FUSE: Inter-Application Security for Android

High Confidence Software & Systems (HCSS 2012)

Project Lead: Joe Hurd <joe@galois.com>

Galois Team: Aaron Tomb, David Archer, Jonathan Daugherty

The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Approved for Public Release, Distribution Unlimited.

galois

Talk Plan

© 2012 Galois, Inc. *All Rights Reserved.*

| galois |

Android & Security

- There is a powerful need for mobile devices that allow users to run a diverse array of apps while keeping confidential information secure.
- In recent times the Android mobile platform has rapidly grown in popularity, but there have been many security problems.
    - e.g., jail-breaking, permission escalation, trojan apps.
- *"Mobile is the new platform. Mobile is a very intimate platform. It's where the attackers are going to go."* [Schneier]

© 2012 Galois, Inc. *All Rights Reserved.*

Security Evaluation in the App Life-Cycle

The security of Android apps may be evaluated:

- by the developer (during coding)
  - ...but more than 50% of apps include 3rd-party libraries, some of which download and run code from remote servers [NCSU]
- by the marketplace owner (at release time)
  - ...but traditional app evaluation can't keep up without automatic tool support
- by the user (at installation time)
  - ...but *"given a choice between dancing pigs and security, users will pick dancing pigs every time"* [Felten & McGraw]
- by anti-virus software on the device (at run-time)
  - ...but by then it's too late

And all of these evaluations are restricted to individual apps.

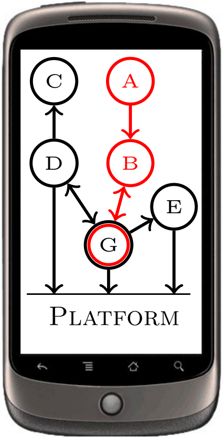
© 2012 Galois, Inc. *All Rights Reserved.*

Android Security Layers

The security of an Android device processing confidential data breaks down into three categories:

1. **Platform:** Apps cannot bypass the platform security mechanisms.
2. **App:** Apps contain no exploitable security vulnerabilities (e.g., by scanning their source code using static analysis).
3. **Inter-app:** App communications satisfy the security policy (e.g., all information flows from red apps to black apps are mediated by the guard app *G*).

This talk is focused on inter-app security.



© 2012 Galois, Inc. *All Rights Reserved.*

galois

Built-In Android Inter-App Security

- The problem is that the Android security model based on permissions does not provide sufficient protection against inter-app collusions.
- **Example:** We demonstrated this by implementing a simple pair of apps:
  1. App A requires permission to read your contact information and also requires permission P.
  2. App B declares permission P and uses it to protect an inter-app capability to publish information to the internet.
- Apps A & B are individually secure, but collectively insecure.
- **Also:** A user installing apps A & B in this order will not even be told of the existence of permission P.

© 2012 Galois, Inc. *All Rights Reserved.*

FUSE Project Vision

- The FUSE project is an effort funded by the DARPA TransApps program to defend against data exfiltration by multiple colluding apps.
- Galois is developing the FUSE tool to carry out an inter-app security analysis and reveal app collusions on the marketplace.
- The marketplace contains every app available, and app collusions can be discovered even if the vulnerable collection has never been installed on a device.
- Dedicated marketplace servers can perform the analysis, rather than repeating work on limited-power mobile devices.

© 2012 Galois, Inc. *All Rights Reserved.*

Adding an App to the Marketplace

- When adding an app to the marketplace, we carry out the following analysis to compute its inter-app signature:
  1. Extract information from the app package manifest.
  2. Supplement this by using automatic static analysis techniques to carry out a white-box analysis of the app code.
  3. Derive the possible information flows from app sources to sinks (supported by control flows from app entry points).
- Note that the inter-app signature analysis is compositional.
  - i.e., it only analyzes one app at a time.
  - Compositional analyses have better scalability properties.
- Add the inter-app signature into a marketplace database.

© 2012 Galois, Inc. *All Rights Reserved.*

Inter-App Security Analysis

- **Example Use-Case 1:** Data-mining the inter-app signatures.
  - Constantly scan the marketplace database for insecure information flows supported by colluding apps.
  - **Success Metric:** Discover a small set of colluding apps in a large collection of benign apps.
- **Example Use-Case 2:** Device provisioning check.
  - Perform a deeper inter-app security analysis on the set of apps selected for installation on a device.
  - **Success Metric:** Detect subtle inter-app collusions within a small set of apps.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

Feasibility Study

To assess the feasibility of the FUSE project vision, we carried out a study to answer the following two questions:

1. Is the Android security model and packaging of apps amenable to an inter-app security analysis?
2. If so, can the analysis scale up to an entire marketplace?

In this talk we present the results of this feasibility study.

© 2012 Galois, Inc. *All Rights Reserved.*

Background: Android Inter-App Communication

- Android apps are made of components.
  - **Activities** provide a user interface to an app.
  - **Services** perform an action in the background.
  - **Broadcast receivers** listen for messages from other apps.
  - **Content providers** store potentially-shared data.
- Components communicate using intents, composed of:
  - an optional action (e.g., `EDIT`),
  - an optional target component (e.g., a specific editor),
  - and some optional meta-data (e.g., a file name).

© 2012 Galois, Inc. *All Rights Reserved.*

galois

Background: Android Security Model

- App components are annotated with intent filters that describe what intents they can respond to.
- The Android security model allows apps to protect critical components by specifying a permission that calling apps are required to hold.
- The app components, permissions and intent filters are specified in the package manifest, which the user must approve at installation time.

© 2012 Galois, Inc. *All Rights Reserved.*

Feasibility Study: Analyzing App Packages

- Most of the relevant information for an inter-app security analysis is readily available in the package manifest.
  - Components, permissions and intent filters are present.
  - Intent calls are missing.
- The FUSE project vision relies on a capability to automatically extract security-relevant information directly from app packages.
- The absence of intent calls from package manifests offered an opportunity to test the feasibility of this.
- **Feasibility Test:** Is it possible to automatically extract intent call information from an app package?

© 2012 Galois, Inc. *All Rights Reserved.*

Inferring Inter-App Communication

- All inter-app communication occurs in three steps:
  1. Create an intent object.
  2. Set the action, component or meta-data fields of the intent.
  3. Call one of a small set of app communication methods (`startActivity`, `startService`, etc.)
- We can identify all occurrences of these steps by inspecting the bytecode in the app package.
  - No need for the app source code.
  - No need to trust the compiler.
- Standard static analysis techniques can identify object creation, field update and method calls.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

The FUSE App Analysis Tool

- To support the feasibility study we developed the FUSE tool to compute conservative over-approximations of app intent calls.
- A conservative over-approximation is appropriate for an inter-app security analysis.
  - **No False Negatives:** If an intent call is possible, the FUSE tool will identify it.
  - **Some False Positives:** The FUSE tool may identify intent calls that will never be executed.
- **Note:** Computing the precise set of possible intent calls is an undecidable problem.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

External Tools

1. The FUSE tool uses the open source `dex2jar` tool to convert the Dalvik bytecode in the app package to equivalent Java bytecode.
   - **Pro:** This allows us to reuse Java infrastructure (e.g., the bytecode parser).
   - **Con:** `dex2jar` sometimes generates semantically ill-formed Java bytecode (we have filed a bug report).
2. The FUSE tool also uses the open source `apktool` to extract the manifest from the app package.

© 2012 Galois, Inc. *All Rights Reserved.*

Core Static Analysis

The core of the FUSE tool is a static analysis of Java bytecode that operates as follows:

1. Extract information from the bytecode.
    - Identify instructions that create new intent objects.
    - Identify instructions that set intent action or component fields.
    - Identify app communication method calls (e.g., `startService`).
2. For each method of an app component:
    - If it contains one instruction to create an intent object and one app communication method call then one intent call is generated for the component.
    - If it contains multiple create instructions or communication calls then we generate intent calls for all possible combinations: imprecise but conservative.
    - The precision can be improved with well-known techniques.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

FUSE Tool Use

- The FUSE tool computes the intent calls from an app package as follows:
  1. The `dex2jar` tool converts the Dalvik bytecode in the app package to a Java JAR file.
  2. The `apktool` extracts the manifest from the app package.
  3. The core static analysis extracts the intent calls from the Java bytecode in the JAR file.
  4. The intent calls are added to the package manifest, and the result is output in extended package manifest format.
- The extended package manifest format is an extension of the XML standard format for Android package manifests which includes the possible intent calls for each app component.

© 2012 Galois, Inc. *All Rights Reserved.*

Extended Package Manifests

- The `intent-filter` tag already exists in manifest files, describing the form of intents a component can receive.
- The `intent-call` tag is added by the FUSE tool, describing the form of intents a component can issue.

Extended Package Manifest (Excerpt from a password safe)

```
<service android:name=".service.ServiceDispatchImpl">
<intent-call>
<action android:name="org.openintents.action.CRYPTO_LOGGED_OUT" />
</intent-call>
<intent-filter>
<action android:name=".safe.service.ServiceDispatchImpl" />
</intent-filter>
</service>
```

- The Android security model could be extended to enforce intent calls in package manifests as it already does for intent filters, making inter-app communication more explicit.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

Feasibility Study: Analyzing App Packages

- The static app analysis performed by the FUSE tool:
  - works quickly on existing apps; and
  - can be easily integrated with other tools.
- **Feasibility Assessment:** Android app packages can be analyzed using well-understood static analysis techniques.

© 2012 Galois, Inc. *All Rights Reserved.*

Feasibility Study: Scaling to the Marketplace

- To test the scalability of the inter-app security analysis we assembled a benchmark of 104 apps:
  - 42 apps from the Android SDK R8 (Android 2.2);
  - plus 62 open source apps hosted on code.google.com.
- The 104 apps consisted of:
  - 920 components;
  - 861 intent filters;
  - plus 357 intent calls added by the FUSE tool.
- **Feasibility Test:** Is it possible to analyze the inter-app calls in this benchmark set of apps?

© 2012 Galois, Inc. *All Rights Reserved.*

Inter-App Control Flow Implemented in SQL

- To support the feasibility study, we developed an inter-app control flow analysis as a sequence of SQL statements:
  1. **Initialization:** For each app, insert the information from the extended package manifests.
  2. **Inter-App Component Calls:** Create a database table relating app components with matching intent calls and intent filters (and also respecting permissions).
  3. **Inter-App Calls:** Create a database table projecting the inter-app component calls to the owning app, relating apps that may call each other.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

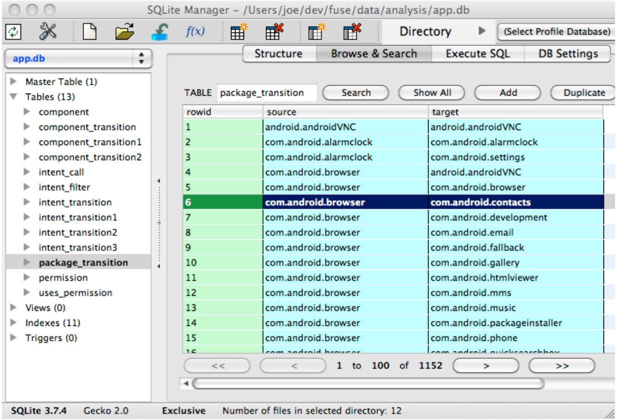
Feasibility Study: Scaling Up To An Apps Marketplace

- We used the simple database engine SQLite (version 3.6.12) to compute the inter-app control flow on our benchmark set of 104 apps.
- The inter-app component call table resulted in 3,290 possible intent calls between app components.
- The inter-app call table resulted in 1,152 possible intent calls between apps.
- On a 2.53GHz MacBook Pro with 8Gb of RAM the experiment completed within 10 seconds.
- **Feasibility Assessment:** Existing database technology gives promising results for scaling up the inter-app security analysis.

© 2012 Galois, Inc. *All Rights Reserved.*

Viewing Inter-App Control Flow

The SQLite database containing all the inter-app control-flow data:



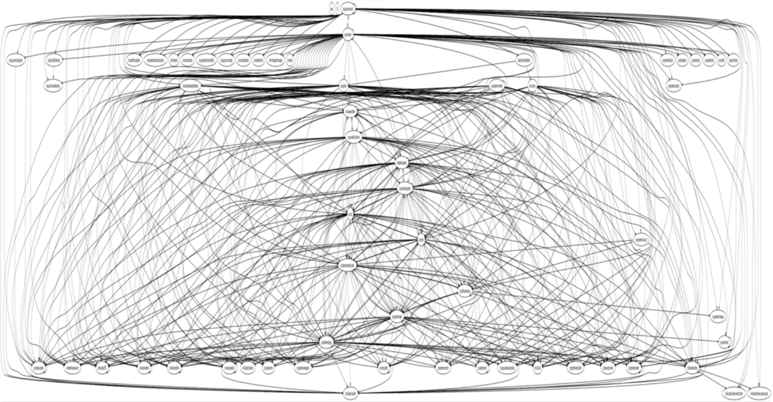
The highlighted row shows a possible intent call from the Android browser to the contacts app.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

Visualizing Inter-App Control Flow

This is a directed graph view of the calls between the 62 open source apps from `code.google.com`:

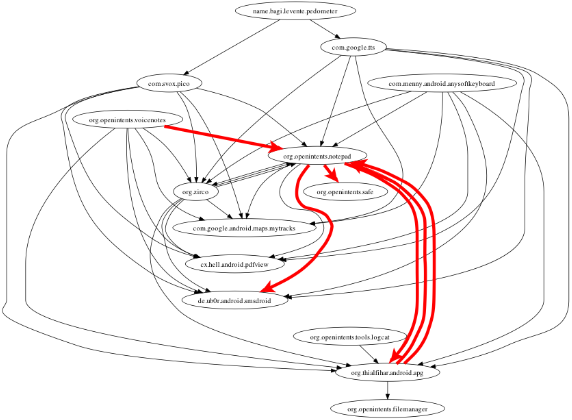


© 2012 Galois, Inc. *All Rights Reserved.*

galois

Zooming in on Inter-App Control Flow

- Potential insecurities can be observed in the inter-app call graph.
- **Example:** A notepad app has access to both the password safe and an SMS app:



© 2012 Galois, Inc. *All Rights Reserved.*

galois

Feasibility Study Results

- We carried out a study to test the feasibility of the following approach to inter-app security analysis:
  1. Use static program analysis on individual apps to extract inter-app signatures.
  2. Query combinations of these signatures to reveal insecure inter-app behavior.
- We demonstrated the feasibility of this method by implementing an inter-app control-flow analysis on a benchmark set of 100 apps:
  1. The FUSE static analysis tool extracted the possible intent calls from individual apps.
  2. The intent calls and package manifest data were used to populate a database, and SQL queries extracted possible inter-app calls exercising dangerous permissions.

© 2012 Galois, Inc. *All Rights Reserved.*

galois

Future Work

- We are currently extending the FUSE tool to perform inter-app information flow and value analysis.
  - Information flow analysis reveals threats to confidentiality and integrity on mobile device data.
  - Value analysis allows us to precisely define possible app behavior (e.g., narrowing down the possible target URLs when exercising an `INTERNET` permission).
- We are also developing a sample set of security policies that constrain inter-app communication.
  - The policy rules will be automatically compiled to queries over the marketplace database of inter-app signatures.
  - It is important to separate the policy from the checking tool to ease maintainence and support deployment in new domains.

© 2012 Galois, Inc. *All Rights Reserved.*

Questions?

`http://www.galois.com/`

`joe@galois.com`

© 2012 Galois, Inc. *All Rights Reserved.*

galois