Inside Job: Understanding and Mitigating the Threat of External Device Mis-Bonding (DMB) on Android

Muhammad Naveed¹
Xiaoyong Zhou²
Soteris Demetriou¹
XiaoFeng Wang²
Carl A. Gunter¹

¹University of Illinois at Urbana-Champaign
²Indiana University at Bloomington
External devices enhance smartphone’s capabilities
iThermometer

Temperature monitoring device for babies and elderly persons
Other devices

• FDA approved devices
• Sync information to EHR or web-account
• Wrong amount of insulin can kill
Bluetooth Devices

Bluetooth Enabled Device Annual Shipments, Major Markets
World Market, Forecast: 2000 to 2018

Source: ABI Research, Bluetooth Service

Source (for both numbers and figure): http://www.bluetooth.com
Bluetooth Devices

Total devices shipped

2012
9 Billion

Source (for both numbers and figure): http://www.bluetooth.com
Bluetooth Devices

Total devices shipped

2012: 9 Billion
2016: 20 Billion

Source: ABI Research, Bluetooth Service

Source (for both numbers and figure): http://www.bluetooth.com
Bluetooth Devices

**Total devices shipped**

<table>
<thead>
<tr>
<th>Year</th>
<th>Total Units Shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>9 Billion</td>
</tr>
<tr>
<td>2016</td>
<td>20 Billion</td>
</tr>
<tr>
<td>2018</td>
<td>30 Billion</td>
</tr>
</tbody>
</table>

Source: ABI Research, Bluetooth Service

Source (for both numbers and figure): http://www.bluetooth.com
Fundamental Problem
Fundamental Problem
Fundamental Problem

Pair with phone
Fundamental Problem
Fundamental Problem

Doesn’t pair with apps
External devices and Android design

- Android is not designed to protect these external devices

- We designed the following two attacks to show the problem:
  - Data-stealing attack
  - Data-injection attack
Device Mis-bonding Attacks
Adversary Model

- A malicious app with BLUETOOTH and BLUETOOTH_ADMIN permissions is installed on the victim’s phone

- Additionally, physical proximity is required for data-injection attacks
Normal Scenario
Normal Scenario
Normal Scenario
Normal Scenario
Normal Scenario
Normal Scenario
Normal Scenario
Normal Scenario
Data-stealing Attack
Data-stealing Attack
Data-stealing Attack
Data-stealing Attack
Data-stealing Attack
Data-stealing Attack
Technical Challenges
Technical Challenges

• When to steal data?

• Device is not always connected

• Naive strategy: Periodic device discovery
  • Increased power usage
  • Not stealthy
Technical Challenges

• When to steal data?
  • Device is not always connected
  • Naive strategy: Periodic device discovery
    • Increased power usage
    • Not stealthy

• Observation: Execution of device’s official app is a strong indication of the device being ON and in connection range.
  • getRunningAppProcesses() or linux command ps can find if the official app is running in O(n)
  • getRunningTasks() can find if the official app is running in O(1), with additional GET_TASKS permission
Technical Challenges
Technical Challenges

• If official app is in communication with the target device, the malicious app cannot connect to it.
Technical Challenges

• If official app is in communication with the target device, the malicious app cannot connect to it.

• To get data, malicious app needs to connect to the target device using one of the following strategies:
Technical Challenges

• If official app is in communication with the target device, the malicious app cannot connect to it.

• To get data, malicious app needs to connect to the target device using one of the following strategies:
  • disruption: simply disrupt the official app connect, reliable but less stealthy
Technical Challenges

- If official app is in communication with the target device, the malicious app cannot connect to it.

- To get data, malicious app needs to connect to the target device using one of the following strategies:
  - **disruption**: simply disrupt the official app connect, reliable but less stealthy
  - **pre-connection**: right before the official app connects, reliable and stealthy
Technical Challenges

- If official app is in communication with the target device, the malicious app cannot connect to it.

- To get data, malicious app needs to connect to the target device using one of the following strategies:
  - **disruption**: simply disrupt the official app connect, reliable but less stealthy
  - **pre-connection**: right before the official app connects, reliable and stealthy
  - **post-connection**: right after the official apps disconnects, reliable and stealthy
## Success Rate

<table>
<thead>
<tr>
<th>Target Device</th>
<th>Pre-connection</th>
<th>Post-connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bodymedia Link Armband</td>
<td>99/100</td>
<td>100/100</td>
</tr>
<tr>
<td>iThermometer</td>
<td>42/100</td>
<td>100/100</td>
</tr>
<tr>
<td>Nonin Pulseoximtereter</td>
<td>99/100</td>
<td>92/100</td>
</tr>
<tr>
<td>MyGlucoHealth Glucometer</td>
<td>100/100</td>
<td>0/100*</td>
</tr>
</tbody>
</table>

*device turns off automatically after sending data to the phone
# Stealthiness

<table>
<thead>
<tr>
<th>Technique</th>
<th>Avg. Power Consumption</th>
<th>Sampling Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>getRunningAppProcesses()</td>
<td>8mW</td>
<td>2 samples/s</td>
</tr>
<tr>
<td>getRunningTasks()</td>
<td>3mW</td>
<td>2 samples/s</td>
</tr>
<tr>
<td>connect()</td>
<td>17mW</td>
<td>0.18 samples/s</td>
</tr>
<tr>
<td>startDiscovery()</td>
<td>15mW</td>
<td>0.054 samples/s</td>
</tr>
<tr>
<td>Facebook</td>
<td>18mW</td>
<td></td>
</tr>
<tr>
<td>Gmail</td>
<td>1mW</td>
<td></td>
</tr>
</tbody>
</table>
Data-injection Attack
Data-injection Attack
Data-injection Attack
Data-injection Attack
Data-injection Attack
Data-injection Attack
Data-injection Attack
Data-injection Attack
Data-injection Attack
Device Cloning

- Target device MAC address is sufficient for cloning
- Target device name and UUID can make clone indistinguishable from original device
- This information can be obtained using `BluetoothAdapter.getBondedDevices()`
- SpoofTooph temporarily overwrites the MAC address of bluetooth dongle
Link key reset

- `createSecureRfcommSocket()` uses a link-key for encryption and authentication
- Clone cannot connect without this key
Link key reset

- `createsecureRfcommSocket()` uses a link-key for encryption and authentication

- Clone cannot connect without this key

- Observation: We cannot get the link key, but can simply replace one
Link key reset

- `createsecureRfcommSocket()` uses a link-key for encryption and authentication

- Clone cannot connect without this key

- Observation: We cannot get the link key, but can simply replace one

- Android’s pairing and un-pairing methods are not directly available to programmers
Connection Race
Connection Race

• When both clone and original device are in vicinity, which will connect to the phone?
Connection Race

• When both clone and original device are in vicinity, which will connect to the phone?

• Observation: How Bluetooth socket works?
  • Devices are in slave mode and smartphone initiate connection
  • **Paging**: Devices switches between page sleep and page scan mode
  • Device accept connection only in page scan mode
  • To save power these devices have large page sleep period and small page scan period
  • Adversary can set arbitrary page sleep and page scan period in allowed range
**Adversary always wins!**

<table>
<thead>
<tr>
<th>Distance of cloned device</th>
<th>1 feet</th>
<th>20 feet (with wall in between)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of observations</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>No. of times original device responded</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>No. of times cloned device responded</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

- Using default page sleep and page scan time period (much more than minimum)
- Clone’s radio had 2.5mW radio while original device had 100mW radio
Pervasiveness of Device-Misbonding attacks
Measurement

• The problem discussed before are caused by lack of bonding between external device and app

• Device and app manufacturers can fix this issue using appropriate authentication mechanism

• We conducted a measurement study to see if any device already have such security mechanism
Methodology

• App collection: Manually searched for bluetooth apps using following search queries:
  • “Bluetooth Door Lock”
  • “Bluetooth Health”
  • “Bluetooth Medical Devices”
  • “Bluetooth Meter”

• Out of 90 apps, 68 apps involved some private information

• Decompiled the 68 apps and studied the source code
Classification of apps

- Heart Rate Monitor
- Activity Monitor
- Medical Devices (Blood pressure, Glucose meter, thermometer etc)
- Remote Actuators (Remote door opener, remote car starter, etc)
- Baby Monitor
- Sound Recorder
- Other (File transfer, bluetooth chat etc)
Methodology

• Searched for authentication-related programming structures

• Authentication is always based on some secret. It can come from:
  • external inputs e.g. UI,
  • web communication,
  • internal memory or
  • generated by some cryptographic operations
Manual Analysis

- Manual analysis of 20 apps. The other 48 apps were filtered out by locations of their suspicious APIs.

<table>
<thead>
<tr>
<th>Authentication Methods</th>
<th>Libraries/ Functions used</th>
<th>Total</th>
<th>Apps with app-device authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Crypto</strong></td>
<td>javax.crypto, bouncycastle</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td><strong>Internal storage</strong></td>
<td>openFileInput()</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td><strong>Web communication</strong></td>
<td>HttpClient</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td><strong>UI for app-device authentication</strong></td>
<td>Manual</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Defense
Dabinder

Source code
https://github.com/DabinderAndroid/extDroid.git
Solution

- Theoretically, device manufactures can provide protecting
  - Upgrading both app and hardware, some apps come from third parties
  - Billions of existing devices
  - Case-by-case fix can be ugly
- Better alternative is to provide an Android OS-level solution
Dabinder Design

- Pairing Control
  - Maps external device MAC address to app

- Connection Control
  - Before socket established device-app mapping is checked

- Unpairing Control
  - Unpairing needs user interaction
## Performance

<table>
<thead>
<tr>
<th>Functions</th>
<th>Original</th>
<th>Dabinder</th>
<th>Delays</th>
</tr>
</thead>
<tbody>
<tr>
<td>BluetoothSocket</td>
<td>mean 0.0317 SD 0.0059 ms</td>
<td>mean 0.0353 SD 0.0153 ms</td>
<td>0.0036 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>connectSocket</td>
<td>mean 63.1670 SD 14.7098 ms</td>
<td>mean 86.5152 SD 14.2201 ms</td>
<td>23.3482 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>removeBond</td>
<td>mean 0.5319 SD 0.1863 ms</td>
<td>mean 0.5493 SD 0.1822 ms</td>
<td>0.017ms</td>
</tr>
</tbody>
</table>
Dabinder Architecture
Conclusion

- Device Mis-Bonding (DMB) threat is serious
- Confidentially threat: Can lead to theft of private information
- Integrity threat: Can also compromise the integrity of sensitive data
- OS-level solution provides reasonable protection to bind app to the device
Thank you!
Please watch video demos at:

http://goo.gl/XXSGGU

(link is case-sensitive)

Defense: https://github.com/DabinderAndroid/extDroid.git