QARK:
Android App Exploit and SCA tool

Who are we?

Tony Trummer
Staff Engineer, Information Security at LinkedIn
Penetration tester and mobile security enthusiast
#3 in Android Security Acknowledgements

Tushar Dalvi
Sr. Security Engineer at LinkedIn
Penetration Tester

Responsible for securing a large suite mobile apps

What is this about?

QARK

QUICK ANDROID REVIEW KIT

A new tool to test apps for vulnerabilities and automate exploitation

Agenda

1. Review of reversing APKs
2. Review of Android app structure
3. Review of Android components
4. Review of attack surfaces and vectors
5. Review of current tools
6. QARK introduction and demonstration
7. Lab time for hands-on

Android Challenges

Long-tail of supported versions
Ship-once, own forever
Pace of development
Numerous inter-app communication methods
Plenty of baked-in gotchas
Poor documentation

The Good News

The known app attack surface is relatively small and largely transparent

The AndroidManifest.xml file reveals many of the potential vulnerabilities

Java is a known quantity - plenty of tools to examine the Java code

App Structure

APKs
Reversing APKs
Code Structure

App Structure

APKs

- Compressed
- Compiled
- Signed

App Structure

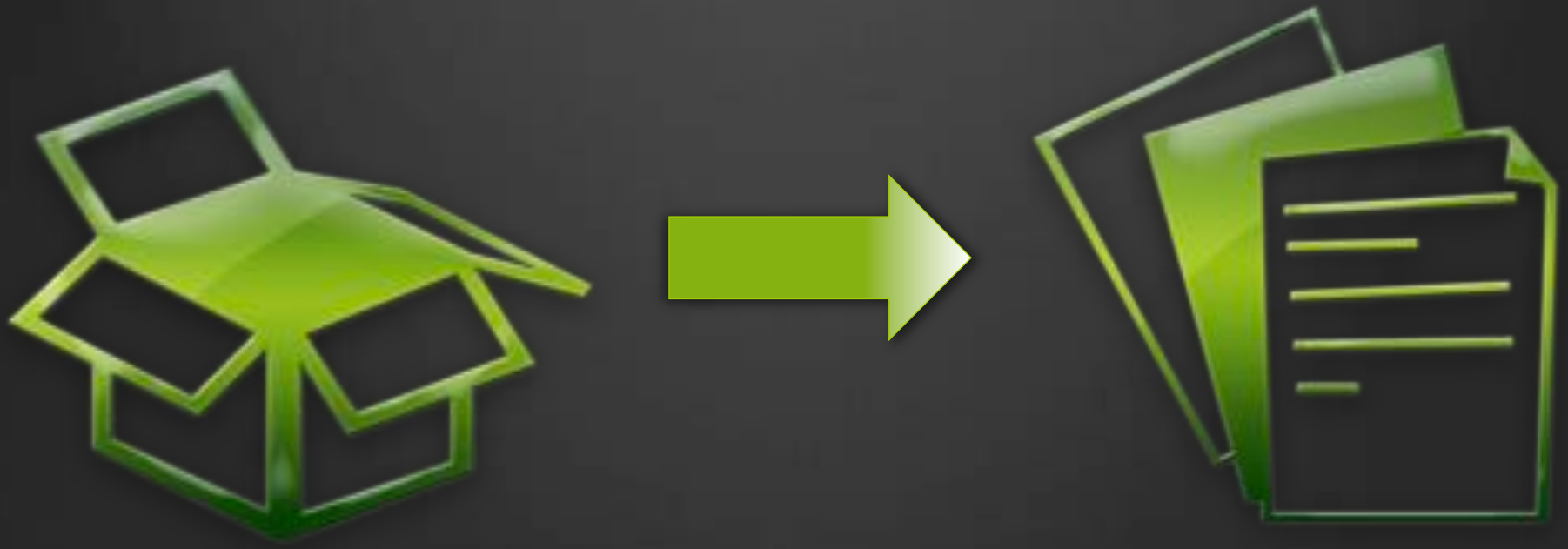
Reversing APKs

- apktool
- dex2jar
- JD-GUI

App Structure

Reversing APKs

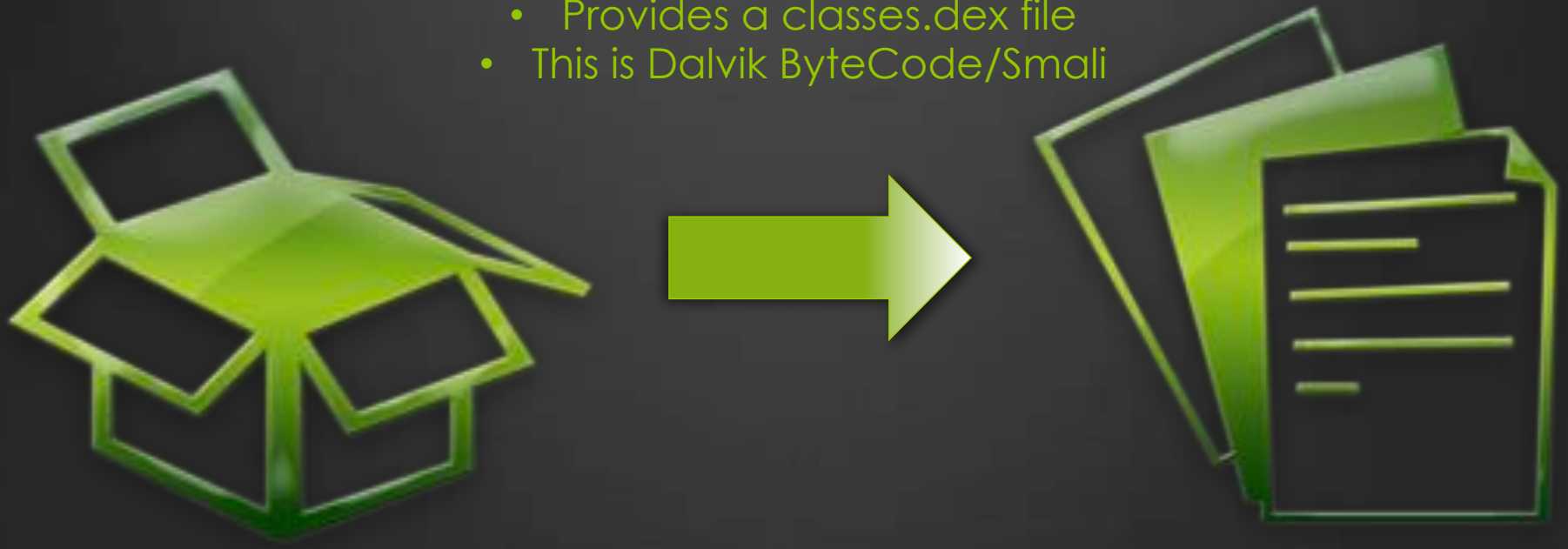
- apktool d foo.apk
- Provides readable AndroidManifest.xml



App Structure

Reversing APKs

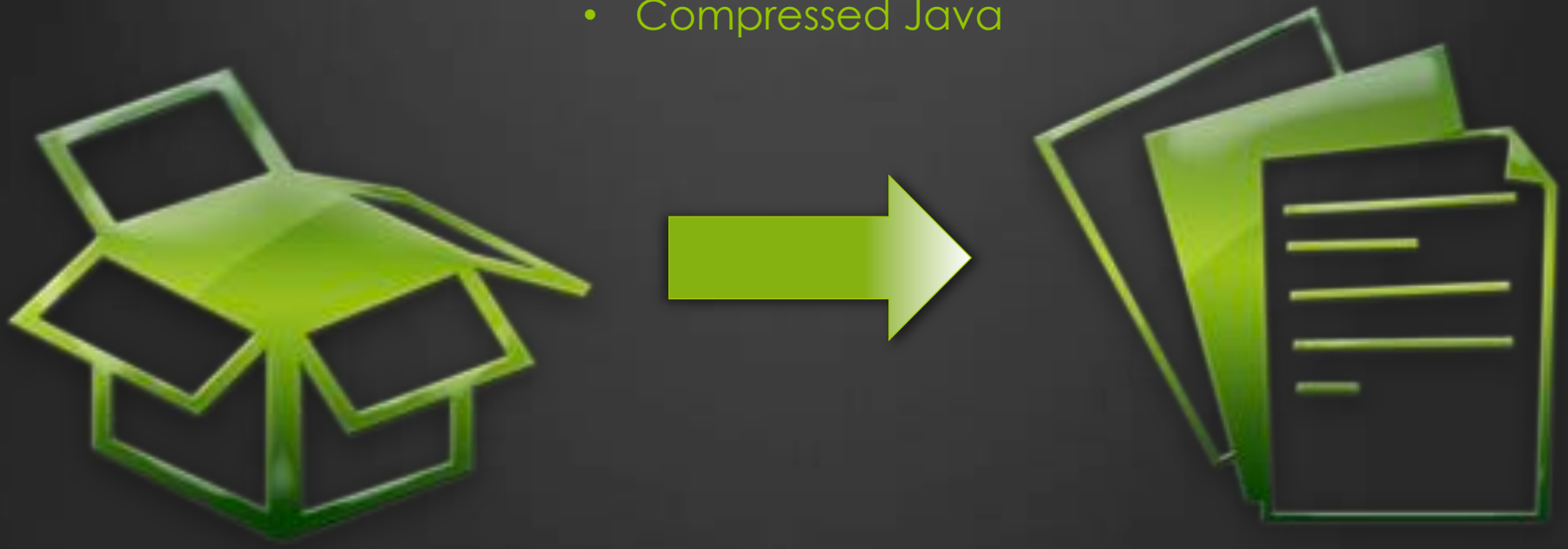
- cp foo.apk foo.zip
- unzip foo.zip
- Provides a classes.dex file
- This is Dalvik ByteCode/Smali



App Structure

Reversing APKs

- dex2jar.sh classes.dex
- Gives you classes_dex2jar.jar
- Compressed Java



App Structure

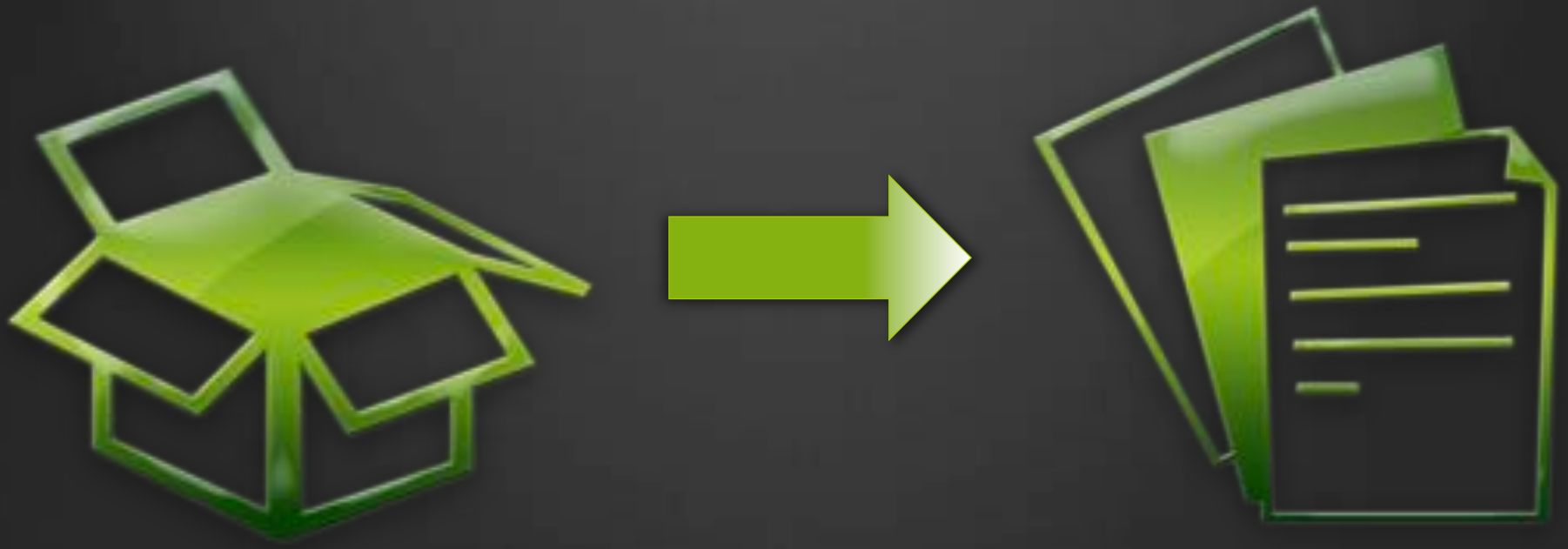
Reversing APKs

- Use JD-GUI to open classes_dex2jar.jar
- Choose ***Save All Sources*** from the ***File*** menu
- Creates classes_dex2jar.src.zip
- unzip classes_dex2jar.src.zip

App Structure

Code Structure

- AndroidManifest.xml
- Java



App Structure

AndroidManifest.xml

android:name: *fooActivity*

1-to-1 Mapping

Java class

fooActivity.java

Attack Surface

AndroidManifest.xml:

Permissions
Content Providers
Services
Activities
Receivers
Intent Filters

Others:

Pending Intents
WebViews
Local Files

Attack Surface

AndroidManifest.xml

- Defines most of the attack surface
- minSdkVersion tradeoff

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
android"
    package="com.foo.android"
    android:installLocation="auto"
    android:versionCode="125"
    android:versionName="3.4.6" >
```

Attack Surface

Permissions

- Protection Levels:
  - normal
  - dangerous
  - signature
  - signatureOrSystem
- Can declare custom permissions
  - Protect custom permissions with signatures
- Even signature based permissions can be stolen (pre-Lollipop)

```
<permission
    android:name="com.linkedin.android.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
```

Attack Surface

IPC Mechanisms

- Intents
  - Explicit vs. Implicit
  - Broadcast
  - Used to start Activity, Service or deliver Broadcast
  - Bundle/Extras
- AIDL
- Binder

```
<intent-filter>
 <action android:name="android.accounts.AccountAuthenticator" />
</intent-filter>
```

Attack Surface

Pending Intents

- Similar to callbacks
- Allow apps to act as one another

```
final PendingIntent contentIntent = PendingIntent.getBroadcast
        (this, notificationId, clickIntent,
PendingIntent.FLAG_CANCEL_CURRENT);
```

Attack Surface

Intent Filters

- Not a security feature
- Often causes unintended exporting of features

```
<intent-filter>
 <action android:name="android.accounts.AccountAuthenticator" />
</intent-filter>
```

Attack Surface

Activities

- How users interact with the app

```
<activity
        android:name="com.foobar.android.home.v2.ShareActivity"
        android:configChanges="keyboardHidden | orientation"
        android:screenOrientation="portrait"
        android:windowSoftInputMode="stateHidden | adjustResize" >
</activity>
```

Attack Surface

Activity LifeCycle

Attack Surface

Fragment LifeCycle

Attack Surface

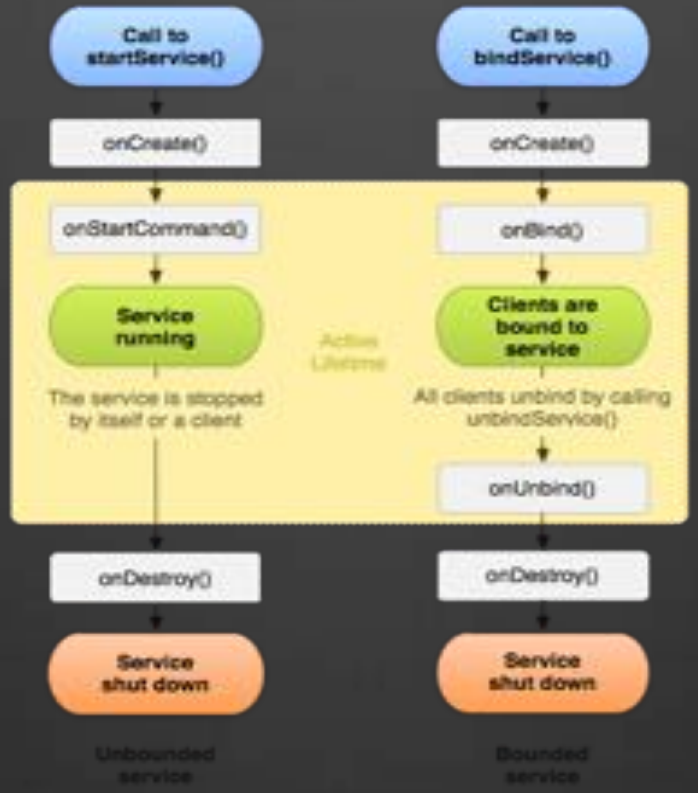
Services

- Processes that run in the background without a UI

```
<service
        android:name="com.foobar.android.foo.AuthService"
        android:exported="true”>
          <intent-filter>
            <action android:name="com.foo.android.auth.DO_STUFF" />
          </intent-filter>
</service>
```

Attack Surface

Service LifeCycle



Attack Surface

Receivers

- Listen for events

```
<receiver android:name=".authenticator.AccountChangeReceiver">
        <intent-filter >
            <action android:name="android.accounts.LOGIN"/>
        </intent-filter>
</receiver>
```

Attack Surface

Content Providers

- Creates interface to app data
- Usually SQLite DB

```
<provider
        android:name=".provider.FoobarProvider"
        android:authorities="com.foobar.android"
        android:exported="”true"
        android:label="@string/foobar_data" >
</provider>
```

Attack Surface

WebViews

- A horrible idea
- Build your own browser
- Can potentially access files and content providers
- Can potentially interact with Java classes
- Can run JavaScript and other plugins
- Same Origin Policy bypasses
- On-device HTML templates

```
private WebView fooWebView;
```

Attack Vectors

Malicious Apps

- Malicious Intents (injection attacks)
- Pending Intent addition
- Broadcast Intent interception
- Intent spoofing
- Implicit Intent interception
- Permission squatting

Attack Vectors

Malicious Web Content

- OWASP top 10
- Deeplinking

Attack Vectors

Remote Attackers

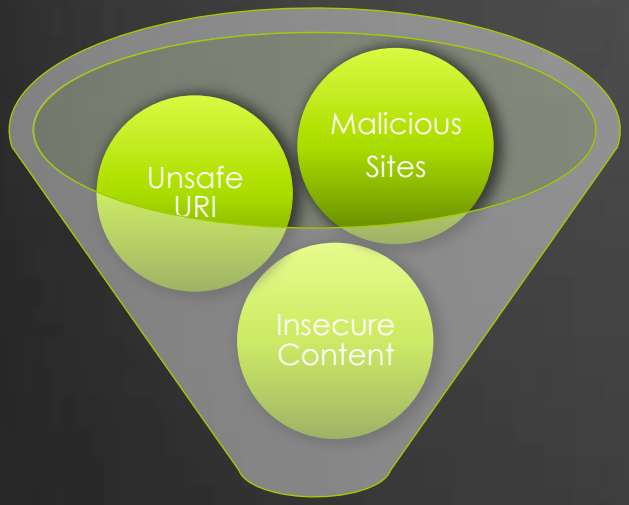
- Insecure communications
- Improper certificate validation

Attack Vectors

Local Attackers ☹

- Per FTC, 50% of users don't set PIN (I'm skeptical)
- Difference of perception (FUD + Media)
- FDE is available/default (now)
- Debugging enabled ? Then, turn it off
- World readable files
- World writeable files – injection

WebView weakness



WebView

Javascript

Bridge

Java classes

Exposed methods

Attack Surface

Local Files

- World readable SDCARDS
- World readable/writeable files
- World readable log files

Common Vulnerabilities

Insecure WebView content
Improper Certificate Validation
Insecure URL handling
Insecure Pending Intents
Insecure Data Storage
SQL Injection

Existing Tools

- Drozer: pretty good, reads manifest to determine attack surface, can be used for advanced exploitation
- ADB: A debugger, log viewer, provides a shell and can send Intents manually
- IDE: Can report some vulnerabilities during build and view logs

ADB

- adb shell – CLI shell
- adb push/pull – move files
- adb root – restarts daemon as root
- adb shell pm list packages – shows installed apps
- adb shell pm path com.foo.bar

ADB

- # specifying the action and data uri

  adb shell am start -a "android.intent.action.VIEW" -d "http://developer.android.com"

- # specifying the action, mime type and an extra string

  adb shell am start -a "android.intent.action.SEND" --es "android.intent.extra.TEXT" "Hello World" -t "text/plain"

- # specifying an explicit component name

  adb shell am start -n "com.example.application/.MainActivity"

- # specifying an explicit component name

  adb shell am startservice –n "com.example.application/.BackgroundService"

- # specifying the action

  adb shell am broadcast –a "android.intent.action.PACKAGE_FIRST_LAUNCH" -d "com.example.application"

Thanks: http://xgouchet.fr/

QARK

A lazy tester's friend

- Attempts to improve on these tools
- Can be used for attacking or auditing
- Written in Python
- Combination of XML parser and Android (Java) SCA

Current Tools

	Strengths	Weakness
Drozer	• Ease of (basic) use • Exploitation options	• Not SDLC friendly • Free version limited • Unfamiliar to devs • Poor Docs • Requires Android knowledge
COTS tools	• Thorough – in theory • Well maintained?	• Expen$ive • Many are geared toward forensics • Little/no POC support or exploit options

Why Use QARK?

Strengths	Weakness
• Automatic PoC exploit app generation • Exploitation options • SDLC friendly • Learning • Red & Blue Team • Extensible	• CLI-only for now • SQLMap integration still in the works • Work in progress?

What is it?

- Python
- XML Parsing
- Java Parsing (PLYJ)
- Grep
- Regex
- Time
- Experience
- Googling
- Python-Adb
- Dex2jar
- Multiple rounds of decompilation
- Best effort error handling for decompilation

What does it do?

- Processes Manifest
  - Determines supported API versions and version specific vulnerabilities
  - Identifies insecure app configurations
  - Identifies all explicitly and implicitly exported inter-process communication processes (aka sources)
  - Evaluates permissions and protections
- SCA-light for Android-specific weaknesses and vulnerabilities
- Source – Sink tracking from Manifest to Class
- SDLC-friendly for use on raw source by Security or Devs
- Can be used by researchers on already published APKs, with the extraction and de-compilation occurring automatically
- Automatic generation of ADB exploit examples which are available in-app
- Automatic generation of WebView exploit files
- **Automatic generation of APK to provide POC apps**

diff QARK

- Clear/Concise reporting of issues
- Reporting includes (or will soon)
  - Severity
  - Issue explanation
  - References
  - Exploit Instructions
  - Customized exploit code / steps whenever possible
- Automatic POC APK generation
- Somewhere between Drozer and Metasploit for Android Apps

Demo Time

Your prayers are appreciated!

All hail the mighty demo gods!!

Future plans

GUI
Additional output formats
Enhanced SCA, with more source -> sink mapping
Automate APK retrieval
SQLMap for Content Provider sploiting
Hosted version?
Dealing with obfuscation?

Citations

Drozer-MWR Labs

JSSEC-Android Application Secure Design/Secure Coding Guidebook

Developer.android.com

Contact

www.linkedin.com/in/tonytrummer
@SecBro1

www.secbro.com

www.linkedin.com/in/tushardalvi
@tushardalvi