Android Security and the Elusive HSM

Mobile Digital Wallet Security Summit

Jon Oberheide
Duo Security
jono@duosecurity.com
Introduction

- Jon Oberheide
- CTO, Duo Security

Today

- High level look at Android security and how HSMs can help
- Only 30 minutes! Lots of external refs!
Use case

- Mobile platform security is important (surprise!)
- Apps have data to protect (confidentiality/integrity)

Examples
- Duo Mobile 2FA app
- Visa Mobile app? V.me?
Threat model

- Attacker wants access to sensitive data
  - Stages of attack/capability
  - Points of attack disruption/mitigation

Diagram:
- Unprivileged code execution
- Privileged code execution
- App tampering and data theft

???

???

???
Agenda

- Introduction
- Gaining code execution
- Escalating privileges
- Recent HSM developments
- Wrap-up
Code execution

- First step is getting a foothold on the mobile device
  - Need code execution on victim's device

- A couple easy vectors for attackers:
  - Social vector: malicious applications
  - Technical vector: exploitation of existing applications
PoC malicious apps

- In the past, mostly researcher PoCs
Real-world malicious apps

- Nowadays, real-world malware is out there
Stopping malicious apps

- Mobile antivirus
  - Reactive signature-based detection of malicious apps

- Mobile malware exists, but these folks tend to push the hype a bit beyond reality
Google's Bouncer

- Google's Bouncer
  - Guards the entry point to the Android Market
  - Dynamically analyze submitted apps and block malicious apps from being published

- Dynamic analysis is **hard**
- Fingerprinting Bouncer is **easy**
- Catches some malware, but easy to bypass

BUSTICATI PRODUCTIONS PRESENTS

DISSECTING THE ANDROID BOUNCER

STARRING

DR. OBERHEIDE and DR. MILLER
Client-side exploitation

- Threat of client-side applications
  - Large attack surface of native code
  - Traditional memory corruption vulns
  - Similar to desktop client-side threats
- Browser/PDF/Docs = huge attack surface
- Standard Linux hardening mechanisms
  - NX, ASLR, RELRO, BIND_NOW, etc
- Android exploit mitigations are getting better...
Exploit mitigation evolution

- Android exploit mitigations have **slowly** evolved over the years...
- Before Android 2.3.x, no NX/ASLR:

```
af01000-af02000 r-w 00001000 1f:03 607
/system/lib/libstdc++.so
af00000-af039000 r-xp 00000000 1f:03 487
/system/lib/libc.so
af039000-af0c000 r-w 00039000 1f:03 487
/system/lib/libc.so
af0c000-af047000 r-w afe3c000 00:00 0
b000000-b0013000 r-xp 00000000 1f:03 382
/system/bin/linker
b0013000-b0014000 r-w 00013000 1f:03 382
/system/bin/linker
b0014000-b001a000 r-xp b0014000 00:00 0
bed29000-c0000 r-xp befeb000 00:00 0
[stack]
#
```
Exploit mitigation evolution

- Android 2.3.x – Gingerbread
  - Finally got NX support!
  - But still ineffective ASLR:

```
Nexus S - 2.3.4 - vold
00008000-00014000 00008000-00014000 r-xp /system/bin/vold
00015000-0001b000 00015000-0001b000 rw-p [heap]
afd00000-afd40000 afd00000-afd40000 r-xp /system/lib/libc.so
b0001000-b0009000 b0001000-b0009000 r-xp /system/bin/linker
bea31000-bea52000 beccb000-bececc000 rw-p [stack]
EXEC 1 EXEC 2
```
Exploit mitigation evolution

• Android 4.0 – ICS
  • ASLR listed in the release notes as a new security feature!
  • But upon deeper inspection...

https://blog.duosecurity.com/2012/02/a-look-at-aslr-in-android-ice-cream-sandwich-4-0/
Exploit mitigation evolution

• Android 4.1 – Jelly Bean
  • Ok, this time we have ASLR for real...

https://blog.duosecurity.com/2012/07/exploit-mitigations-in-android-jelly-bean-4-1/
Code execution wrap-up

- First goal of attacker is getting a foot hold on the device with code execution.
- Either by compromising an existing app or tricking user into installing a malicious app.
  - Some strides made in exploit mitigations.
  - More general problem of malicious apps is hard.

Best to assume that malicious code/apps will be present on the user's device!
Agenda

- Introduction
- Gaining code execution
- Escalating privileges
- Recent HSM developments
- Wrap-up
So what happens now?

- With code execution the attacker can:
  - Change the behavior of the exploited app
  - Steal data used by the exploited app
- The attacker can't:
  - Affected other apps on the device
  - Steal data from other apps
- Thanks to the Android “sandbox”
Android app “sandboxing”

- Calling it a sandbox is a stretch
- Each application gets a unique uid/gid upon install

```
-rwxr-xr-x  1 10027 10027 2048 Nov 9 01:59 org.dyndns.devesh.flashlight
-rwxr-xr-x  1 10046 10046 2048 Dec 8 07:18 org.freedictionary
-rwxr-xr-x  1 10054 10054 2048 Feb 5 14:19 org.inodes.gus.scummvm
-rwxr-xr-x  1 10039 10039 2048 Mar 8 12:32 org.oberheide.org.brickdroid
```
Escalating privileges

• What does the privileged attack surface look like on an Android device?
  • Entire vanilla Linux kernel
  • + custom kernel modifications by Google
  • + custom drivers by third-party devs
  • Privileged system daemons (vold, etc)
  • Poorly written setuid binaries

Bottom line: Lots of attack surface to exploit!
Overview of privesc vulns

• Some vulns affect nearly all Android devices:

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASHMEM</td>
<td>kernel</td>
<td>custom Google mod</td>
</tr>
<tr>
<td>Exploid</td>
<td>init daemon</td>
<td>netlink source check</td>
</tr>
<tr>
<td>Gingerbreak</td>
<td>vold daemon</td>
<td>netlink source check</td>
</tr>
<tr>
<td>Levitator</td>
<td>kernel device driver</td>
<td>third-party kernel mod</td>
</tr>
<tr>
<td>Mempodroid</td>
<td>kernel</td>
<td>affected vanilla kernel</td>
</tr>
<tr>
<td>RageAgainstTheCage</td>
<td>adb daemon</td>
<td>setuid(2) return value</td>
</tr>
<tr>
<td>Wunderbar</td>
<td>kernel</td>
<td>affected vanilla kernel</td>
</tr>
<tr>
<td>ZergRush</td>
<td>libsysutils</td>
<td>memory corruption</td>
</tr>
<tr>
<td>Zimperlich</td>
<td>zygote</td>
<td>setuid(2) return value</td>
</tr>
</tbody>
</table>

Levitator exploit

• Levitator exploit
  • Targeted PowerVR vulnerability: /dev/pvrsrvkm
  • Allowed arbitrary kmem read/write
  • Affected popular S series devices
  • Patched in 2.3.6 after 10+ months

• Chain of custody?
  • Researcher → Google → Samsung → Imagination Tech → Manufacturers → Carriers

http://jon.oberheide.org/files/levitator.c
Carrier patching problem

- Carriers are **terrible** at patching
  - Slow, conservative patch practices
  - Inverted user/economic incentives
  - 6+ months typical vulnerability window
- One of the biggest causes of mobile insecurity
  - Carrier's tight grip on control
  - Complex ecosystem of software responsibility
  - Third-parties have no opportunity to intervene
## Carrier patching problem

<table>
<thead>
<tr>
<th>2009 Android</th>
<th>1st Year After Release</th>
<th>2nd Year After Release</th>
<th>3rd Year After Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC MyTouch 3G</td>
<td>8/5/09 T-Mobile $200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTC Hero</td>
<td>10/11/09 Sprint $180*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samsung Moment</td>
<td>11/1/09 Sprint $180*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola Cliq</td>
<td>11/2/09 T-Mobile $200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola Droid</td>
<td>11/6/09 Verizon $200*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTC Droid Eris</td>
<td>11/6/09 Verizon $100*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samsung Behold II</td>
<td>11/18/09 T-Mobile $230</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2010 Q1&amp;2 Android</th>
<th>1st Year After Release</th>
<th>2nd Year After Release</th>
<th>3rd Year After Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC Nexus One</td>
<td>1/5/10 T-Mobile $180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola Devour</td>
<td>2/25/10 Verizon $150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola Backflip</td>
<td>3/7/10 AT&amp;T $100*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola Cliq XT</td>
<td>3/17/10 T-Mobile $130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTC Droid Incredible</td>
<td>4/29/10 Verizon $200*</td>
<td>Possibly still being updated</td>
<td></td>
</tr>
<tr>
<td>LG Ally</td>
<td>5/20/10 Verizon $100*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTC MyTouch 3G Slide</td>
<td>6/2/10 T-Mobile $180*</td>
<td>Still being sold</td>
<td></td>
</tr>
<tr>
<td>HTC Evo 4G</td>
<td>6/4/10 Sprint $200*</td>
<td>Possibly still being updated</td>
<td></td>
</tr>
<tr>
<td>Garmin Garminfone</td>
<td>6/9/10 T-Mobile $200*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTC Aria</td>
<td>6/20/10 AT&amp;T $130*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vulnerability assessment on mobile

- We can't patch the vulns*, can we at least enumerate?
- X-Ray app
  - Vulnerability assessment for Android devices
  - Launched just weeks ago
- Full stats coming next month at United Summit

http://xray.io
A privileged attacker

- If attacker escalates privileges, it's game over, right?
  - Can break out of “sandbox”
  - Tamper with applications
  - Sensitive data can be accessed/stolen

- How can we maintain security guarantees given this threat model?
  - Generally speaking, we can't!
  - But for some use cases...
Agenda

- Introduction
- Gaining code execution
- Escalating privileges
- Recent HSM developments
- Wrap-up
How a HSM helps

- If we're dealing with privileged attacker, assume that all system memory is compromised
  - So, key material must be kept out of memory
- HSM can provide:
  - Smartcard-style crypto engine
  - Hardware-backed tamper-proof credential storage
  - Key generation, signature computation
Past approaches

- SIM-based
  - mSign, etc

- SOC approaches
  - OMAP M-Shield, ARM TrustZones, etc

- Android-specific initiatives
  - Google Wallet, SEEK, etc
Recent Android developments

- Keychain API released in Android 4.0
- Primitives for credential storage

```java
public final class KeyChain
extends Object

java.lang.Object
  android.security.KeyChain

Class Overview

The KeyChain class provides access to private keys and their corresponding certificate chains in credential storage.

Applications accessing the KeyChain normally go through these steps:

1. Receive a callback from an X509KeyManager that a private key is requested.
2. Call choosePrivateKeyAlias to allow the user to select from a list of currently available private keys and corresponding certificate chains.
3. Call getPrivateKey(Context, String) and getCertificateChain(Context, String) to retrieve the credentials.

An application may remember the value of a selected alias to avoid prompting the user with choosePrivateKeyAlias on a private key that is no longer valid, null will be returned on lookups using that value.
```

Recent Android developments

- Further HSM support in Android 4.1
  - keymaster framework implementation
- Galaxy Nexus hardware support
  - OMAP 4, TI M-Shield platform
- Basic HSM crypto operations
  - generate_keypair, import_keypair, sign_data, verify_data, get_keypair_public, delete_keypair, delete_all

Changes in threat model

- With a HSM, attacker can no longer extract sensitive data / key material post-exploitation

- Example:
  - Duo Push private key generated on-device
  - Signatures generated within the HSM
  - Key material never leaves the credential store

Not a silver bullet: attacker can still compute signatures. Need trusted input and trusted display paths.
Why HSM approach?

- Need HSM = failure of system security
  - Similar to some virtualization use cases
- Need for third-party availability
- Not the only feasible approach
  - Other avenues to contain/survive exploitation
  - On-device system containers
  - On-device hardware-level virtualization
  - Threshold crypto techniques?
Agenda

• Introduction
• Gaining code execution
• Escalating privileges
• Recent HSM developments
• Wrap-up
Multiple parties are attacking the mobile security problem at a variety of layers.

HSM plays an important role in device security to disrupt the attack chain even if fully compromised.