Mobile Malware

John Mitchell
Outline

• Mobile malware
  – Common cases involve command and control, information theft

• Identifying malware
  – Detect at app store rather than on platform

• Classification study of mobile web apps
  – Entire Google Play market as of 2014
  – 85% of approx 1 million apps use web interface
Mobile Malware

What are cyber criminals doing with smartphones?
Cyber criminals had 8 primary motivations for creating malware.

MALWARE/PERCENTAGE OF MALWARE

Remote controlling smartphones
Obtaining services or benefits
Stealing private data
Charging unauthorized fees
Damaging systems
Installing malicious software
Installing fraudulent applications
Spreading malicious code
Some Trends

- Aggressively displaying ads
- Sending SMS
- Potentially malicious software
- Remote control over device
- Download malware
- Stealing user's data
- Stealing money from bank accounts
- Others

© Kaspersky Lab
iPhone: Operation Pawn Storm

blog.trendmicro.com/.../pawn-storm-update-ios-espionage-app-found/

• Trend Micro:
  – “an active economic and political cyber-espionage operation that targets … military, governments, defense industries, and the media.”
  – Infects individuals to get to organizations

• Xagent
  – iOS 7: app icon is hidden, runs in background, restarts if terminated
  – iOS 8: app icon is visible; doesn’t automatically restart

• Apparently, iOS device needs to be jailbroken
  – Exact install process unknown
  – May require social engineering.
XAgent app

• Collects user information
  – Collect text messages
  – Access contact lists, pictures, geo-location data
  – Start voice recording, read WiFi status
  – Get a list of installed apps, list of processes

• Command and Control (C&C) Communication
  – HTTP POST request to send messages
  – GET request to receive commands
Android malware example

Install malicious “conference app”

WUC’s Conference in Geneva

On behalf of all at the World Uyghur Congress (WUC), the Unrepresented Nations and Peoples Organization (UNPO) and the Society for Threatened Peoples (STP), Human Rights in China: Implications for East Turkestan, Tibet and Southern Mongolia.

In what was an unprecedented
Malware behavior triggered by C&C server (Chuli)

WUC's Conference in Geneva
On behalf of all at the Word Uyghur Congress (WUC), the Unrepresented Nations and Peoples Organization (UNPO) and the Society for Threatened Peoples (STP), Human Rights in China: Implications for East Turkestan, Tibet and Southern Mongolia
In what was an unprecedented
Outline

• Mobile malware
  – Common cases involve command and control, information theft

→ Identifying malware
  – Detect at app store rather than on platform

• Classification study of mobile web apps
  – Entire Google Play market as of 2014
  – 85% of approx 1 million apps use web interface
STAMP Admission System

Static Analysis
More behaviors, fewer details

Dynamic Analysis
Fewer behaviors, more details

Alex Aiken,
John Mitchell,
Saswat Anand,
Jason Franklin
Osbert Bastani,
Lazaro Clapp,
Patrick Mutchler,
Manolis Papadakis
Abstract program execution

- States: mapping of variable names to values
- Transitions: relation on pairs of states
- Traces: sequence of states or state, transition pairs
Analysis

Step 1: Convert bytecode to intermediate format (called Quads)

Step 2: Compute call graph using Class Hierarchy Analysis

Step 3: Build an edge-labeled graph $G$ by processing Quads of each class

Step 4: Add new edges to $G$ as per a set of rules until no rules apply
Data Flow Analysis

- Source-to-sink flows
  - Sources: Location, Calendar, Contacts, Device ID etc.
  - Sinks: Internet, SMS, Disk, etc.
Data Flow Analysis in Action

**Malware/Greyware Analysis**
- Data flow summaries enable enterprise-specific policies

**API Misuse and Data Theft Detection**

**Automatic Generation of App Privacy Policies**
- Avoid liability, protect consumer privacy

**Vulnerability Discovery**

Privacy Policy
This app collects your:
- Contacts
- Phone Number
- Address
Challenges

• Android is 3.4M+ lines of complex code
  o Uses reflection, callbacks, native code

• **Scalability:** Whole system analysis impractical

• **Soundness:** Avoid missing flows

• **Precision:** Minimize false positives
• Model Android/Java
  ○ Sources and sinks
  ○ Data structures
  ○ Callbacks
  ○ 500+ models

• Whole-program analysis
  ○ Context sensitive
Building Models

• 30k+ methods in Java/Android API
  o 5 mins x 30k = 2500 hours

• Follow the permissions
  o 20 permissions for sensitive sources
    ▪ ACCESS_FINE_LOCATION (8 methods with source annotations)
    ▪ READ_PHONE_STATE - (9 methods)
  o 4 permissions for sensitive sinks
    ▪ INTERNET, SEND_SMS, etc.
Identifying Sensitive Data

android.Telephony.TelephonyManager: String getDeviceId()

- Returns device IMEI in String
- Requires permission GET_PHONE_STATE

@STAMP(
    SRC ="$GET_PHONE_STATE.deviceid",
    SINK ="@return"
)
Data We Track (Sources)

- Account data
- Audio
- Calendar
- Call log
- Camera
- Contacts
- Device Id
- Location
- Photos (Geotags)
- SD card data
- SMS

30+ types of sensitive data
Data Destinations (Sinks)

- Internet (socket)
- SMS
- Email
- System Logs
- Webview/Browser
- File System
- Broadcast Message

10+ types of exit points
Currently Detectable Flow Types

396 Flow Types

Unique Flow Types = Sources \times Sink
Contact Sync for Facebook (unofficial)

Description:
This application allows you to synchronize your Facebook contacts on Android.

IMPORTANT:
* "Facebook does not allow [sic] to export phone numbers or emails. Only names, pictures and statuses are synced."
* "Facebook users have the option to block one or all apps. If they opt for that, they will be EXCLUDED from your friends list."

Privacy Policy: (page not found)
Chuli source-to-sink flows
## Contact Sync Permissions

<table>
<thead>
<tr>
<th>Category</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your Accounts</td>
<td>AUTHENTICATE_ACCOUNTS</td>
<td>Act as an account authenticator</td>
</tr>
<tr>
<td></td>
<td>MANAGE_ACCOUNTS</td>
<td>Manage accounts list</td>
</tr>
<tr>
<td></td>
<td>USE_CREDENTIALS</td>
<td>Use authentication credentials</td>
</tr>
<tr>
<td>Network Communication</td>
<td>INTERNET</td>
<td>Full Internet access</td>
</tr>
<tr>
<td></td>
<td>ACCESS_NETWORK_STATE</td>
<td>View network state</td>
</tr>
<tr>
<td>Your Personal Information</td>
<td>READ_CONTACTS</td>
<td>Read contact data</td>
</tr>
<tr>
<td></td>
<td>WRITE_CONTACTS</td>
<td>Write contact data</td>
</tr>
<tr>
<td>System Tools</td>
<td>WRITE_SETTINGS</td>
<td>Modify global system settings</td>
</tr>
<tr>
<td></td>
<td>WRITE_SYNC_SETTINGS</td>
<td>Write sync settings (e.g. Contact sync)</td>
</tr>
<tr>
<td></td>
<td>READ_SYNC_SETTINGS</td>
<td>Read whether sync is enabled</td>
</tr>
<tr>
<td></td>
<td>READ_SYNC_STATS</td>
<td>Read history of syncs</td>
</tr>
<tr>
<td>Your Accounts</td>
<td>GET_ACCOUNTS</td>
<td>Discover known accounts</td>
</tr>
<tr>
<td>Extra/Custom</td>
<td>WRITE_SECURE_SETTINGS</td>
<td>Modify secure system settings</td>
</tr>
</tbody>
</table>
Possible Flows from Permissions

Sources
- READ_CONTACTS
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- GET_ACCOUNTS
- INTERNET

Sinks
- INTERNET
- WRITE_SETTINGS
- WRITE_CONTACTS
- WRITE_SECURE_SETTINGS
- WRITE_SETTINGS
Expected Flows

Sources
- READ_CONTACTS
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- GET_ACCOUNTS
- INTERNET

Sinks
- INTERNET
- WRITE_SETTINGS
- WRITE_CONTACTS
- WRITE_SECURE_SETTINGS
- WRITE_SETTINGS
Observed Flows

FB API
Source: FB_Data
Write Contacts
Sink: Contact_Book

Read Contacts
Source: Contacts
Send Internet
Sink: Internet
Outline

• Mobile malware
  – Common cases involve command and control, information theft

• Identifying malware
  – Detect at app store rather than on platform

Classification study of mobile web apps
  – Entire Google Play market as of 2014
  – 85% of approx 1 million apps use web interface
  – 28% have at least one vulnerability
A Large-Scale Study of Mobile Web App Security

Patrick Mutchler, Adam Doupe, John Mitchell, Chris Kruegel, Giovanni Vigna
Mobile Apps
Mobile Apps
Mobile Apps
Mobile Web Apps

- Mobile web app: embeds a fully functional web browser as a UI element
Obj foo = new Object();
addJavascriptInterface(foo, 'f');
JavaScript Bridge

f.bar();
Why?

• Full-featured mobile web apps

• Expose phone functionality to JavaScript
Security Concerns

• Who can access the bridge?
  – Everyone
46 U.S. CRUISE MISSILES 'ONE OR TWO' KEY KHORASAN KILLED 'A DOZEN' CIVILIANS DEAD

Complete Isolation
Java

JavaScript

```javascript
f.bar();
```
f.bar();
Static Analysis

• How many mobile web apps?

• How many use JavaScript Bridge?

• How many vulnerable?
Experimental Results

- 737,828 free apps from Google Play (Oct ’13)
- 563,109 apps embed a browser
- 219,404 use the JavaScript Bridge
- 107,974 have at least one security violation
Most significant vulnerabilities

1. Loading untrusted web content

2. Leaking URLs to foreign apps

3. Exposing state changing navigation to foreign apps
1. Loading untrusted web content

2. Leaking URLs to foreign apps

3. Exposing state changing navigation to foreign apps
“You should restrict the web-pages that can load inside your WebView with a whitelist.”

- Facebook
“…only loading content from trusted sources into WebView will help protect users.”

- Adrian Ludwig, Google
1. Navigate to untrusted content
// In app code
myWebView.loadUrl(“foo.com”);
// In app code
myWebView.load("foo.com");

<!-- In HTML -->
<a href="foo.com">click!</a>
// In app code
myWebView.load("foo.com");

<!-- In HTML -->
<a href="foo.com">click!</a>

<!-- More HTML -->
<iframe src="foo.com"/>
// In app code
myWebView.loadUrl("foo.com");

<!-- In HTML -->
<a href="foo.com">click!</a>

<!-- More HTML -->
<iframe src="foo.com"/>

// In JavaScript
window.location = "foo.com";
public boolean shouldOverrideUrlLoading(WebView view, String url) {

    // False  -> Load URL in WebView
    // True    -> Prevent the URL load

}
public boolean shouldOverrideUrlLoading(WebView view, String url) {

    String host = new URL(url).getHost();
    if (host.equals("stanford.edu"))
        return false;
    log("Overrode URL: " + url);
    return true;
}
public boolean shouldOverrideUrlLoading(WebView view, String url) {

    String host = new URL(url).getHost();
    if (host.equals("stanford.edu"))
        return false;
    log("Overrode URL: " + url);
    return true;
}
Reach Untrusted Content?

• 40,084 apps with full URLs and use JavaScript Bridge

• 13,683 apps (34%) can reach untrusted content
What does untrusted mean?
Use HTTPS?

• 152,706 apps with partially computed URLs
• 87,968 apps (57%) with HTTP URLs
Handling SSL Errors

onReceivedSslError

1. handler.proceed()
2. handler.cancel()
3. view.loadUrl(...)

Mishandling SSL Errors

• 117,974 apps implement `onReceivedSslError`
• 29,652 apps (25%) **must** ignore errors
Results
## Primary results

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>% Relevant</th>
<th>% Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsafe Nav</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>HTTP</td>
<td>40</td>
<td>56</td>
</tr>
<tr>
<td>Unsafe HTTPS</td>
<td>27</td>
<td>29</td>
</tr>
</tbody>
</table>
Outdated Apps

- Unsafely navigated applications
- HTTP
- Unsecurely transmitted HTTPS

The normalized vulnerability rate is shown for each category, indicating the risk associated with each type of outdated app.
Libraries

- 29% unsafe nav
- 51% HTTP
- 53% unsafe HTTPS
Additional security issues

Analyze 998,286 free web apps from June 2014

<table>
<thead>
<tr>
<th>Mobile Web App Feature</th>
<th>% Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript Enabled</td>
<td>97</td>
</tr>
<tr>
<td>JavaScript Bridge</td>
<td>36</td>
</tr>
<tr>
<td>shouldOverrideUrlLoading</td>
<td>94</td>
</tr>
<tr>
<td>shouldInterceptRequest</td>
<td>47</td>
</tr>
<tr>
<td>onReceivedSslError</td>
<td>27</td>
</tr>
<tr>
<td>postUrl</td>
<td>2</td>
</tr>
<tr>
<td>Custom URL Patterns</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vuln</th>
<th>% Relevant</th>
<th>% Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsafe Navigation</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>Unsafe Retrieval</td>
<td>40</td>
<td>56</td>
</tr>
<tr>
<td>Unsafe SSL</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>Exposed POST</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Leaky URL</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>
Takeaways

• Apps must not load untrusted content into WebViews

• Able to identify violating apps using static analysis

• Vulnerabilities are present in the entire app ecosystem
Outline

• Mobile malware
  – Common cases involve command and control, information theft

• Identifying malware
  – Detect at app store rather than on platform

• Classification study of mobile web apps
  – Entire Google Play market as of 2014
  – 85% of approx 1 million apps use web interface
  – 28% have at least one vulnerability
Summary

• Analyze a dataset of 737,828 Android apps
• Found large number of apps contain severe vulnerabilities
• 37,418 apps are vulnerable to a remote code execution exploit when run on any Android device, because of security oversight in older versions and slow adoption of safe versions
• 45,689 apps are vulnerable to a remote code execution exploit when run on 73% of the in-use Android devices.
• Offer recommendations for developers who wish to avoid these vulnerabilities.