Android patching
From a Mobile Device Management perspective

Cedric Van Bockhaven
cbockhaven@os3.nl
System and Network Engineering – RP2
University of Amsterdam

July 2nd, 2014
Outline

Introduction

Background information

Kernel patching

Evaluation

Proof of concept
Introduction

- 80% of smartphones run Android
- Yet only 26% of devices in a BYOD setting run Android
- Different Android versions, ROMs, kernels, hardware

**BYOD** – Bring your own device
Numbers by Joost Kremers [2], TechCrunch [http://goo.gl/FJKHC6](http://goo.gl/FJKHC6), and Fjmustak’s Android version history
Introduction

Cedric Van Bockhaven
July 2nd, 2014
Older Android versions 2.3+ still omnipresent
Responsibility of the vendors to push updates
Many devices remain unpatched and vulnerable

→ Out-of-band update mechanism needed that doesn’t rely on the vendor.
Main research question:

Is it possible to patch security vulnerabilities in Android devices through the MDM?

MDM – Mobile Device Management solution
Related work

Collin Mulliner, Jon Oberheide, William Robertson, and Engin Kirda.
Architecture

- Kernel vulnerabilities
  - \textit{E.g.} Towelroot\textsuperscript{1}

- Framework vulnerabilities
  - \textit{E.g.} Master Key exploit

\textsuperscript{1}Towelroot uses CVE-2014-3153 only, by George Hotz
Runtime hooking

- Available for Dalvik VM
  - DDI toolkit (Dynamic Dalvik Instrumentation) [1]
  - Xposed framework

- No hooks yet for ART
Patching the kernel

- Kernel module
  - Hooking with Kprobes
  - Kernel sources are needed
- Kpatch / Kgraft / Ksplice
  - Easy patch creation with unified diff
  - Kernel sources are needed
- Dynamic patching: *expatting*
  - Universal, cross-device solution
  - Using exploit or other kernel memory access technique
  - Slightly unorthodox

Needed for each vuln and device
Dynamic patching

How to modify kernel memory and hook/patch vulnerable functions?

1. Find the kernel symbols
2. Get read/write access to kernel
3. Conduct patches
1 Finding the kernel symbols

- Read `/proc/kallsyms` or `/proc/ksyms`
  - `kptr_restrict` nullifies kernel pointers `%pK` in user space

- Scanning the memory for the correct addresses
  - Using `/dev/mem`, `/dev/kmem`, or `/proc/kcore`
  - Using exploit to read kernel memory
  - *E.g.* locate `%pK %c %s` and replace with `%p %c %s`
Get read/write access to kernel

- Using exploit or /dev/(k)mem
- `mmap`: map devices or files into memory
- Backdoor original `mmap` system call
- Allows to read/write arbitrary kernel memory from user space
3 Conduct patches

Use the `mmap` backdoor to:

- Hook vulnerable kernel functions in-memory
- Patch Dalvik/ART framework functions as root
Evaluation

- Kernel patches become device independent
  - Still need to make the patch work for different architectures...
  - Quasi all Android devices are ARM
- Tricky: an error can cause kernel panic
  - Needs some fault tolerance
- Expat lives only in memory, non-permanent
  - Gone after reboot
Conclusion

- Patches can be made in a universal way
  - For both the kernel and the runtime
- Basis for an MDM setup to provide patches
DEMO!

- Expat MDM, consists of agent and server module
- Exploiting and patching a kernel vulnerability

Many thanks to Deloitte!
References

Collin Mulliner, Jon Oberheide, William Robertson, and Engin Kirda.

Joost Kremers.
## Appendix: Boot hooking

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>init script</td>
<td>+ cross-platform</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>app_process binary</td>
<td>+ always in the same place</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>broadcast receiver</td>
<td>+ cleanest</td>
</tr>
</tbody>
</table>
Appendix: Exploiting

- *E.g.* PTMX device\(^2\) as stepping stone: `ptmx_fops->fsync`
  - Open `/dev/ptmx` and call `fsync`
- Transfer kernel execution to payload in user space:
  - Use `commit_creds` to run as fully privileged root user

\(^2\)Doesn’t reside in read-only kernel memory