Design and Implementation of an Android Host-based Intrusion Prevention System

Mingshen Sun†, Min Zheng†, John C.S. Lui†, Xuxian Jiang*

†The Chinese University of Hong Kong
*NC State University & Qihoo 360

December 11, 2014
Introduction

Android Security Status

- smartphone: home, company, car
- smartphone security: hackers’ biggest target
- Android: 80% of all smartphone shipments
- 97% malware target Android
- 17,000 new Android malware found in the second quarter of 2013
Protection

- permission mechanism
  - alert users about required permission of an installed application
  - plenty of Android permission abuses

- anti-virus software
  - static analysis & signature detection
  - malware transformation can bypass anti-virus software
  - ADAM, DroidChameleon

- dynamic analysis
  - run-time behaviors
  - TaintDroid, DroidScope, DroidBox for malware analysis
  - post-mortem analysis: the malware under investigation may have already infected many devices
Introduction

Host-based intrusion prevention system (HIPS)

- installed software on a mobile device
- monitor suspicious activities
- block and report malicious behaviors
- event at run-time
- dynamically intercept the applications
- notify users when the malware invoke dangerous APIs
- Jinshan Mobile Duba, LBE, 360 Mobile Safe, etc
Introduction

Current status of HIPS: three approaches

- system patching: modify Android OS with new permission management functions
- application repackaging
  - disassemble a mobile application
  - add new policy enforcement
- API hooking: intercept mobile application’s API calls at run-time
- limitations and bring new vulnerabilities
**Introduction**

**Patronus**

- enhanced HIPS
- performs a secure policy enforcement
- dynamically detects existing malware using run-time information
- does **not** require any modifications on the Android firmware
- system level inspection
- host-based run-time detection
Android Architecture

- five functional layers: kernel, libraries, runtime support, application framework and applications
- Java, Android SDK, Dalvik Virtual Machine
- Sandbox
- Inter-process communication
Background

Binder Mechanism

- inter-process communication mechanism
- IPC workflow
  1. contact Service Manager
  2. ask ISms Service to send message
  3. process the request and send through driver
- Binder transaction
- client-server communication model

Client

- zygote
- fork
- Applications

Server

- Service Manager
- Telephony Service
- ISms Service
- System Services
- Binder Driver (/dev/binder)
- SMS Driver

User Space

Kernel Space
Systematical analysis on three popular HIPS frameworks

- system patching
- application repackaging
- API hooking
System Patching
- third-party firmware
- patches from hackers, developers for policy enforcement
- hidden function (testing function) in latest Android system: App 0ps

Limitations
- 10% of Android phones are powered by latest Android
- third-party firmware is not matched
- App 0ps cannot prevent intrusion at run-time
Application Repackaging: manifest file

- Android application file (apk file)
- AndroidManifest.xml file
- delete certain permission declarations in the AndroidManifest.xml file

Application Repackaging: reverse engineering

- Dalvik byte code is easy to reversed to readable codes
- several tools provide assembling and disassembling functions
- HIPS can insert stubs around sensitive Android APIs
- e.g., requestLocationUpdates() API
Application Repackaging: vulnerabilities

- incomplete security coverage
- policy enforcement on all possible APIs
- native code to bypass policy stubs
- bugs of disassembling tools
Basic Flow of API Hooking (four steps)

1. gaining root or system privilege
   - root tools
   - third-party firmwares
   - root loopholes can be patched by existing system using hooking

2. injecting a shared library object file (so file)
   - ptrace() target process
   - inject shellcode
   - utlize dlopen and dltsy to inject an so file

3. carrying out hooking on target APIs
   - global object in DVM: gDvm in libdvm.so
   - pointers to the calling function

4. loading policy enforcement function: jar library file
sending an SMS message

```java
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(
    "phoneNumber", null, "message", null, null
);```

Mingshen Sun (CUHK)
Android HIPS
December 11, 2014
hooking on the client side

- JNI method: native transact()
- corresponding native method: android_os_BinderProxy_transact()
- zygote process
- hook native transact method to intercept all method callings
- security problems?
Vulnerability of Existing HIPS Products

- The hooking operations are done in the same sandbox of the application.
- Client HIPS (LBE) has more than ten million users and pre-installed in a number of Android devices.
- Memory structure

```bash
# cat /proc/1459/maps

... 6b811000-6b813000 r--s 00015000 b3:1d 927  <-- Mapped memory region
/data/data/com.lbe.security/app_hips/client.jar  <-- File path
6b831000-6b860000 r--p 00000000 b3:1d 1185
/data/dalvik-cache/data@com.lbe.security@app_hips@client.jar@classes.dex
6cc47000-6cc52000 r-xp 00000000 b3:1d 262788
/data/data/com.lbe.security/app_hips/libclient.so
...```
Vulnerability of Existing HIPS Products (Proof of concept)

- modify method pointer to the original one
- create own transact implementation to bypass using native code
- hooking on the client side cannot prevent intrusion

```java
Parcel data = Parcel.obtain();
data.writeString("com.android.internal.telephony.ISms");
data.writeString("12345678");
data.writeString(null);
data.writeString("Premium SMS Message!");
data.writeInt(0);
data.writeInt(0);

// Bypass
isms.transact(5, data, reply, 0);
native pwnTransact(isms, 5, data, reply, 0);

// JNI
IBinder *target = ibinderForJavaObject(jniEnv, isms);
target->transact(code, *data, reply, flags);
```

Client

HIPS

Java

JNI

Native (C/C++)
Patronus

- realize API hooking on both client side and server side

Hooking on the server side

- two approaches to conduct API hooking on the server side
- Java API hooking
  - JNI method: execTransact
- hook the Service Manager
System Design of Patronus

Building Blocks

- Patronus Application
  - `ptrace()` injection

- Patronus Service
  - intercept all transactions
  - system protection

- Injected files
  - injected .so, .jar
  - applications
  - Service Manager

- Policy database
  - policy rules
  - sensitive transactions: `sendText`, `requestLocationUpdates`

Mingshen Sun (CUHK)  
Android HIPS  
December 11, 2014
### Table: Intrusive Transaction List

<table>
<thead>
<tr>
<th>TD</th>
<th>TC</th>
<th>TC Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.android.internal</td>
<td>4</td>
<td>sendData</td>
</tr>
<tr>
<td>.telephony.ISms</td>
<td>5</td>
<td>sendText</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>sendMultipartText</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>com.android.internal</td>
<td>1</td>
<td>dial</td>
</tr>
<tr>
<td>.telephony.ITelephony</td>
<td>2</td>
<td>call</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>getCellLocation</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>android.location</td>
<td>1</td>
<td>requestLocationUpdates</td>
</tr>
<tr>
<td>.ILocationManager</td>
<td>5</td>
<td>getLastLocation</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

TD: Transaction Descriptor, TC: Transaction Code
Intrusion Prevention of Patronus

- client-side policy enforcement
- alert notification for users
- server-side policy enforcement
- not checked by client? suspicious transaction!

Figure: Alert of intrusive transaction (sendText).
Dynamic Detection: Malware Transaction Forensics

- malware triggering
  - record run-time transaction information

- malware transaction tagging
  - tagging malicious transaction

- transaction footprint
Dynamic Detection: Two-phase Dynamic Detection

- Phase one: correlation detection
  - $V_{\text{runtime}}$
  - $V_{\text{malware}}$
  - Pearson correlation
  - Correlation comparison

Definition

Define $V$ as the transaction vector over a transaction footprint $S$ where $V = [v_1, v_2, \ldots, v_n | v_i = N_i]$.

$$r = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}}.$$  \hspace{1cm} (1)

where $X = V_{\text{runtime}}$, $Y = V_{\text{malware}}$
Dynamic Detection: Two-phase Dynamic Detection

- Phase two: transaction footprint comparison
  - \( r \) is higher than a pre-defined threshold
  - malicious transaction
  - decisive fields indicating the malicious behaviors
  - e.g., sendText transaction: destination address
Dynamic Detection

- run-time information
- semantic information rather than system calls
- performance overhead compared to system-call-based system
Large Scale Evaluation

Intrusion Prevention & Dynamic Detection

- 500 legitimate applications
- 213 BaseBridge, 9 FakeAV, 15 MobileTx
- 49 intrusive transactions

Evaluation intrusive transaction

- 500 pseudo-random user events: clicks, touches, gestures, etc.

Table: Top 10 Intrusive Transactions

<table>
<thead>
<tr>
<th>Transaction Name</th>
<th>Total #</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL_TRANSACTION</td>
<td>3,508</td>
</tr>
<tr>
<td>REGISTER_RECEIVER_TRANSACTION</td>
<td>2,960</td>
</tr>
<tr>
<td>START_ACTIVITY_TRANSACTION</td>
<td>1,734</td>
</tr>
<tr>
<td>TRANSACTION_getDeviceId</td>
<td>1,732</td>
</tr>
<tr>
<td>GET_CONTENT_PROVIDER_TRANSACTION</td>
<td>1,400</td>
</tr>
<tr>
<td>QUERY_TRANSACTION</td>
<td>1,303</td>
</tr>
<tr>
<td>TRANSACTION_getSubscriberId</td>
<td>333</td>
</tr>
<tr>
<td>TRANSACTION_requestLocationUpdates</td>
<td>228</td>
</tr>
<tr>
<td>INSERT_TRANSACTION</td>
<td>139</td>
</tr>
<tr>
<td>TRANSACTION_getCallState</td>
<td>90</td>
</tr>
</tbody>
</table>
Dynamic Detection

- true positive (TP), true negative (TN), false positive (FP), false negative (FN), precision and F-score
- MobileTX crashed

<table>
<thead>
<tr>
<th>Malware</th>
<th># of Samples</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseBridge</td>
<td>213</td>
<td>186</td>
<td>495</td>
<td>5</td>
<td>27</td>
<td>0.87</td>
<td>0.92</td>
</tr>
<tr>
<td>FakeAV</td>
<td>9</td>
<td>9</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MobileTx</td>
<td>15</td>
<td>11</td>
<td>494</td>
<td>6</td>
<td>4</td>
<td>0.65</td>
<td>0.69</td>
</tr>
</tbody>
</table>
Large Scale Evaluation

Benchmark Results

- Quadrant Standard Edition
- LG Nexus 5 (Qualcomm Snapdragon 800 2.26GHz CPU)
- Android 4.4.2
- Baseline v.s. Patronus

<table>
<thead>
<tr>
<th>Test</th>
<th>Baseline</th>
<th>Patronus</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>8,914</td>
<td>8,285</td>
<td>7.1%</td>
</tr>
<tr>
<td>CPU</td>
<td>20,383</td>
<td>20,205</td>
<td>0.9%</td>
</tr>
<tr>
<td>Memory</td>
<td>14,354</td>
<td>13,211</td>
<td>8.0%</td>
</tr>
<tr>
<td>I/O</td>
<td>7,274</td>
<td>6,482</td>
<td>10.9%</td>
</tr>
<tr>
<td>2D</td>
<td>333</td>
<td>330</td>
<td>0.1%</td>
</tr>
<tr>
<td>3D</td>
<td>2,230</td>
<td>2,195</td>
<td>1.6%</td>
</tr>
</tbody>
</table>
Large Scale Evaluation

Evaluation
- 1,000 com.android.internal.telephony.IPhoneSubInfo transaction
- 890 milliseconds
- 998 milliseconds (Patronus)
- 11.1 % performance overhead

Power consumption
- daily usage
- standby mode for 24 hours: 1% power consumption
- heavy user
- game playing: 3% power consumption
Conclusion

Summary

- systematically analyze three popular HIPS frameworks
- design and implement a secure architecture HIPS: patronus
- two-phase detection algorithm based on run-time information
- large scale evaluation on effectiveness and performance
Thank you!

Thank you very much! Questions?