Preventing Accidental Data Disclosure in Modern Operating Systems

CCS `13
Adwait Nadkarni

- Ph.D student at North Carolina State University
- Research interests: Information flow control, smartphone OS security, and smartphone application security.
William Enck

- An Assistant Professor in the Department of Computer Science at NC State University.

- Research focuses on the design, optimization, and measurement of security for operating systems, specifically on mobile phones, and the complex environments in which they operate.
Background

- Modern OSes run each application as a unique security principal.
- Navigates through a series of screens.
- OSes provide modular apps.
Example—Problem

• Data intermediary problem:
  • share sensitive data with other app
  • result from user choices in selecting apps
  • Most apparent in modern OSes
Example—Signing a Doc

Figure 1: Document signing use case with four apps. A confidential contract received via Email is 1) read in a viewer, 2) converted to PDF, 3) embedded with a written signature, and 4) Emailed back to the sender.
Example—Problem

• Accidental data disclosure:
  • share data with wrong app
  • poorly programmed app
• No malicious data disclosure
Example—Signing a Doc

Figure 1: Document signing use case with four apps. A confidential contract received via Email is 1) read in a viewer, 2) converted to PDF, 3) embedded with a written signature, and 4) Emailed back to the sender.
Aquifer
Aquifer

• designed to protect large, application-specific, user data objects

• developer specify secrecy restrictions

• all apps participating in a user interface workflow
Android background

- Four component type: activity, service, content provider, and broadcast receiver
- Each UI is defined by a activity
- Other components run in background
- binder framework provides process control and IPC
User Interface workflow

Figure 2: Aquifer policy abstraction
Architecture

Figure 3: Aquifer architecture for Android
Principle

- Decentralized policy specification
- Developers & Users define policy
- Compatibility with legacy applications (default policy)
- Minimizing policy violations (filter)
- Compatibility with background functionality (file description)
Aquifer Policy

- Policy types:
  - Export Restrictions
  - Required Restrictions
  - Filters
Aquifer Policy

Definition 1 (Export list). An export list $E$ is a set of applications that may access the network while participating in the UI workflow.

Definition 2 (Required list). A required list $R$ is a set of applications that all must have been present on the UI workflow at sometime in the past for any application on the UI workflow to access the network.

Definition 3 (Workflow filter). A workflow filter $F$ is a set of tuples $\{(s_1, T_1), \ldots, (s_n, T_n)\}$, each containing an action string $s_i$ and a set of targets $T_i$. If the normal resolution of an intent message sent to action string $s_i$ is a set of apps $N$, then the resulting allowed target applications is $N \cap T_i$.

Definition 4 (Workflow label). A workflow label $L$ is an expression $L = \{O_1 : (E_1, R_1, F_1); \ldots; O_n : (E_n, R_n, F_n)\}$, where $O_i$ is an owner (application) and $E_i$, $R_i$, and $F_i$ are an export list, required list, and workflow filter, respectively, specified by $O_i$.

Definition 5 (Effective export list). For a workflow label $L$, the effective export list $E_e = \bigcap \text{exports}(L, O), \forall O \in \text{owners}(L)$.

Definition 6 (Effective required list). For a workflow label $L$, the effective required list $R_e = \bigcup \text{requires}(L, O), \forall O \in \text{owners}(L)$.

Definition 7 (Effective workflow filter). For a workflow label $L$, the effective workflow filter $F_e$ is the set of tuples containing action string and corresponding target application set created by taking the union of all action strings and the intersection of the targets for those action strings. More precisely, $F_e = \{(s_i, T_i) \mid s_i \in \bigcup \text{actions}(F) \text{ and } T_i = \bigcap \text{targets}(F, s_i), \forall F \in \text{filters}(L, O), \forall O \in \text{owners}(L)\}$.

Definition 8 (Label join $\sqcup$). For workflow labels $L_1$ and $L_2$, the join $L = L_1 \sqcup L_2$ is a new label ensuring the following for all owners $O$:

$$\text{owners}(L) = \text{owners}(L_1) \cup \text{owners}(L_2)$$

$$\text{exports}(L, O) = \text{exports}(L_1, O) \cap \text{exports}(L_2, O)$$

$$\text{requires}(L, O) = \text{requires}(L_1, O) \cup \text{requires}(L_2, O)$$

$$\text{filters}(L, O) = \{(s_i, T_i) \mid s_i \in \text{actions}(F_1) \cup \text{actions}(F_2), T_i = \text{targets}(s_i, F_1) \cap \text{targets}(s_i, F_2),$$

where $F_1 = \text{filters}(L_1, O)$,

$$F_2 = \text{filters}(L_2, O)$$
Aquifer System

- Identifying the Workflow:
  - a list of applications the workflow has visited
  - a list of metadata for currently “running” UI screens

- Policy Administration

- Removing Unrelated Policy
Aquifer System

- Controlling the network access of the UI screen
- Enable network if and only if:

\[(E_e = \emptyset \lor app(p) \in E_e) \land (\forall r \in R_e, r \in W_V)\]
Background Functionality

• Accessing a Daemon:
  • Fine-grained tracking (taintDroid and CleanOS)
  • leveraging Linux’s file_permission LSM hook

• Accession a File:
  • save label with file
  • the hook notifies the Aquifer Service
Evaluation

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Number of Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data sources</td>
<td>85 (17%)</td>
</tr>
<tr>
<td>Data intermediaries</td>
<td>140 (28%)</td>
</tr>
<tr>
<td>Value from Export Policy</td>
<td>70 (14%)</td>
</tr>
<tr>
<td>Value from Regulate Policy</td>
<td>78 (15.6%)</td>
</tr>
</tbody>
</table>

based on 500 apps from Google Play Store
## Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Android</th>
<th>Aquifer</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>App load</td>
<td>188.49±5.36 ms</td>
<td>192.07±6.30 ms</td>
<td>1.9%</td>
</tr>
<tr>
<td>App filter</td>
<td>194.12±7.91 ms</td>
<td>195.22±7.52 ms</td>
<td>0.55%</td>
</tr>
<tr>
<td>Net access</td>
<td>108.60±6.48 ms</td>
<td>109.64±6.31 ms</td>
<td>0.53%</td>
</tr>
<tr>
<td>Policy change</td>
<td>-</td>
<td>1.98±1.27 ms</td>
<td>-</td>
</tr>
<tr>
<td>File Read (1MB)</td>
<td>4.76±0.09 ms</td>
<td>5.23±0.22 ms</td>
<td>9.87%</td>
</tr>
<tr>
<td>File Write (1MB)</td>
<td>23.89±0.45 ms</td>
<td>25.44±0.86 ms</td>
<td>6.49%</td>
</tr>
</tbody>
</table>
Case-Study

• K-9 Mail and PDFView (sending to a server; saving file and sending it later)

• modified K-9 to be Aquifer-aware. Policy:

\[
E = \{ \text{K9EMail} \} \\
R = \{} \\
F = \{ (\text{ACTION\_SEND}, \{ \text{K9EMail} \}) \}
\]

• re-performed and PDFView can not access to the network
Discussion

- lead to usability failures:
  - cause by developer
  - due to Aquifer policy result
Conclusion

• presented the Aquifer security framework that assigns host export restrictions on all data accessed as part of a UI workflow.

• key insight was that when applications in modern operating systems share data, it is part of a larger workflow to perform a user task.

• enable applications to sensibly retain control of their data after it has been shared as part of the user’s tasks.