Adaptive Android Kernel
Live Patching

Tim Xia, Yulong Zhang
Baidu X-Lab
May 2016
Outline

• Android Kernel Vulnerability Landscape
• The Problem:
  – Devices Unpatched Forever/for A Long Period
  – Difficult to Patch due to Fragmentation
• The Solution: Adaptive Kernel Live Patching
• Establishing the Ecosystem
Threats of Kernel Vulnerabilities

- Unprivileged User
  - Info-leak Vulnerability
  - Information Leakage
- Root
  - Code Execution Vulnerability
  - Privilege Escalation
Threats of Kernel Vulnerabilities

• Most security mechanisms rely on kernel integrity/trustworthiness, thus will be broken
  – Access control, app/user isolation
  – Payment/fingerprint security
  – KeyStore
  – Other Android user-land security mechanisms

• TrustZone will also be threatened
  – Attack surfaces exposed
  – Many TrustZone logic trusts kernel input
## Kernel Vulnerabilities in Android Security Bulletin

<table>
<thead>
<tr>
<th>Month</th>
<th>Vulnerability List</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015/09</td>
<td>CVE-2015-3636</td>
<td>1</td>
</tr>
<tr>
<td>2015/12</td>
<td>CVE-2015-6619</td>
<td>1</td>
</tr>
<tr>
<td>2016/02</td>
<td>CVE-2016-0801  CVE-2016-0802  CVE-2016-0805  CVE-2016-0806</td>
<td>4</td>
</tr>
<tr>
<td>2016/03</td>
<td>CVE-2016-0728  CVE-2016-0819  CVE-2016-0820  CVE-2016-0823</td>
<td>5</td>
</tr>
</tbody>
</table>

Information Leakage  Privilege Escalation
The Growing Trend Indicates

- More and more attentions are drawn to secure the kernel
  - More and more vulnerabilities are in the N-Day exploit arsenal for the underground businesses
Recent Vulnerabilities with Great Impact

- CVE-2014-3153 (Towelroot)

  - The futex_requeue function in kernel/futex.c in the Linux kernel through 3.14.5 does not ensure that calls have two different futex addresses, which allows local users to gain privileges.
Recent Vulnerabilities with Great Impact

• CVE-2015-3636 (PingPong Root)

  – The ping_unhash function in net/ipv4/ping.c in the Linux kernel before 4.0.3 does not initialize a certain list data structure during an unhash operation, which allows local users to gain privileges or cause a denial of service.
Recent Vulnerabilities with Great Impact

- CVE-2015-1805 (used in KingRoot)
  - The pipe_read and pipe_write implementations in kernel before 3.16 allows local users to cause a denial of service (system crash) or possibly gain privileges via a crafted application.
  - A known issue in the upstream Linux kernel that was fixed in April 2014 but wasn’t called out as a security fix and assigned CVE-2015-1805 until February 2, 2015.
Many Vulnerabilities Have Exploit PoC Publicly Disclosed

<table>
<thead>
<tr>
<th>Vulnerability/Exploit Name</th>
<th>CVE ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mempodipper</td>
<td>CVE-2012-0056</td>
</tr>
<tr>
<td>exynos-abuse/Framaroot</td>
<td>CVE-2012-6422</td>
</tr>
<tr>
<td>diagexploit</td>
<td>CVE-2012-4221</td>
</tr>
<tr>
<td>perf_event_exploit</td>
<td>CVE-2013-2094</td>
</tr>
<tr>
<td>fb_mem_exploit</td>
<td>CVE-2013-2596</td>
</tr>
<tr>
<td>msm_acdb_exploit</td>
<td>CVE-2013-2597</td>
</tr>
<tr>
<td>msm_cameraconfig_exploit</td>
<td>CVE-2013-6123</td>
</tr>
<tr>
<td>get/put_user_exploit</td>
<td>CVE-2013-6282</td>
</tr>
<tr>
<td>futex_exploit/Towelroot</td>
<td>CVE-2014-3153</td>
</tr>
<tr>
<td>msm_vfe_read_exploit</td>
<td>CVE-2014-4321</td>
</tr>
<tr>
<td>pipe exploit</td>
<td>CVE-2015-1805</td>
</tr>
<tr>
<td>PingPong exploit</td>
<td>CVE-2015-3636</td>
</tr>
<tr>
<td>f2fs_exploit</td>
<td>CVE-2015-6619</td>
</tr>
<tr>
<td>prctl_vma_exploit</td>
<td>CVE-2015-6640</td>
</tr>
<tr>
<td>keyring_exploit</td>
<td>CVE-2016-0728</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>
There’re also exploits made public but

• Never got officially reported to vendors
• Disclosed before being patched
• Not getting timely fix
• ......
Exploits made public but not reported

“... We are able to identify at least 10 device driver exploits (from a famous root app) that are never reported in the public...”

*Android Root and its Providers: A Double-Edged Sword*

H. Zhang, D. She, and Z. Qian, CCS 2015
Exploits disclosed but not timely patched

Note that this patch was not applied to all msm branches at the time of the patch release (July 2015) and no security bulletin was issued, so the majority of Android kernels based on 3.4 or 3.10 are still affected despite the patch being available for 6 months.

https://bugs.chromium.org/p/project-zero/issues/detail?id=734&can=1&sort=-id
Malware/Adware with Root Exploits

GHOSTPUSH

More than 30+ apps (WiFi Enhancer, Talking Tom 3 etc.) infected by the virus

Some app stores (not including Google Play) popular download sites

Over 600,000 phones are being infected per day

Virus installs itself deeply in the phone

This virus has become worldwide: 3,658 brands and 14,846 types of phone have been infected

Root your phone, and install the virus to your ROM

‘Ghost Push’ will consume your cellular data by turning off your Wi-Fi connection and then downloading lots of ads and unwanted apps

Virus will autostart with the phone and is hard to remove

The hackers are looking to make money from these ads and apps
Malware/Adware with Root Exploits

“This is the first time, to my knowledge; an exploit kit has been able to successfully install malicious apps on a mobile device without any user interaction on the part of the victim... the payload of that exploit, a Linux ELF executable named module.so, contains the code for the futex or Towelroot exploit that was first disclosed at the end of 2014.”
Outline

• Android Kernel Vulnerability Landscape

• The Problem
  – Devices Unpatched Forever/for A Long Period
  – Difficult to Patch due to Fragmentation

• The Solution: Adaptive Kernel Live Patching

• Establishing the Ecosystem
iOS More Secure?
Kernel Vulnerability Disclosure
Frequency Is Comparable

<table>
<thead>
<tr>
<th>iOS Version</th>
<th>Date</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4.1</td>
<td>8/13/15</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>9/16/15</td>
<td>12</td>
</tr>
<tr>
<td>9.1</td>
<td>10/21/15</td>
<td>6</td>
</tr>
<tr>
<td>9.2</td>
<td>12/8/15</td>
<td>5</td>
</tr>
<tr>
<td>9.2.1</td>
<td>1/19/16</td>
<td>4</td>
</tr>
<tr>
<td>9.3</td>
<td>3/21/16</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Month</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015/09</td>
<td>1</td>
</tr>
<tr>
<td>2015/12</td>
<td>1</td>
</tr>
<tr>
<td>2016/01</td>
<td>4</td>
</tr>
<tr>
<td>2016/02</td>
<td>4</td>
</tr>
<tr>
<td>2016/03</td>
<td>5</td>
</tr>
<tr>
<td>2016/04</td>
<td>7</td>
</tr>
<tr>
<td>2016/05</td>
<td>15</td>
</tr>
</tbody>
</table>
However...

• If Apple wants to patch a vulnerability
  – Apple controls the entire (mostly) supply chain
  – Apple has the source code
  – Apple refuses to sign old versions, forcing one-direction upgrade
  – All the iOS devices will get update in a timely manner

• Android
  – Many devices stay unpatched forever/for a long period...
Devices Unpatched Forever/for A Long Period

• Cause A: The long patching chain

- Researchers found the vulnerability
- Hardware vendors/Google finalized the patch
- Phone vendors tested and took the patch
- Carriers tested and approved the patch
- Customer delays or unwilling to take the OTA
# Device Fragmentation

Google Dashboard (2016/04/04)

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>8</td>
<td>0.1%</td>
</tr>
<tr>
<td>2.3.x</td>
<td>Gingerbread</td>
<td>10</td>
<td>2.6%</td>
</tr>
<tr>
<td>4.0.x</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>2.2%</td>
</tr>
<tr>
<td>4.1.x</td>
<td></td>
<td>16</td>
<td>7.8%</td>
</tr>
<tr>
<td>4.2.x</td>
<td>Jelly Bean</td>
<td>17</td>
<td>10.5%</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>18</td>
<td>3.0%</td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>19</td>
<td>33.4%</td>
</tr>
<tr>
<td>5.0</td>
<td>Lollipop</td>
<td>21</td>
<td>16.4%</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>22</td>
<td>19.4%</td>
</tr>
<tr>
<td>6.0</td>
<td>Marshmallow</td>
<td>23</td>
<td>4.6%</td>
</tr>
</tbody>
</table>

Lollipop was released in November 12, 2014, but **60%** of the devices are still older than that!

Google stopped patching for Android older than 4.4, but **26.2%** of the devices are still older than that!
# Chinese Market Is Even Worse

(Stats from devices with Baidu apps installed, 03/21/2016-04/21/2016)

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.x</td>
<td>Gingerbread</td>
<td>10</td>
<td>3.2%</td>
</tr>
<tr>
<td>4.0.x</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>3.6%</td>
</tr>
<tr>
<td>4.1.x</td>
<td>Jelly Bean</td>
<td>16</td>
<td>7.6%</td>
</tr>
<tr>
<td>4.2.x</td>
<td></td>
<td>17</td>
<td>12.4%</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>18</td>
<td>13.6%</td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>19</td>
<td>42.4%</td>
</tr>
<tr>
<td>5</td>
<td>Lollipop</td>
<td>21</td>
<td>9.8%</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>22</td>
<td>6.6%</td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td>-</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

Lollipop was released in November 12, 2014, but **82.8%** of the devices are still older than that!

**40.4%** of the devices are <4.4! And China **blocks** Google....
Devices with Unpatched Kernels
(Stats from devices with Baidu apps installed, May 2016)

• CVE-2014-3153 (Towelroot)
  – Advisory/Patch Publication Date: \textbf{Jun. 3rd, 2014}
  – Device distribution with kernel build date older/newer than the date:

- Older: 45%
- Newer: 55%
Devices with Unpatched Kernels
(Stats from devices with Baidu apps installed, May 2016)

• CVE-2015-3636 (PingPong Root)
  – Advisory/Patch Publication Date: Sep. 9th, 2015
  – Device distribution with kernel build date older/newer than the date:
Devices with Unpatched Kernels
(Stats from devices with Baidu apps installed, May 2016)

• CVE-2015-1805 (used in KingRoot)
  – Advisory/Patch Publication Date: Mar. 18th, 2016
  – Device distribution with kernel build date older/newer than the date:
Devices Unpatched Forever/for A Long Period

• Cause B: Fragmentation & Capability Miss-matching

Phone Vendors:
• Privileged to apply the patches
• With source code, easy to adapt the patches
• Not enough resources to discover and patch vulnerabilities

Security Vendors:
• Capable to discover and patch vulnerabilities
• Not privileged enough
• Without source code, difficult to adapt the patches
My first priority is not on vulnerability discovery and real-world exploits...

So challenging to protect the world...

I’ve tried my best...

Image sources:
https://d.gr-assets.com/hostedimages/1417789603ra/12537314.gif
http://1.bp.blogspot.com/-InMpoEJ4zgk/TknyHEBtD4I/AAAAAAAACRY/6ogSBIPjFWI/s1600/obama%2Bsweats.jpg
How/Who to Secure Them???
Outline

• Android Kernel Vulnerability Landscape
• The Problem:
  – Devices Unpatched Forever/for A Long Period
  – Difficult to Patch due to Fragmentation
• The Solution: Adaptive Kernel Live Patching
• Establishing the Ecosystem
Kernel Live Patching

kGraft as an example
Kernel Live Patching

• Load new functions into memory
• Link new functions into kernel
  – Allows access to unexported kernel symbols
• Activeness safety check
  – Prevent old & new functions from running at same time
  – stop_machine() + stack backtrace checks
• Patch it!
  – Uses ftrace etc.
Challenges for Third Party

• Most existing work requires source code
  – Phone vendor is the only guy that can generate the live patches

• Unable to directly apply patches to other kernel builds
  – Load code into kernel adaptively
Our Solution - Adaptive Live Patching

Auto patch adaption
- Kernel info gathering
- Data structure filling

Patching payload injection
- Install kernel module
- Shellcode injection via mem device

Patching payload execution
- Replace/hook vulnerable functions
Kernel Info Collection

• Kernel version
  – /proc/version
  – vermagic

• Symbol addresses/CRC
  – /proc/kallsyms (/proc/sys/kernel/kptr_restrict)

• Other kernel modules
  – Symbol CRC/module init offset

• Boot image
  – decompress gzip/bzip/lzma/lzo/xz/lz4
  – some are raw code or even ELF file
Patching payload injection
Device Coverage

INSMOD 95%
(K)MEM 60%

99.4% 0.6%
Method A: Kernel Module Injection

- **init_module**
  - CONFIG_MODVERSIONS
  - CONFIG_MODULE_FORCE_LOAD
- **finit_module**
  - Linux 3.8+
  - MODULE_INIT_IGNORE_MODVERSIONS
  - MODULE_INIT_IGNORE_VERMAGIC
- **restrictions**
  - vermagic check
  - symbol CRC check
  - module structure check
  - vendor’s specific check
    - Samsung lkmauth
Bypass vermagic/symbol CRC

- Big enough vermagic buffer
- Copy kernel vermagic string to module
- Copy kernel symbol CRCs to module

```c
#include/linux/vermagic.h
#define VERMAGIC_STRING
#define UTS_RELEASE " "
#define MODULE_VERMAGIC_SMP MODULE_VERMAGIC_PREEMPT
#define MODULE_VERMAGIC_MODULE_UNLOAD MODULE_VERMAGIC_MODVERSIONS
#define MODULE_ARCH_VERMAGIC

#define VERMAGIC_STRING "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=YAY!"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
```
Bypass module structure

- offsetof(init) difference
- Big enough struct module

Code snippet:

```c
#include <linux/module.h>
struct module {
...
    /* Startup function. */
    int (*init)(void);
...
    #ifdef CONFIG_CONSTRUCTORS
        /* Constructor functions. */
        ctor_fn_t *ctors;
        unsigned int num_ctors;
    #endif
    int padding[XX];
...}
```
Bypass Samsung lkmauth1
Bypass Samsung lkmauth2

```assembly
LDR    R3, [R8,#4]
CMP    R3, #0  ; make lkmauth_bootmode=BOOTMODE_RECOVERY to skip
            skip_lkmauth
MOV    R0, #0xC094ACA4 ; <4>TIMA: lkmauth--verification succeeded
            printk
BL     R0, =lkmauth_mutex
       mutex_unlock
LDR    R5, [R4,#0x20]

#define BOOTMODE_RECOVERY 2
```
Method B: Shellcode Injection

- Symbol addresses
  - vmalloc_exec
  - module_alloc
- Structured shellcode
- Allocate/reuse memory
- Write into memory
- Trigger the running

```c
struct shell_code_binary {
    unsigned long magic;
    unsigned long version;
    unsigned long header_size;
    unsigned long shellcode_size;
    unsigned long shellcode_entry;
    unsigned long lookup_name_offset;
    unsigned long mmap_ram_start_offset;
    unsigned long mmap_ram_end_offset;
    unsigned long vuln_count_offset;
    unsigned long vuln_ids_offset;
    unsigned long current_pid_offset;
    unsigned long kmem_write_count;
    unsigned long patch_count;
    unsigned long* write_offset_array;
    unsigned long* patch_ids_array;
    unsigned long* patch_offset_array;
    unsigned char* shellcode_body;
};
```
Memory Allocation

```
STMFD    SP!, {R3,LR}
LDR      R0, loc_4C; //size
LDR      R3, loc_4C;
BLX      R3  //vmalloc_exec
LDMFD    SP!, {R3,PC}
size
vmalloc_exec_addr
```

Diagram:
- User
- Kernel
- System Call Table
- System Call Pointer
- System Call Function
- Shellcode to Allocate Memory
Shellcode Execution
Patching Payload Execution

• Overwrite the function pointer
  – with our own implementation

• Overwrite with patch code directly
  – Need permission, CP15 to help

• Inline hook
  – Atomic with best effort
  – Hook from prolog
  – Hook from middle of the function
    • Need save some context
Vulnerable Function Hook

STM FD SP!, {R3, LR} → LDR/B Patch → Patch Func

RET ADDR

STM FD SP!, {R3, LR} → LDR PC, [PC, #-4] → RET ADDR

Patch Func
Vulnerable Function Hook (cont.)

- The patch has the option to execute the original function or just do not

- No option if patch hook from the middle of the vulnerable function

- Painful in 64bit, no explicit operation on PC
Optimizations

• Utilizing kallsyms_lookup_name
  – minimize the symbols imported
• Utilizing existing kernel mem write functions
  – mem_text_write_kernel_word
  – set_memory_rw
• CP15 to change permission
Challenges Solved

• No source code & fragmentation problem solved
  ➢ Patch automatic adaption
Challenges Solved

✔ Most existing work requires source code
  – Phone vendor is the only guy that can generate the live patches

✔ Unable to directly apply patches to other kernel builds
  – Load code into kernel adaptively

[Diagram showing a transition from vulnerable to immutable]
Successfully Evaluated CVEs

- mmap CVEs (Framaroot)
- CVE-2014-3153 (Towelroot)
- CVE-2015-0569
- CVE-2015-1805
- CVE-2015-3636 (PingPong Root)
- CVE-2015-6640
- CVE-2016-0728
- CVE-2016-0805
- CVE-2016-0819
- CVE-2016-0844
- ......
Successfully Evaluated on Most Popular Phones

GT-I8552  GT-S7572  S4  A7  SM-G5308W

Grand 2  Note 4
Successfully Evaluated on Most Popular Phones

C8813
P6-U06
Hornor
U8825D
Successfully Evaluated on Most Popular Phones

M7

M8Sw

S720e

T528d
Successfully Evaluated on Most Popular Phones

A630t  A788t  A938t  K30-T
Successfully Evaluated on Most Popular Phones

![Logos of various phone brands]
Demo

Before Patch: **PingPong** Root succeed

After Patch: **PingPong** Root fail

Samsung S4
Outline

• Android Kernel Vulnerability Landscape
• The Problem
  – Devices Unpatched Forever/for A Long Period
  – Difficult to Patch due to Fragmentation
• The Solution: Adaptive Kernel Live Patching
• Establishing the Ecosystem
Recall the Two Problems

- The long patching chain
  - Solved by adaptive live patching
- Capability miss-matching
  - To be solved by a joint-effort
Incentives

• Vendors
  – More secure products
  – More users & sales

• Security Providers
  – Reputation
  – profits
Transition to Cooperative Patching

- Exploit existing vulnerabilities to gain root
- Vendor cooperation & pre-embedded kernel agent
Establishing the Ecosystem

Open Cooperative Patching Ecosystem

- Vendor qualification
- Signature based patch distribution
- Security vetting procedure
- Reputation ranking
To Be Announced

• Ecosystem alliance
• Flexible & easy-to-review patching mechanism
Thanks!

Tim Xia, Longri Zheng, Yongqiang Lu, Chenfu Bao, Yulong Zhang, Lenx Wei
Baidu X-Lab
May 2016