Adaptive Kernel Live Patching:
An Open Collaborative Effort to
Ameliorate Android N-day Root Exploits

Yulong Zhang and Lenx (Tao) Wei
Baidu X-Lab
August 2016
Agenda

• The Problem
  • Android Kernel Vulnerability Landscape
  • Why Are They Long-lasting?
  • Case Studies

• The Solution
  • AdaptKpatch: Adaptive Kernel Live Patching
  • LuaKpatch: More Flexibility, Yet More Constraint

• The Future
  • Establishing the Ecosystem
Threats of Kernel Vulnerabilities

- Unprivileged User
  - Information Leakage
  - Info-leak Vulnerability
- Root
  - Privilege Escalation
  - Code Execution Vulnerability

User Mode vs. Kernel Mode
Threats of Kernel Vulnerabilities

• Most security mechanisms relying on kernel integrity/trustworthiness 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
  • Not enough input validation
Kernel Vulnerabilities in Android Security Bulletin

Monthly Disclosed Number of Android Kernel Vulnerabilities

<table>
<thead>
<tr>
<th>Year/Month</th>
<th>Number</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>3</td>
</tr>
<tr>
<td>2016/02</td>
<td>4</td>
</tr>
<tr>
<td>2016/03</td>
<td>4</td>
</tr>
<tr>
<td>2016/04</td>
<td>7</td>
</tr>
<tr>
<td>2016/05</td>
<td>15</td>
</tr>
<tr>
<td>2016/06</td>
<td>19</td>
</tr>
<tr>
<td>2016/07</td>
<td>66</td>
</tr>
</tbody>
</table>
The Growing Trend Indicates

<table>
<thead>
<tr>
<th>Month</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015/09</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2015/12</td>
<td>1</td>
</tr>
<tr>
<td>2016/01</td>
<td>3</td>
</tr>
<tr>
<td>2016/02</td>
<td>4</td>
</tr>
<tr>
<td>2016/03</td>
<td>4</td>
</tr>
<tr>
<td>2016/04</td>
<td>7</td>
</tr>
<tr>
<td>2016/05</td>
<td>15</td>
</tr>
<tr>
<td>2016/06</td>
<td>19</td>
</tr>
<tr>
<td>2016/07</td>
<td>66</td>
</tr>
</tbody>
</table>

• 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
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>Ping Pong Root</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>
KEMOGE

Infection Vectors
1. Third Party App Stores
2. Web/Ad Promoted Installation
3. Drop 8 root exploits to root the device

Initial Behaviors
2. Local info collection
3. Aggressive ad serving

Persistent Behaviors
4. Remote Control:
   - Install any app
   - Uninstall any app
   - Launch any app

CN C Server

https://www.fireeye.com/blog/threat-research/2015/10/kemoge_another_mobi.html
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

Virus will autostart with the phone and is hard to remove

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

The hackers are looking to make money from these ads and apps

"... 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."
“All combined, the campaign includes nearly 85 million devices...
HummingBad attempts to gain root access on a device with a rootkit that
exploits multiple vulnerabilities... It tries to root thousands of devices
every day, with hundreds of these attempts successful.”
iOS More Secure?
<table>
<thead>
<tr>
<th>iOS Version</th>
<th>Release Date</th>
<th>Kernel Vulnerability #</th>
<th>Android # In This Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4.1</td>
<td>8/13/15</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>9/16/15</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>9.1</td>
<td>10/21/15</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>9.2</td>
<td>12/8/15</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9.2.1</td>
<td>1/19/16</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>9.3</td>
<td>3/21/16</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>9.3.2</td>
<td>5/16/16</td>
<td>11</td>
<td>22</td>
</tr>
</tbody>
</table>
So the problem is: **Android has MORE vulnerabilities**

Vulnerabilities remain **UNFIXED** over a long time
Agenda

• The Problem
  • Android Kernel Vulnerability Landscape
  • Why Are They Long-lasting?
  • Case Studies

• The Solution
  • AdaptKpatch: Adaptive Kernel Live Patching
  • LuaKpatch: More Flexibility, Yet More Constraint

• The Future
  • Establishing the Ecosystem
• 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...
Why Are Android Kernel Vulnerabilities Long Lasting?

- The long patching chain delays the patch effective date
- Fragmentation makes it challenging to adapt the patches to all devices
- Capability mismatching between device vendors and security vendors
Cause A: The long patching chain

1. Researchers found the vulnerability
2. Hardware vendors/Google finalized the patch
3. Phone vendors tested and took the patch
4. Carriers tested and approved the patch
5. Customer delays or unwilling to take the OTA
There are exploits appeared in public but

- Never got officially reported to vendors
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
There are exploits disclosed but

- Not getting timely patches
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
There are exploits patched but
- Delayed by the carriers
Exploits patched but delayed by carriers

“... It’s each carrier’s job to test all the different updates for all their different smartphones, and they may take many months to do so. They may even decline to do the work and never release the update...”

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

User delays the OTA due to rebooting
Why Are Android Kernel Vulnerabilities Long Lasting?

• The long patching chain delays the patch effective date
• Fragmentation makes it challenging to adapt the patches to all devices
• Capability mismatching between device vendors and security vendors
Cause B: Fragmentation
Lollipop was released in November 12, 2014, but **51.6%** of the devices are still older than that!

Google stopped patching for Android older than 4.4, but **21.5%** of the devices are still older than that!

<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>1.9%</td>
</tr>
<tr>
<td>4.0.x</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>1.7%</td>
</tr>
<tr>
<td>4.1.x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.x</td>
<td>Jelly Bean</td>
<td>17</td>
<td>8.8%</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>18</td>
<td>2.6%</td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>19</td>
<td>30.1%</td>
</tr>
<tr>
<td>5.0</td>
<td>Lollipop</td>
<td>21</td>
<td>14.3%</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>22</td>
<td>20.8%</td>
</tr>
<tr>
<td>6.0</td>
<td>Marshmallow</td>
<td>23</td>
<td>13.3%</td>
</tr>
</tbody>
</table>
Chinese Market Is Even Worse
(Stats from devices with Baidu apps installed, July 2016)

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.x</td>
<td>Gingerbread</td>
<td>3%</td>
</tr>
<tr>
<td>4.0.x</td>
<td>Ice Cream Sandwich</td>
<td>3%</td>
</tr>
<tr>
<td>4.1.x</td>
<td>Jelly Bean</td>
<td>36%</td>
</tr>
<tr>
<td>4.2.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>39%</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Lollipop</td>
<td>19%</td>
</tr>
</tbody>
</table>

Lollipop was released in November 12, 2014, but **80%** of the devices are still older than that!

**42%** of the devices are <4.4!
Why Are Android Kernel Vulnerabilities Long Lasting?

• The long patching chain delays the patch effective date
• Fragmentation makes it challenging to adapt the patches to all devices
• Capability mismatching between device vendors and security vendors
Security Vendors:
- Capable to discover and patch vulnerabilities
- Not privileged enough
- Without source code, difficult to adapt the patches

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

• The Problem
  • Android Kernel Vulnerability Landscape
  • Why Are They Long-lasting?
  • Case Studies

• The Solution
  • AdaptKpatch: Adaptive Kernel Live Patching
  • LuaKpatch: More Flexibility, Yet More Constraint

• The Future
  • Establishing the Ecosystem
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.
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.
• 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.
Days since the advisory publication date

- CVE-2014-3153
  - Towelroot
- CVE-2015-3636
  - Ping Pong Root
- CVE-2015-1805
  - Pipe Root
Vulnerability statistics collected from Chinese Android device in July 2016
How/Who to Secure Them???
Agenda

• The Problem
  • Android Kernel Vulnerability Landscape
  • Why Are They Long-lasting?
  • Case Studies

• The Solution
  • AdaptKpatch: Adaptive Kernel Live Patching
  • LuaKpatch: More Flexibility, Yet More Constraint

• The Future
  • Establishing the Ecosystem
Kernel Live Patching

- kpatch
- kGraft
- ksplice
- Linux upstream’s livepatch
- ......
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
AdaptKpatch - Adaptive Live Patching

Auto patch adaption
- Kernel info gathering
- Data structure filling

Patching payload injection
- Choice A: Install kernel module
- Choice B: Binary code 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
Patch Injection Methods Coverage

- INSMOD 95%
- (K)MEM 60%

Total Coverage: 99.4%

Uncovered: 0.6%
Method A: Kernel Module Injection

Kernel checks that need to be resolved for adaptation

- 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
Bypass Samsung lkmauth

```
.text:C00C7718
.text:C00C7718 8C 32 9F E5
.text:C00C771C F0 4F 2D E9
.text:C00C7720 54 D0 4D E2
.text:C00C7724 54 42 9F E5
.text:C00C7728 01 A0 A0 E1
.text:C00C772C 00 90 A0 E1
.text:C00C7730 7C 02 0A E1
.text:C00C7734 00 30 93 E5
.text:C00C7738 4C 30 8D E5
.text:C00C773C 16 FC 1E EB
.text:C00C7740 0A 10 A0 E1
.text:C00C7744 6C 02 9F E5
.text:C00C7748 E6 CA 1E EB
.text:C00C774C 2C 00 8D E2
.text:C00C7750 64 12 9F E5
.text:C00C7754 9A 8C 08 EB
.text:C00C7758 44 11 98 E5
.text:C00C7780 00 90 51 E3
.text:C00C7784 02 00 00 1A
.text:C00C7788 54 01 9F E5
.text:C00C778C 97 CA 1E EB
.text:C00C778E 3C 00 00 EA

EXPORT lkmauth

LDR R3, =__stack_chk_guard
STMFD SP!, {R4-R11,LR}
SUB SP, SP, #0x54
LDR R4, =0xC1254B04
MOV R10, R1
MOV R9, R0
LDR R0, =lkmauth_mutex
LDR R3, [R3]
STR R3, [SP,#0x78+var_2C]
BL mutex_lock
MOV R1, R10
LDR R0, =0xC0CC09D3
BL printk
ADD R0, SP, #0x78+var_4C
LDR R1, =aTima_lkm    ;"tima_lkm"
BL strcpy

LDR R1, [R8,#0x144]
CMP R1, #0
BNE lkmauth_failed    // BNE => NOP

lkmauth_failed
LKMAUTH_FAIL

LDR R0, =0xC0CC0C0B
BL printk
B lkmauth_pass
```
Method B: mem/kmem Injection

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

- Overwrite the function pointer
- Overwrite with patch code directly
- Inline hook

Same with other live patching methods
Adaption Challenges 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.
Successfully Evaluated CVEs

- mmap CVEs ➔ Framaroot
- CVE-2014-3153 ➔ Towelroot
- CVE-2015-0569 ➔ Pipe Root
- CVE-2015-1805 ➔ Ping Pong Root
- CVE-2015-3636
- 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
Demo

Before Patch: **Ping Pong Root** succeed

After Patch: **Ping Pong Root** fail
Recall the Two Problems

• The long patching chain
  • Solved by adaptive live patching

• Capability mismatching
  • To be solved by a joint-effort
Exploit existing vulnerabilities to gain root

Vendor cooperation & pre-embedded kernel agent
Cloud

- Lists of vulnerabilities for devices
- Patches

Phone

- User-space Agent
  - SSL
- User Space
  - SELinux Enforcement
- Kernel-space Patching Module

- Display the list of vulnerabilities and interact with the user
- Patch management
- Patch verification
- Patch adaption
- Patch execution & undo
- Status monitoring
Multi-stage Vetting Mechanism

Vendor qualification

Patch security vetting

Reputation ranking
Agenda

• The Problem
  • Android Kernel Vulnerability Landscape
  • Why Are They Long-lasting?
  • Case Studies

• The Solution
  • AdaptKpatch: Adaptive Kernel Live Patching
    • LuaKpatch: More Flexibility, Yet More Constraint

• The Future
  • Establishing the Ecosystem
We need a patching mechanism

- powerful enough to block most threats;
- agile enough for quick patch generation;
- yet restrictive enough to confine possible damages caused by the patches.
Our Solution -- LuaKpatch

Inserting a type-safe dynamic language engine (Lua) into the kernel to execute patches

- Easy to update
- Naturally jailed in the language VM
- No need to worry about memory overflow etc. of the patches
By hooking the data input entries and validating the input, we can block most of the kernel exploits.
So we have the following restrictions

1) The patch can hook a target function’s entry;

2) In combination with 1), within the target function, the patch can hook the invoking point or returning point of functions that return a status code (e.g., `copy_from_user`);

3) The patch can read anything that can be read (registers, stacks, heaps, code, etc., as long as it does not trigger faults), but cannot modify original kernel memory (no write, and no data can be sent out);

4) After judging whether the input is malicious or not, the patch can return specific error codes.
fun(...) {
   // entry of A can be hooked
   bool result;
   struct *s;

   // foo is allowed to be hooked
   result = foo(...);
   if (result == E_INVALID)
      return;

   // bar cannot be hooked
   s = bar(...);
   if (s)
      s->fun();
}
Implementation of LuaKpatch

• Many practices followed from the lunatik-ng project.
• Line-of-Code (LoC) is ~11K. 600 LoC are the core patching logic.
• Compiled as a 800KB kernel module.
• Capability interfaces:
  o Symbol searching
  o Hooking
  o Typed reading
  o Thread info fetching
Sample Lua patch to fix one of the vulnerable conditions of CVE-2014-3153, known as “Towelroot”
Efficacy Evaluation

<table>
<thead>
<tr>
<th>CVE-2012-4220</th>
<th>CVE-2013-6123</th>
<th>CVE-2015-3636</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2012-4221</td>
<td>CVE-2013-6282</td>
<td>CVE-2015-6619</td>
</tr>
<tr>
<td>CVE-2012-4222</td>
<td>CVE-2014-3153</td>
<td>CVE-2015-6640</td>
</tr>
<tr>
<td>CVE-2013-1763</td>
<td>CVE-2014-4321</td>
<td>CVE-2016-0728</td>
</tr>
<tr>
<td>CVE-2013-2094</td>
<td>CVE-2014-4322</td>
<td>CVE-2016-0774</td>
</tr>
<tr>
<td>CVE-2013-2596</td>
<td>CVE-2015-0569</td>
<td>CVE-2016-0802</td>
</tr>
<tr>
<td>CVE-2013-2597</td>
<td>CVE-2015-1805</td>
<td>CVE-2016-2468</td>
</tr>
</tbody>
</table>

CVEs verified to be protectable by LuaKpatch. Most are Type I vulnerabilities (those that can be patched by simply hooking the entry of the vulnerable functions), but the highlighted/colored ones are Type II vulnerabilities (those that also need to hook the invocations that return status code).
All 21 CVEs can be patched by LuaKpatch. 16 are Type I, and 5 are Type II. So 76% of them can be easily fixed by hooking and checking input at the function entry.
Example I (CVE-2013-1763)

diff --git a/net/core/sock_diag.c b/net/core/sock_diag.c
index 602cd63..750f44f 100644
--- a/net/core/sock_diag.c
+++ b/net/core/sock_diag.c
@@ -121,6 +121,9 @@
     static int __sock_diag_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
         if (nlmsg_len(nlh) < sizeof(*req))
             return -EINVAL;
+
         if (req->sdiag_family >= AF_MAX)
             return -EINVAL;
+
         hndl = sock_diag_lock_handler(req->sdiag_family);
         if (hndl == NULL)
             err = -ENOENT;

LuaKpatch can patch it by hooking the entry of the `__sock_diag_rcv_msg` function, getting the `nlh` argument, obtaining `req` from `nlh`, and then checking whether the condition `req->sdiag_family >= AF_MAX` is satisfied. If this is true, it is an exploit condition and the patch should return an error.
LuaKpatch can patch it by hooking the returning point of the `copy_from_user` invoked by `msm_ioctl_server` to check the exploit condition.
Demo

Before Patch: Vulnerable to Towelroot and Ping Pong Root

After Patch: Immune to Towelroot and Ping Pong Root
Performance Evaluation

<table>
<thead>
<tr>
<th>CF-Bench Performance Score</th>
<th>17473.25</th>
<th>17551.75</th>
<th>17521.4</th>
<th>17482</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patched (Towelroot)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patched (Ping Pong Root)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patched (both vulnerabilities)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Execution Time of chmod (Microseconds)

- No patch: 100.7 µs
- Patched with a direct return: +0.42 µs
- Patched with a conditional comparison: +0.98 µs
- Patched with a memory read: +0.82 µs
- Patched with mixed operations: +3.74 µs
LuaKpatch validation check adds an overhead under 4 microseconds, only 4% of a chmod system call.

Because system calls are not invoked all the time, the impact to the overall system performance should be even less.

- When a user normally browses Internet using Chrome on Nexus 5 + Android 4.4, `gettimeofday` was the mostly-called system call, triggered for ~110,000 times. The overall performance overhead can be estimated as $5\mu s \times 110,000/1\text{min} \approx 0.9\%$, which is quite small.
As an ongoing work, we are migrating LuaKpatch to LuaJIT, which should further improve the performance.
Agenda

• The Problem
  • Android Kernel Vulnerability Landscape
  • Why Are They Long-lasting?
  • Case Studies

• The Solution
  • AdaptKpatch: Adaptive Kernel Live Patching
  • LuaKpatch: More Flexibility, Yet More Constraint

• The Future
  • Establishing the Ecosystem
The patching circle in the open collaborative patching ecosystem
Let’s fight the bad together!

• The number and the complexity of kernel vulnerabilities keep increasing, so more joint effort makes it easier to battle against them.

• In the AdaptKpatch scheme, patches can be vetted and cross-validated by qualified alliance members.

• Last but most importantly, all vendors can join together to develop a patching standard instead of implementing different variants. If different hot patching mechanisms exist, it introduces another layer of fragmentation.
Thanks!

Yulong Zhang, Yue Chen, Chenfu Bao, Liangzhao Xia, Longri Zheng, Yongqiang Lu, Lenx Wei
Baidu X-Lab
August 2016