MALintent

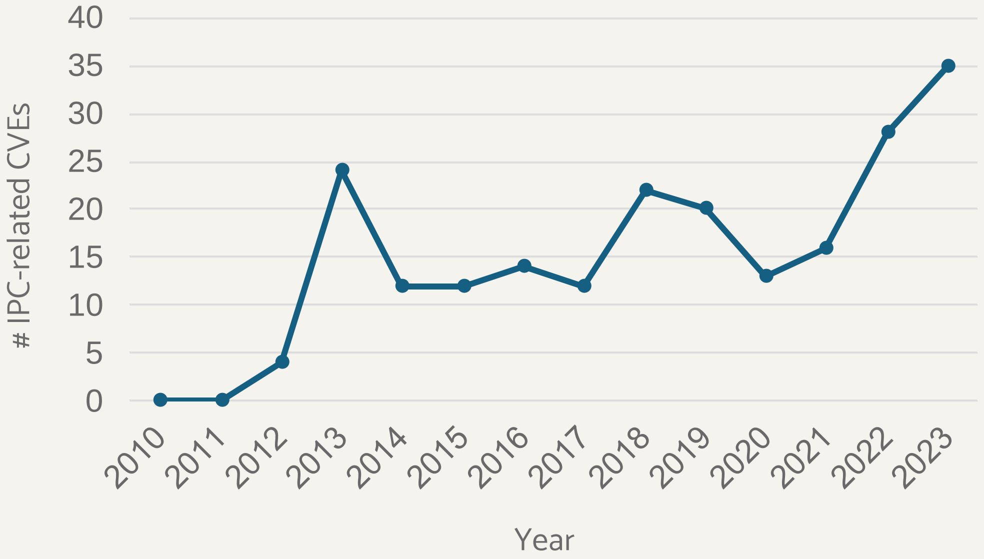
Coverage Guided Intent Fuzzing Framework for Android

Ammar Askar, Fabian Fleischer, Christopher Kruegel, Giovanni Vigna, Taesoo Kim

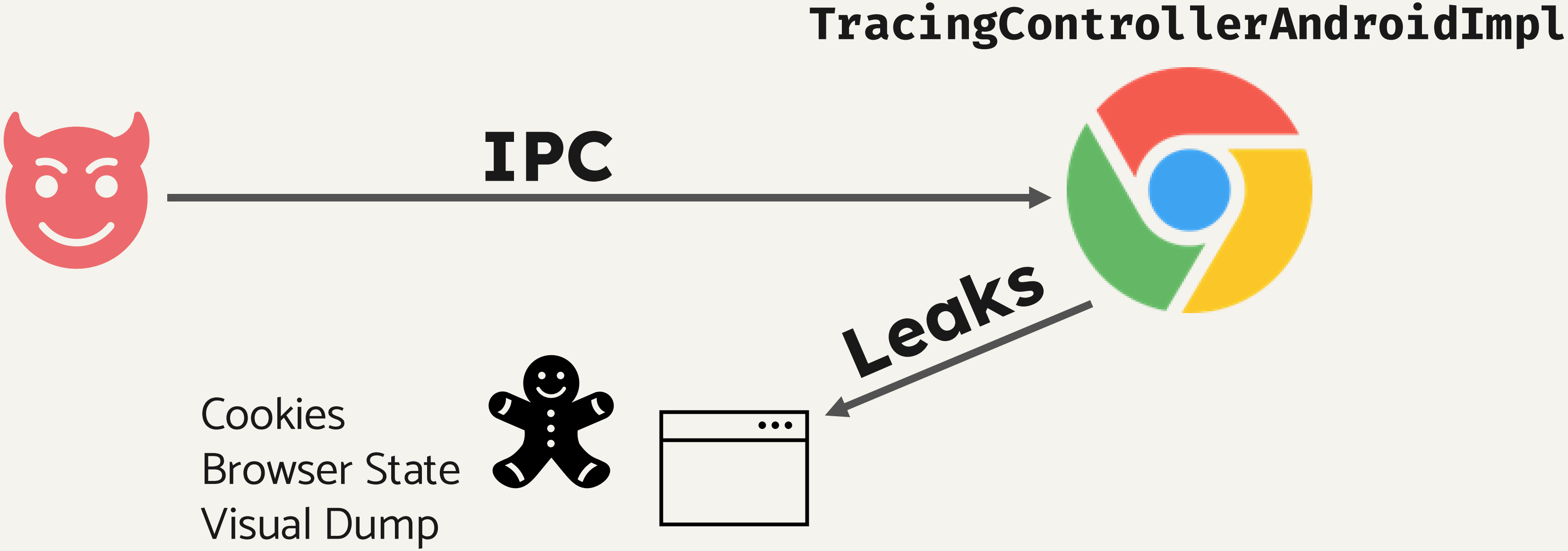
Android IPC-related **CVEs**

App isolation turns IPC into a key attack vector on Android

Increasing number of IPC-related CVEs in Android apps



Motivation



TracingControllerAndroidImpl

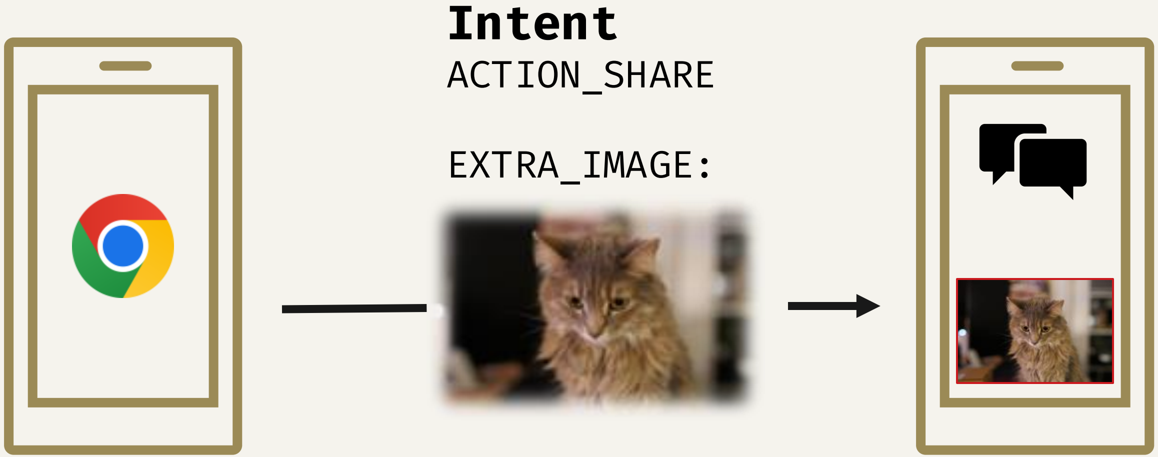
IPC

Leaks

Cookies
Browser State
Visual Dump

Android **Intents**

Primary Inter Process Communication method in Android.



Intent
ACTION_SHARE

EXTRA_IMAGE:

Android **Intents**

An object sent across app boundaries. Android apps are isolated and cannot directly access each other’s data or special permissions.

Intent

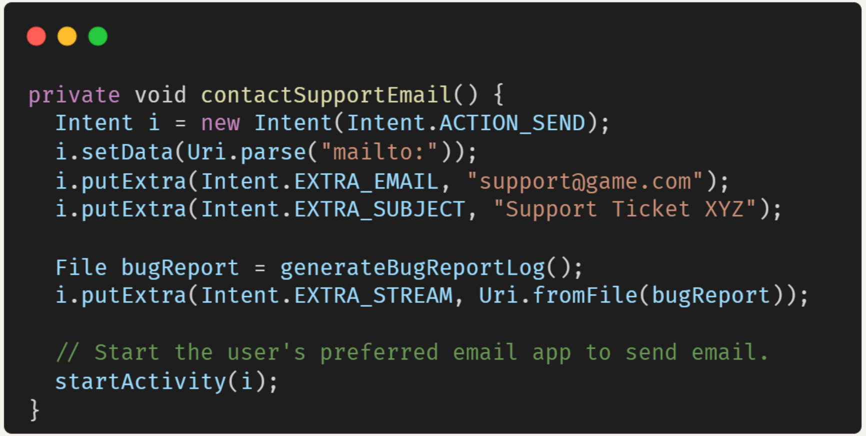
```
Actions: ACTION_VIEW (view an image)
         ACTION_DIAL (dial a number)
         ACTION_SEND (send an email)
```

Metadata

```
         EXTRA_EMAIL (email address to send to)
```

Android **Intents**

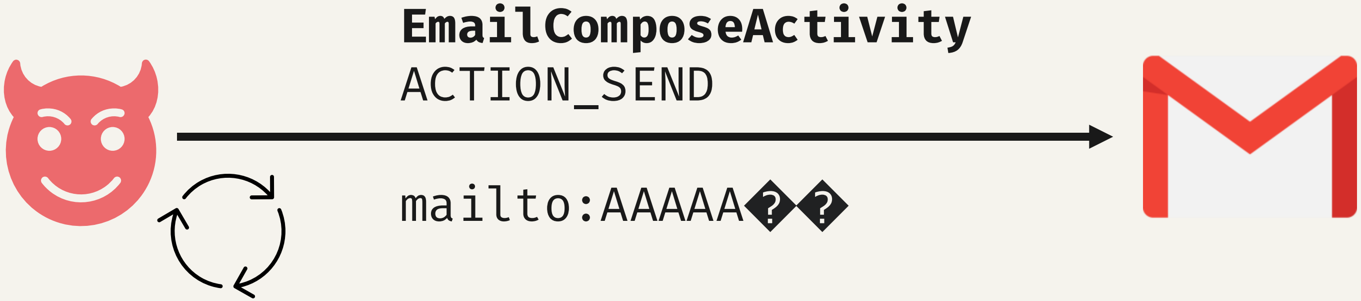
Apps can trigger intents to launch other applications.



```java
private void contactSupportEmail() {
  Intent i = new Intent(Intent.ACTION_SEND);
  i.setData(Uri.parse("mailto:"));
  i.putExtra(Intent.EXTRA_EMAIL, "support@game.com");
  i.putExtra(Intent.EXTRA_SUBJECT, "Support Ticket XYZ");

  File bugReport = generateBugReportLog();
  i.putExtra(Intent.EXTRA_STREAM, Uri.fromFile(bugReport));

  // Start the user's preferred email app to send email.
  startActivity(i);
}
```

Android **Intents**

Intents are ideal for fuzzing. They contained well-structured data and can be sent by apps without any privilege to attack other apps.

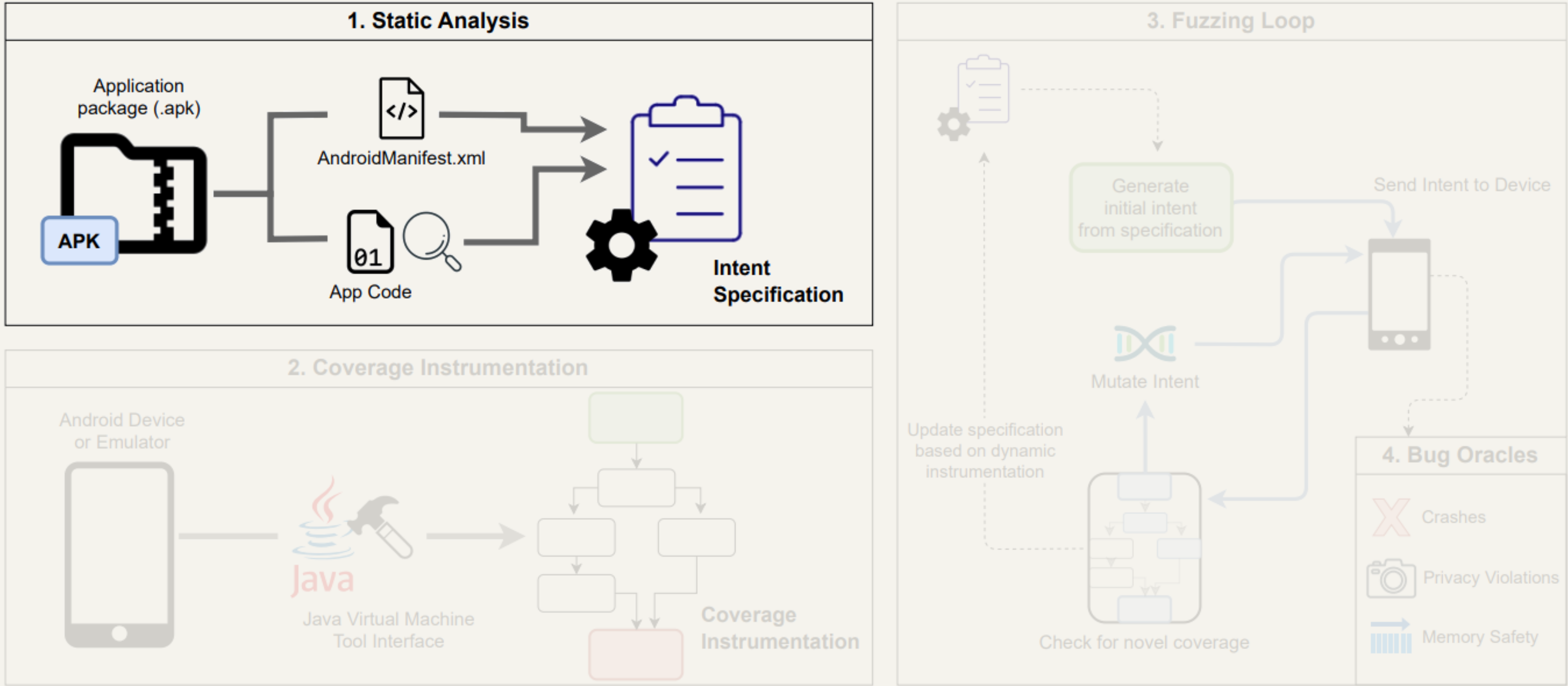


EmailComposeActivity

ACTION_SEND

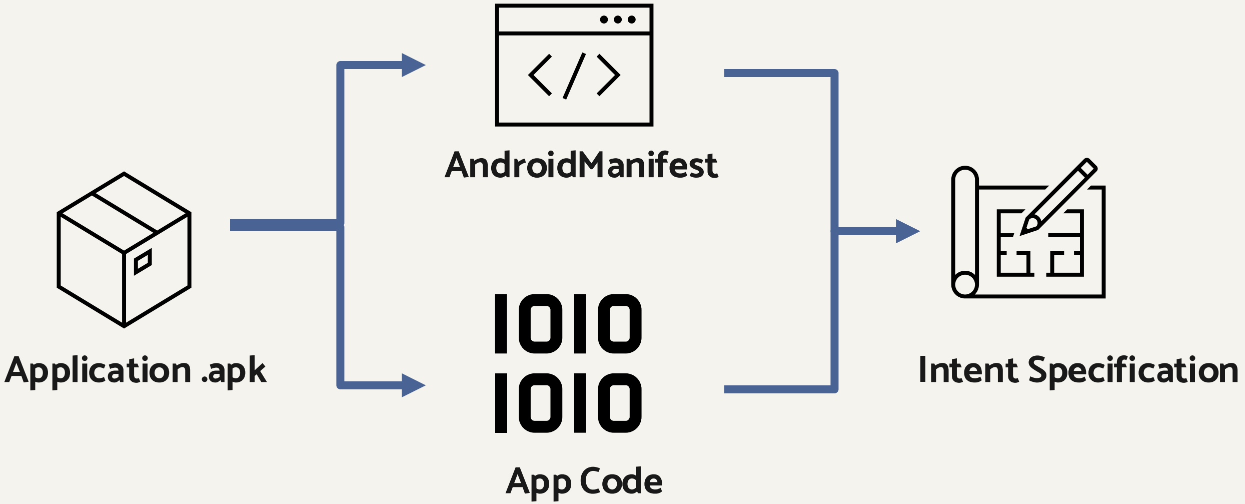
mailto:AAAAA��

Overall **Design**



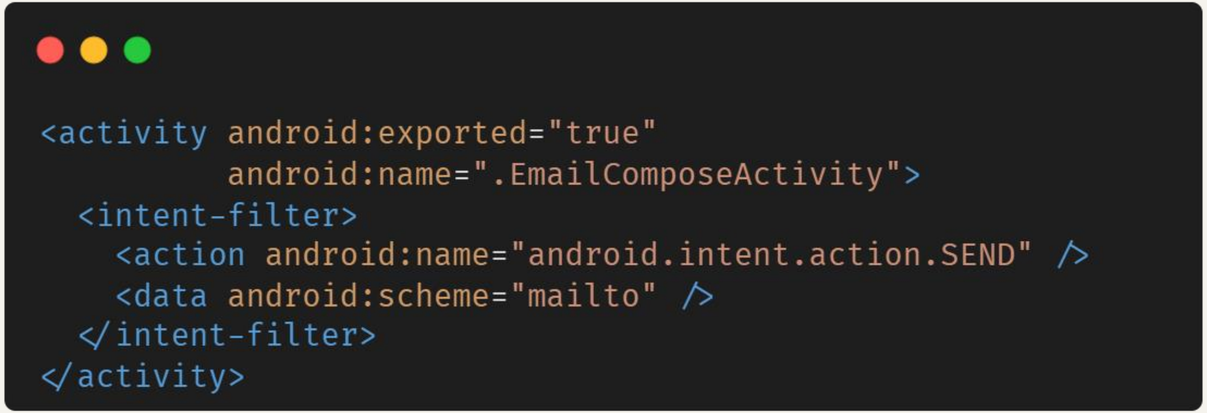
Design: **Static Analysis**

Start with the application to determine what intents can be sent to it.



Design: **Static Analysis**

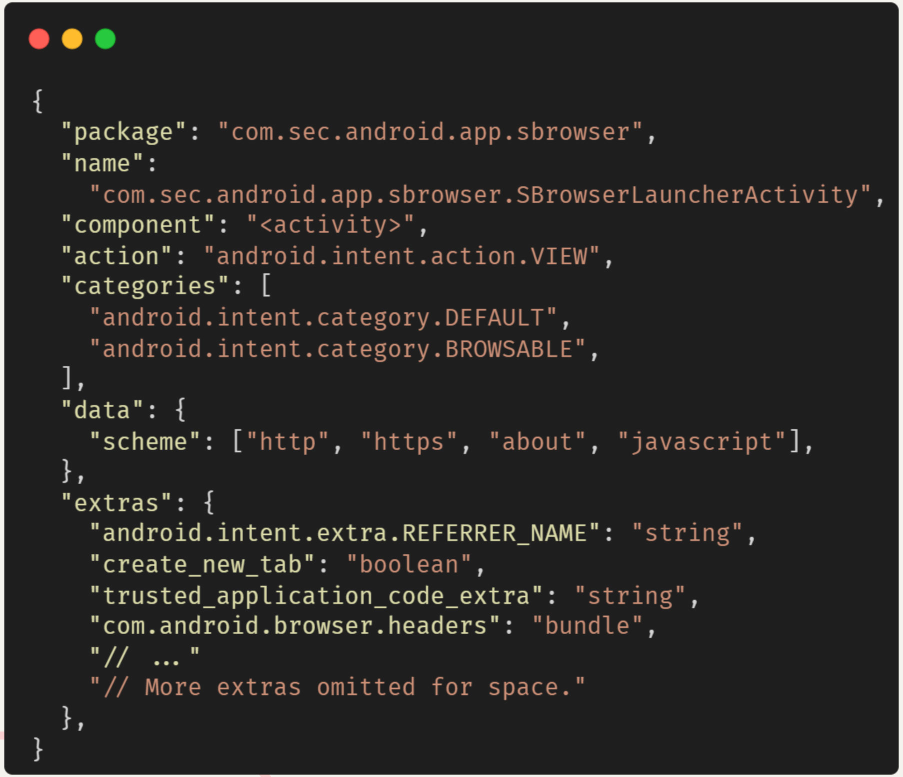
AndroidManifest



```
<activity android:exported="true"
          android:name=".EmailComposeActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND" />
    <data android:scheme="mailto" />
  </intent-filter>
</activity>
```

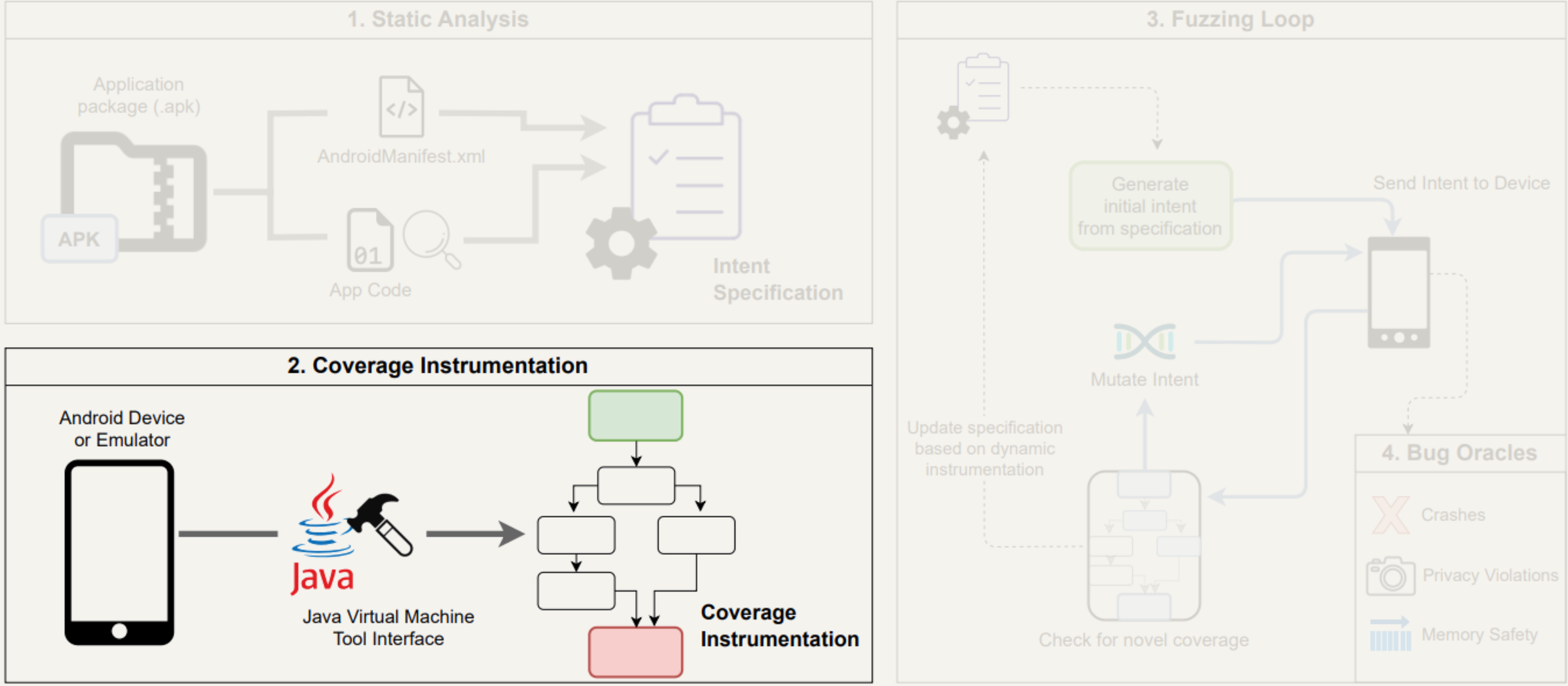
Design: **Static Analysis**

Intent Specification



```
{
  "package": "com.sec.android.app.sbrowser",
  "name":
    "com.sec.android.app.sbrowser.SBrowserLauncherActivity",
  "component": "<activity>",
  "action": "android.intent.action.VIEW",
  "categories": [
    "android.intent.category.DEFAULT",
    "android.intent.category.BROWSABLE",
  ],
  "data": {
    "scheme": ["http", "https", "about", "javascript"],
  },
  "extras": {
    "android.intent.extra.REFERRER_NAME": "string",
    "create_new_tab": "boolean",
    "trusted_application_code_extra": "string",
    "com.android.browser.headers": "bundle",
    "//  ... "
    "// More extras omitted for space."
  },
}
```

Overall Design



1. Static Analysis

Application package (.apk)

APK

AndroidManifest.xml

App Code

Intent Specification

2. Coverage Instrumentation

Android Device or Emulator

Java

Java Virtual Machine Tool Interface

Coverage Instrumentation

3. Fuzzing Loop

Generate initial intent from specification

Send Intent to Device

Mutate Intent

Update specification based on dynamic instrumentation

Check for novel coverage

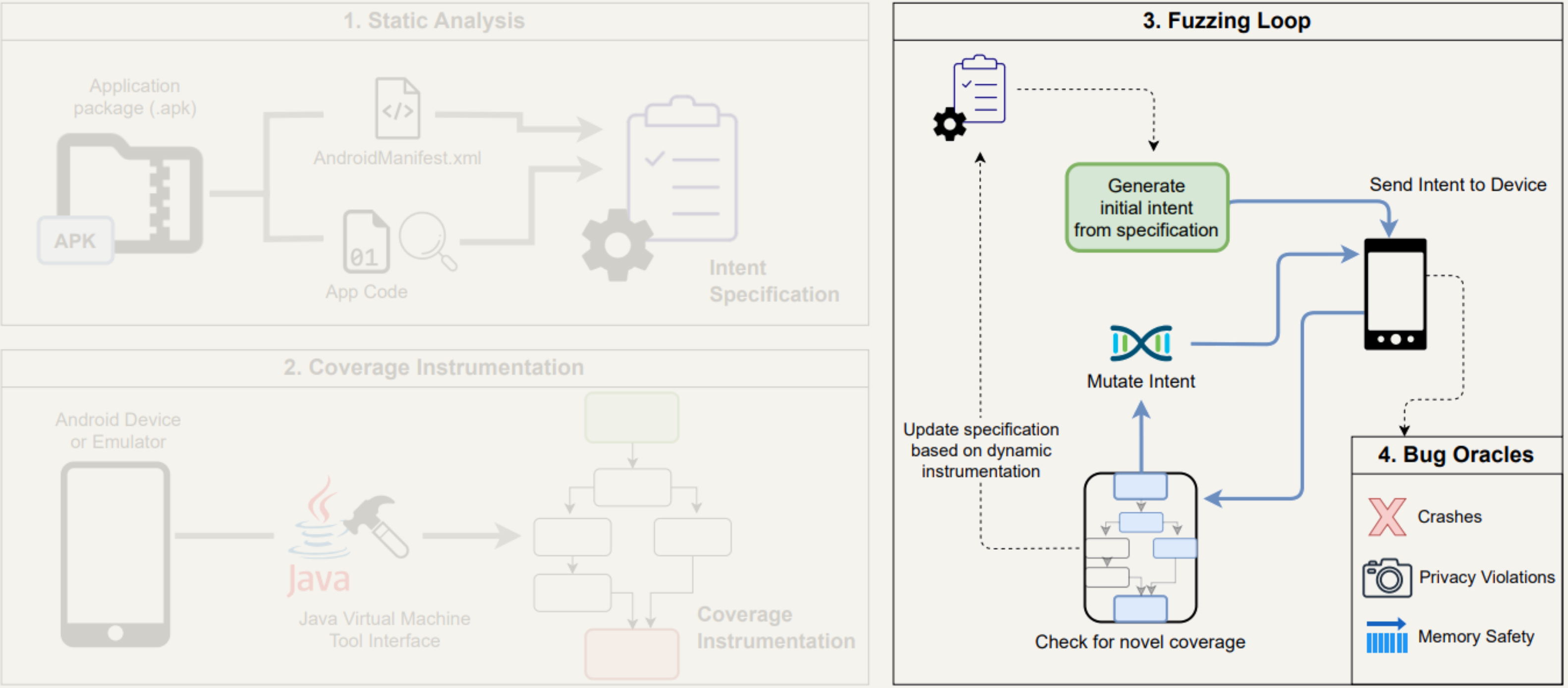
4. Bug Oracles

Crashes

Privacy Violations

Memory Safety

Overall **Design**

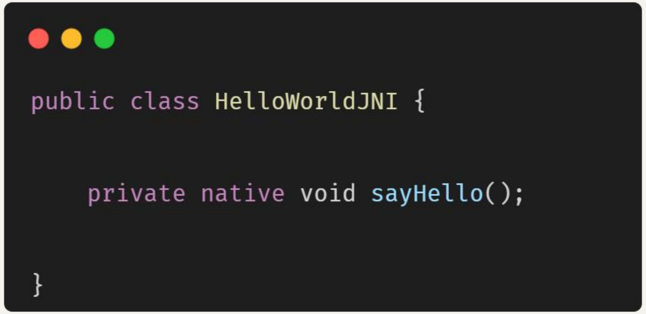


Android **Native Code**

There are a wealth of libraries in C/C++ that app developers use

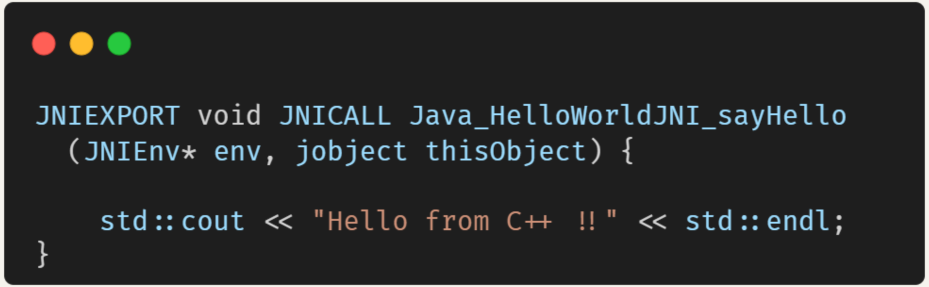
Android **Native Code**

Interfacing with these libraries done with JNI (Java Native Interface)



```
public class HelloWorldJNI {

    private native void sayHello();

}
```

Java

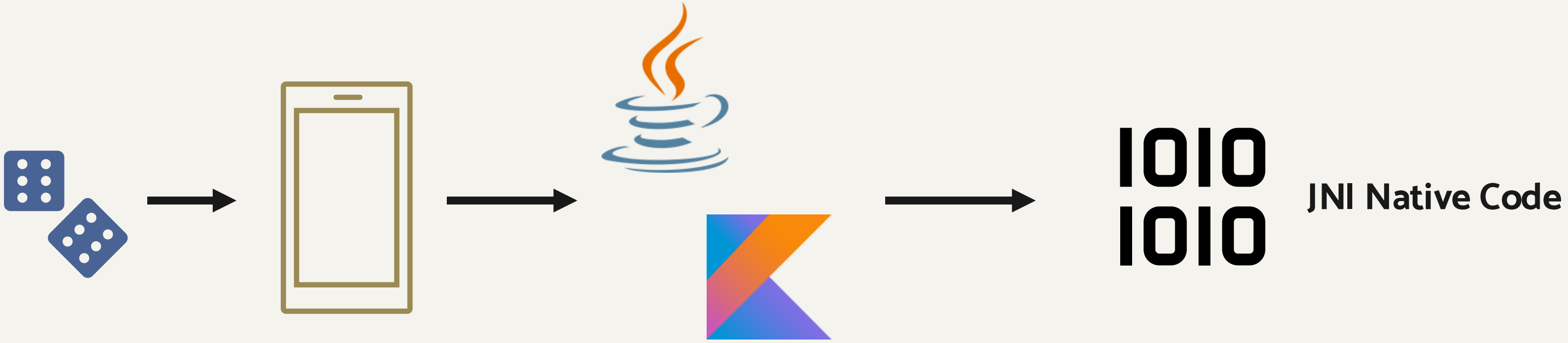


```
JNIEXPORT void JNICALL Java_HelloWorldJNI_sayHello
  (JNIEnv* env, jobject thisObject) {

    std::cout << "Hello from C++ !! " << std::endl;
}
```

C

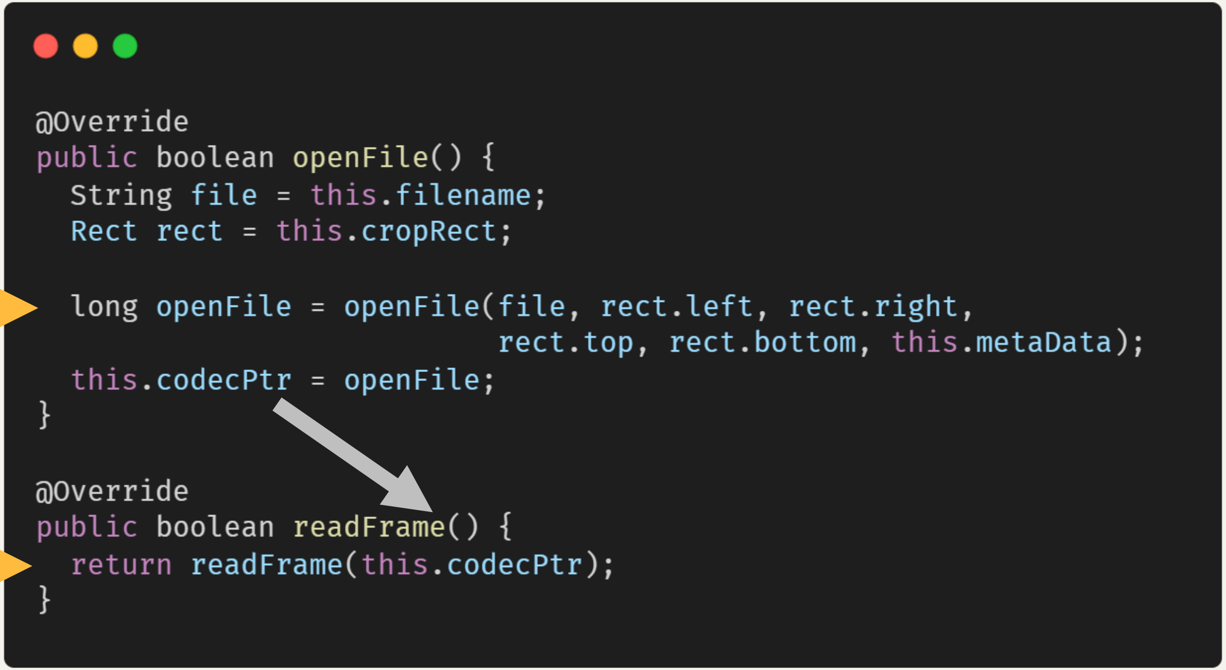
JNI Bug Finding

Basic fuzzing loop works but it is slow for JNI bugs.



JNI Bug Finding

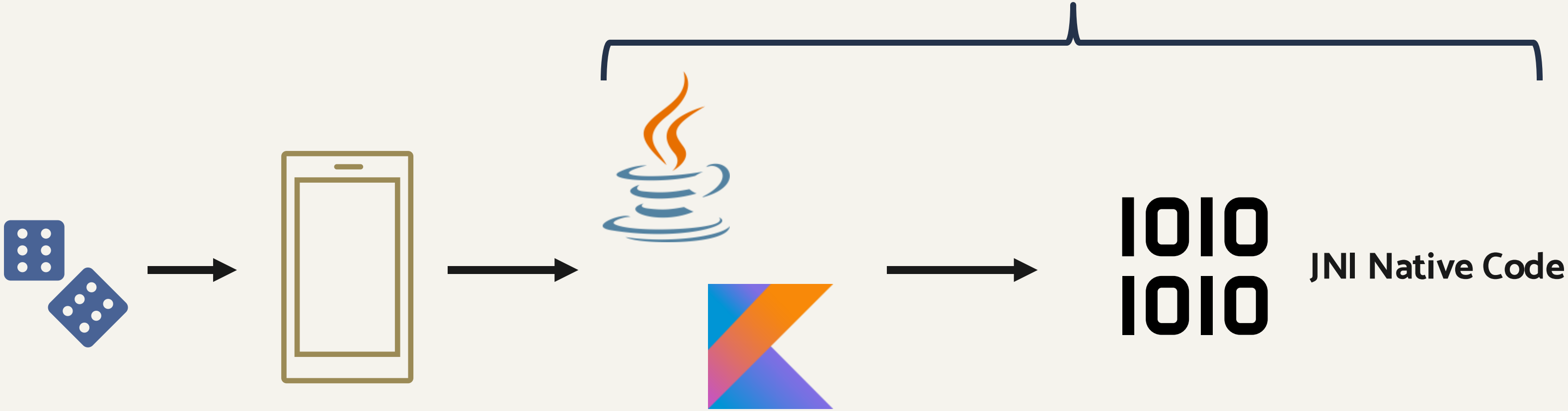
Native Calls



```
@Override
public boolean openFile() {
  String file = this.filename;
  Rect rect = this.cropRect;

  long openFile = openFile(file, rect.left, rect.right,
                           rect.top, rect.bottom, this.metaData);
  this.codecPtr = openFile;
}

@Override
public boolean readFrame() {
  return readFrame(this.codecPtr);
}
```

JNI Bug Finding

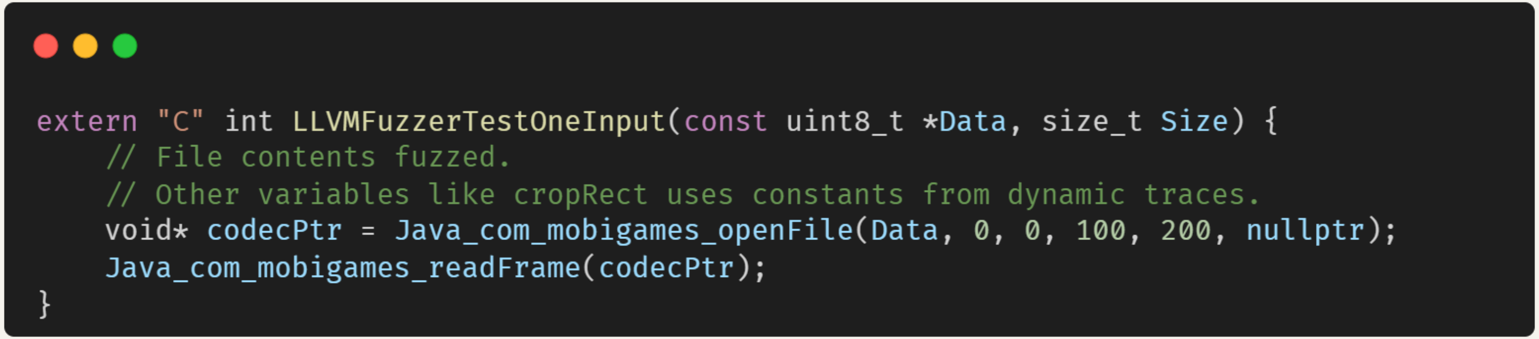
Dynamic traces from real intents invoking native code give us data-flow information



JNI Native Code

JNI Bug Finding

With the data and control flow information, we can generate a libFuzzer harness

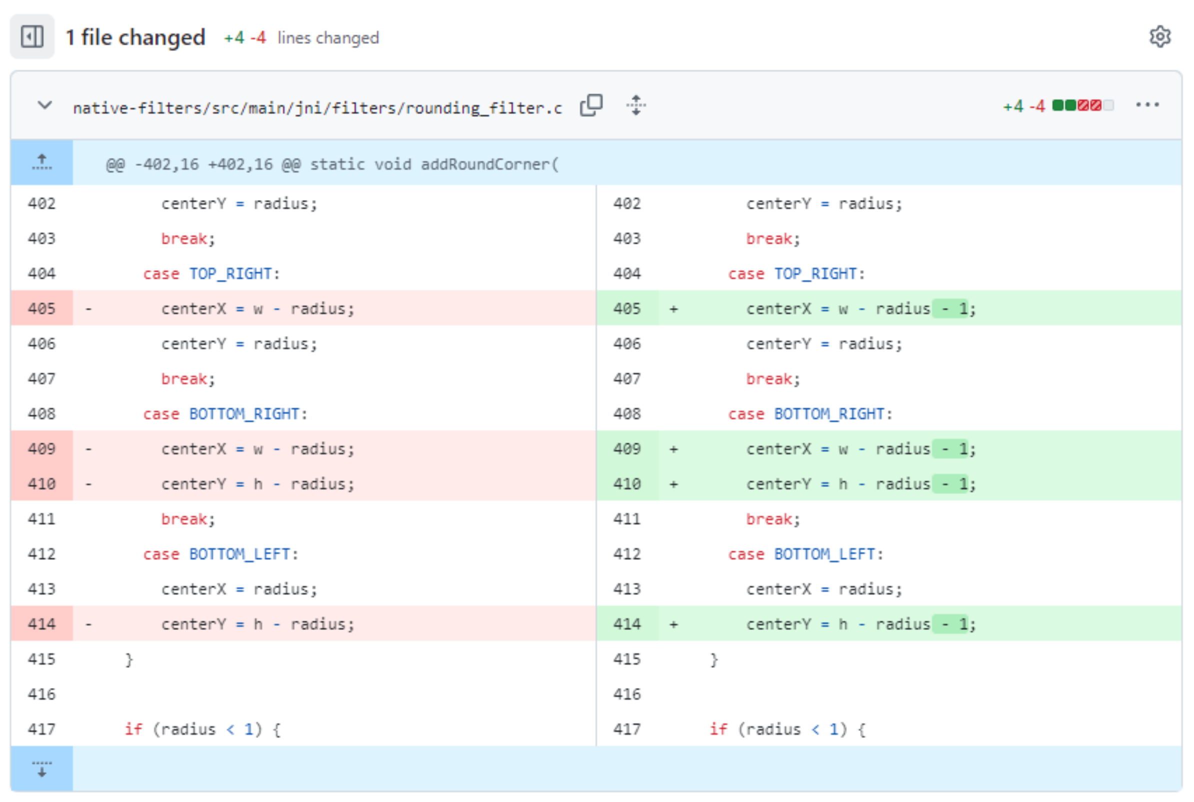


```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    // File contents fuzzed.
    // Other variables like cropRect uses constants from dynamic traces.
    void* codecPtr = Java_com_mobigames_openFile(Data, 0, 0, 100, 200, nullptr);
    Java_com_mobigames_readFrame(codecPtr);
}
```

JNI Bug Finding

Native code: Fixed rounded corners off by one calculations

Example bug found in the Facebook Fresco image library.



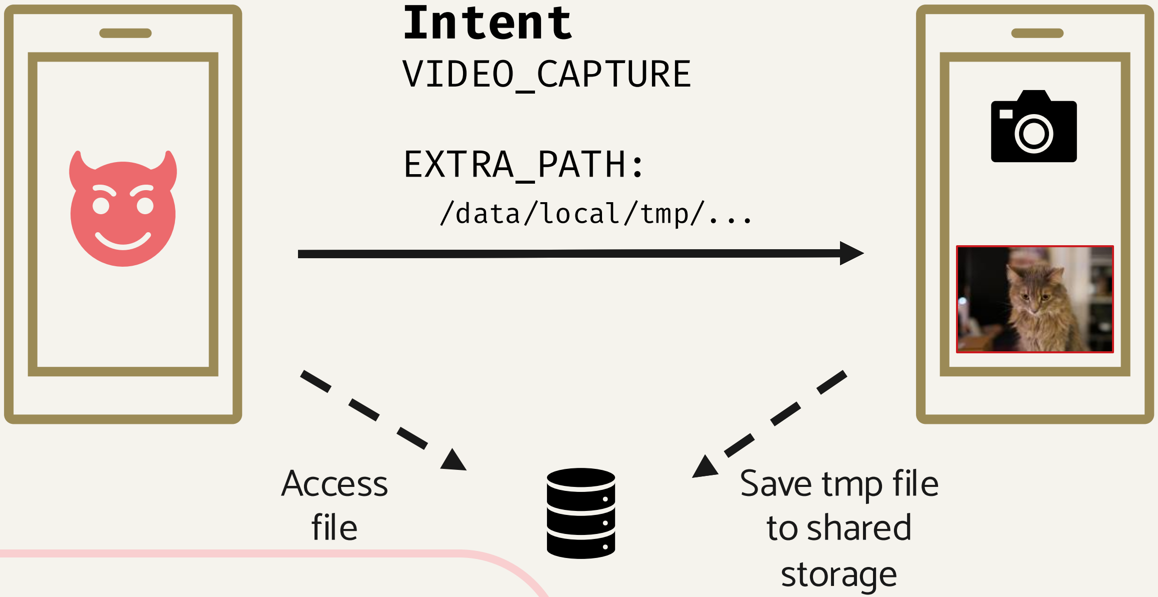
1 file changed +4 -4 lines changed

native-filters/src/main/jni/filters/rounding_filter.c +4 -4

```
@@ -402,16 +402,16 @@ static void addRoundCorner(
          centerY = radius;                          centerY = radius;
          break;                                     break;
        case TOP_RIGHT:                            case TOP_RIGHT:
-         centerX = w - radius;               +      centerX = w - radius - 1;
          centerY = radius;                          centerY = radius;
          break;                                     break;
        case BOTTOM_RIGHT:                         case BOTTOM_RIGHT:
-         centerX = w - radius;               +      centerX = w - radius - 1;
-         centerY = h - radius;               +      centerY = h - radius - 1;
          break;                                     break;
        case BOTTOM_LEFT:                          case BOTTOM_LEFT:
          centerX = radius;                          centerX = radius;
-         centerY = h - radius;               +      centerY = h - radius - 1;
      }                                          }

      if (radius < 1) {                          if (radius < 1) {
```

Privacy Violations

Improper intent handling may introduce vulnerabilities



Intent

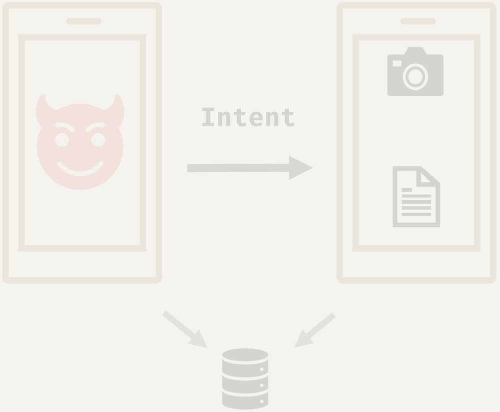
```
VIDEO_CAPTURE

EXTRA_PATH:
  /data/local/tmp/...
```

Access file

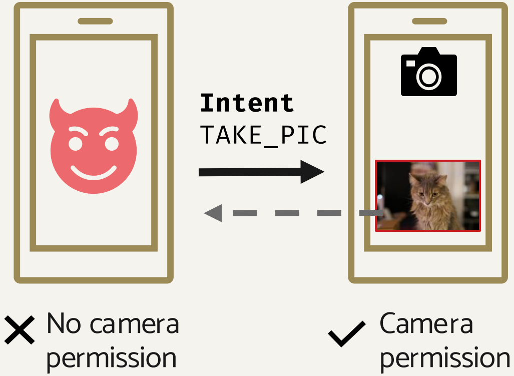
Save tmp file to shared storage

Privacy Violations: Attack Scenarios

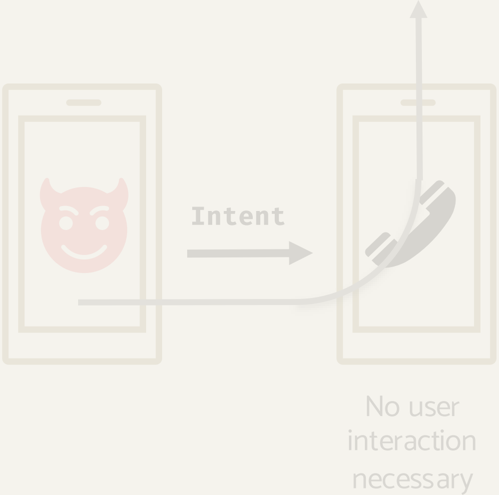
Data leak through file system



Permission escalation



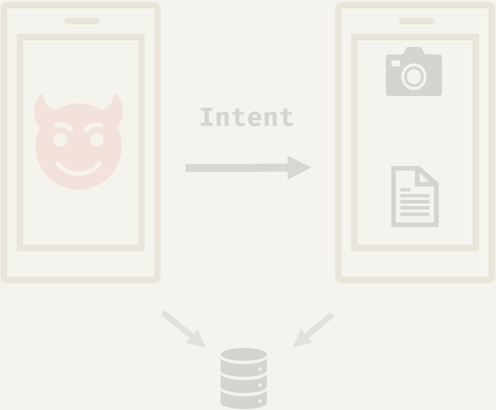
Call without user interaction



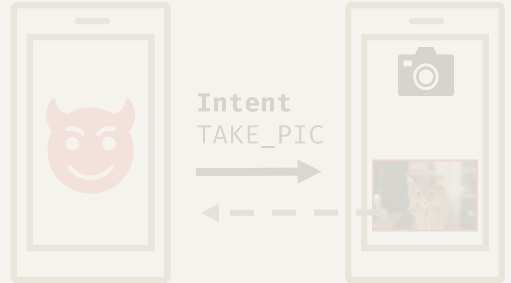
➜ Data flow analysis to detect privacy violations

Privacy Violations: Attack Scenarios

Data leak through file system



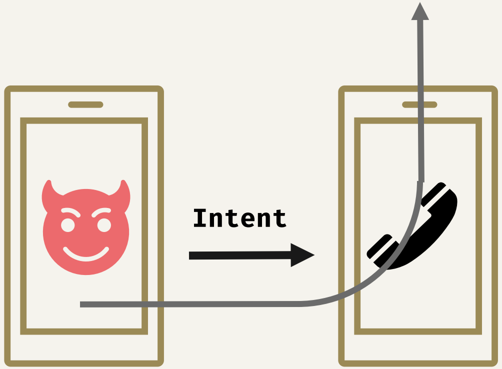
Permission escalation



✕ No camera permission

✓ Camera permission

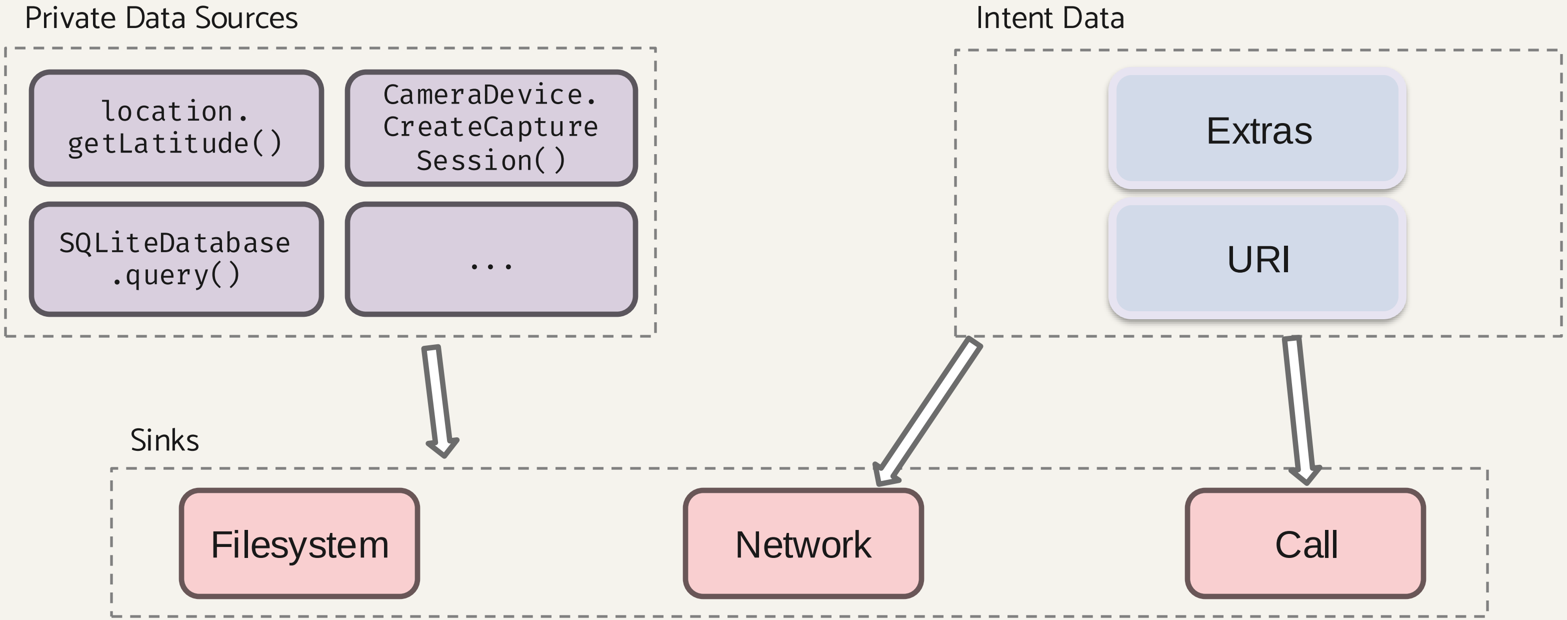
Call without user interaction



No user interaction necessary

Privacy Violations

Dynamic taint analysis to identify leaking resources



Evaluation

Ran against 500 F-Droid and Google Play Store's top-50 and top-50 productivity apps.

F-Droid + **Google Play Store**

16 hours of Fuzzing / App

Evaluation Results

9

Privacy Violations

49

Crashes

1

Memory Safety

MALintent: Coverage Guided Intent Fuzzing Framework for Android

- Framework for fuzzing Intent handlers in Android apps
- Includes oracles for bug detection
  → Privacy violations, memory safety, crashes
- Found 49 crashes, 9 privacy violations, and 1 memory safety bug
- Open-source implementation available

Ammar Askar, Fabian Fleischer, Christopher Kruegel, Giovanni Vigna, Taesoo Kim
Network and Distributed System Security (NDSS) Symposium 2025. San Diego, CA.

aaskar@gatech.edu, fleischer@gatech.edu

Paper

Georgia Tech

UC SANTA BARBARA