other apps to files, even when those files have insecure permissions
    - However, doesn't stop an app from changing permissions of its own data directory
  - MODE_WORLD_READABLE / WRITEABLE not allowed

- **Our proposed next step**
  - Apply SELinux Mandatory Access Control policies to block access to other apps' files regardless of file and directory permissions
    - Phase in by applying to all apps targeting a particular API level and higher (targetSdkVersion)
      - e.g. https://android-review.googlesource.com/#/c/195590/
    - Developers can still use Android Content Provider for controlled data sharing between apps

Dynamic Code Execution

- **Problem: Apps can download and execute new code not included in the original application package**
- **Vulnerable apps**
  - When combined with insecure network communication or insecure file permissions, an adversary can replace the dynamic code with something malicious
  - e.g. Vulnerabilities discovered by NowSecure in Poweramp, SwiftKey, Vungle apps
- **Malicious apps**
  - Deliberately download and execute exploit code after installation to evade security reviews
  - e.g. BeNews Android app allegedly leaked from Hacking Team, Poeplau et al. (NDSS ’14), Victor van der Veen’s Android Security Symposium talk

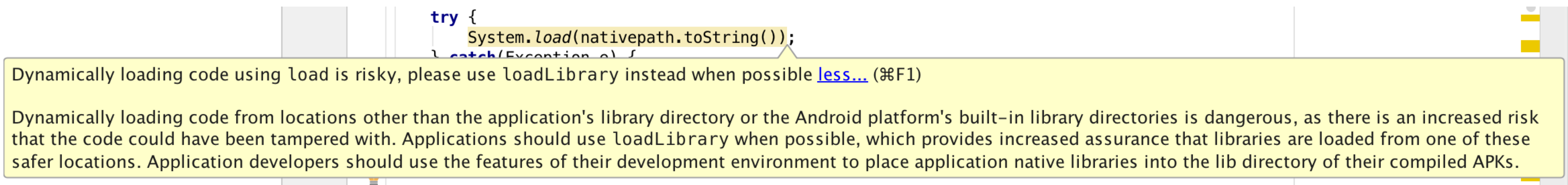
https://developer.android.com/distribute/essentials/quality/core.html#sc

Core App Quality

Execution	SC-E1	App does not dynamically load code from outside the app's APK.

Dynamic Code Execution Solutions: App Development Process

- **We contributed Android Lint rules to encourage app developers to follow best practices**
  - Use loadLibrary, not load. loadLibrary constrains the locations that native code can be loaded from.



  - Detect ELF binaries in the app package outside of the *lib* directory and encourage the developer not to do that.

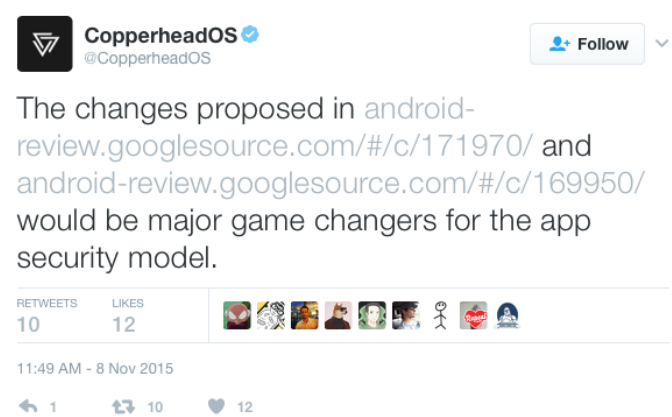
Dynamic Code Execution Solutions: Platform Security Architecture

- **Best Practice: Native shared libraries should be in the app package's *lib* directory**
  - At app install time, Android's Package Manager extracts these into */data/app-lib*
  - Apps themselves cannot modify the libraries
- **Unfortunately, some apps do not follow this best practice**
  - March 2014: **71** out of **2420** top Google Play apps had an executable or shared library in the APK outside of *lib* directory (we recommend performing an updated analysis)

```
root@hammerhead:/data/data/com.maxmpz.audioplayer/files # ls -l
-rw-rw-rw- u0_a96    u0_a96    1187312 2015-07-30 13:43 libaudioplayer_native.so
-rw-rw-rw- u0_a96    u0_a96     690168 2015-07-30 13:43 libpampffmpeg.so
```

(from NowSecure example on previous slide)

- **Our proposal: Enforce this best practice through SELinux policy**
  - Prevent apps from executing code from locations that they can write to
  - Phase in based on app's targetSdkVersion



MITRE

Dynamic Code Execution: Challenges with Modifying Platform Security Architecture

- **Compatibility issues**
  - Apps with shared libraries outside *lib* directory
  - Apps that embed native executables

- **DexClassLoader and Android Runtime (ART)**
  - *dex2oat* runs in app's context, compiles Dalvik bytecode into native code
  - If execution is blocked, the compiled native code can't run
    - The app still works -- Android falls back to use an interpreter to execute the bytecode

- **Apps could still map memory as writable and executable**
  - Copperhead Security proposed addressing this using PaX MPROTECT
    - Can also address with SELinux execmem
  - But restricting executable memory introduces compatibility concerns
    - JIT compilers within web browsers and within the Android Runtime (ART)
    - See our paper for more details

MITRE

Recent Android Open Source Project Changes

- **Changes in AOSP master – not all are in a released Android version yet**

- **Ability to set SELinux domain based on app's targetSdkVersion**
  - Allows phasing in new security policies applied to apps
  - Based on our proposed code
  - https://android-review.googlesource.com/#/q/status:merged+project:platform/system/sepolicy+branch:master+topic:selinux-targetSdkVersion

- **New untrusted_v2_app and ephemeral_app SELinux domains with stricter security policies**
  - Enforces stronger protection on app internal data storage directory
  - Addresses dynamic code execution by preventing execution of app data files
  - https://android.googlesource.com/platform/system/sepolicy/+/master

MITRE

Conclusions and Potential Future Work

- **Mobile platforms provide a security architecture that can leveraged and built upon to gain confidence in and improve mobile app security**
- **Developer behavior can be influenced through the mobile app development process**
  - Android Lint (and other tools) can help developers avoid mistakes, follow best practices
  - Google Play Store (and other app stores) can enforce compliance
- **Potential Future Work**
  - Incentives for app developers to actually use new platform security features
    - e.g. Target the latest Android API level and use Network Security Configuration feature
  - Tools to help app developers use new platform security features
    - e.g. Android Studio feature to help developers write Network Security Configuration policies
  - More Android Lint security checks
  - Continue strengthening Android security policies to reduce attack surfaces, prevent exploitation of vulnerabilities

MITRE

Resources for More Information

- **Our technical reports**
  - https://www.mitre.org/publications/technical-papers/android-security-analysis-final-report
  - https://github.com/mitre/vulnerable-mobile-apps/raw/master/analyzing-effectiveness-mobile-app-vetting-tools.docx

- **Our source code**
  - https://mitre.github.io/vulnerable-mobile-apps/

- **Contact information**
  - cnorthern@mitre.org and mpeck@mitre.org