Adaptive Android Kernel Live Patching

Yue Chen¹, Yulong Zhang², Zhi Wang¹, Liangzhao Xia², Chenfu Bao², Tao Wei²
Florida State University¹
Baidu X-Lab²
Android Kernel Vulnerabilities

- Apps
- Java API Framework
- Native C/C++ Libraries
- Android Runtime
- Hardware Abstraction Layer
- Linux Kernel
- TrustZone
Android Kernel Vulnerabilities

- Apps
- Java API Framework
- Native C/C++ Libraries
- Hardware Abstraction Layer
- Linux Kernel
- Android Runtime
- TrustZone
Number of Disclosed Android Kernel Vulnerabilities
Problem: Old Exploits Remain Effective

<table>
<thead>
<tr>
<th>CVE ID</th>
<th>Release Date</th>
<th>Months</th>
<th>% Vulnerable Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2015-3636</td>
<td>Sep. 2015</td>
<td>14</td>
<td>30%</td>
</tr>
<tr>
<td>CVE-2015-1805</td>
<td>Mar. 2016</td>
<td>8</td>
<td>47%</td>
</tr>
</tbody>
</table>

Number of devices vulnerable to two root exploits as of Nov. 2016

- Android 5.0 released in November 2014
- **46.3%** of devices run an older version in September 2016
Challenges

- *Officially* patching an Android device is a long process → *Third-party*

- *Delayed/non-existing* kernel source code → *Binary-based*
Challenges

- Severely **fragmented** Android ecosystem → Adaptive
Solution

Third-party Binary-based Adaptive Kernel Live Patching

Key requirements:
• **Adaptiveness**
  – It should be adaptive to various device kernels
• **Safety**
  – Patches should be easy to audit
  – Their behaviors must be technically confined
• **Timeliness**
  – Response time should be short, after disclosed vulnerability or exploit
• **Performance**
  – The solution should not incur non-trivial performance overhead
Feasibility Study: Dataset

- Studied **1139** Android kernels

<table>
<thead>
<tr>
<th>Vendor</th>
<th>#Models</th>
<th>#Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung</td>
<td>192</td>
<td>419</td>
</tr>
<tr>
<td>Huawei</td>
<td>132</td>
<td>217</td>
</tr>
<tr>
<td>LG</td>
<td>120</td>
<td>239</td>
</tr>
<tr>
<td>Oppo</td>
<td>74</td>
<td>249</td>
</tr>
<tr>
<td>Google Nexus</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>520</strong></td>
<td><strong>1139</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Countries</td>
<td>67</td>
</tr>
<tr>
<td>Carriers</td>
<td>37</td>
</tr>
<tr>
<td>Android Versions</td>
<td>4.2.x, 4.3.x, 4.4.x, 5.0.x, 5.1.x, 6.0.x, 7.0.x</td>
</tr>
<tr>
<td>Kernel Versions</td>
<td>2.6.x, 3.0.x, 3.4.x, 3.10.x, 3.18.x</td>
</tr>
<tr>
<td>Kernel Architectures</td>
<td>ARM (77%), AArch64 (23%)</td>
</tr>
</tbody>
</table>
Feasibility Study: Observations

- Most kernel functions are **stable** across devices and Android releases
- Most vulnerabilities triggered by **malicious inputs**
- Many functions return **error codes**
  - Return a pointer $\rightarrow$ ERR_PTR
Feasibility Study: Observations

- Most kernel functions are **stable** across devices and Android releases.
- Most vulnerabilities triggered by **malicious inputs**.
- Many functions return **error codes**
  - Return a pointer → ERR_PTR

Gracefully return

Filter them
Overall Approach: Input Validation
KARMA: Kernel Adaptive Repair for Many Androids

- **Adaptive** – Automatically adapt to various device kernels
- **Memory-safe** – Protect kernel from malicious (misused) patches
- **Multi-level** – Flexible for different vulnerabilities
KARMA Design: Safety

- Patches are written in Lua, confined by Lua VM at runtime
- A patch can only be placed at designated locations
- Patched functions must return error codes or void
  - Use existing error handling to recover from attacks
- A patch can read but not write the kernel memory
  - Confined by KARMA APIs
  - Prevent malicious (misused) patches from changing the kernel
  - Prevent information leakage
KARMA Design: Multi-level Patching

• A patch can only be placed at designated locations
  Level 1: Entry or return point of a (vulnerable) function
  Level 2: Before or after the call site to a callee
    e.g., copy_from_user
  Level 3: Binary-based patch

• 76 critical Android kernel vulnerabilities
  Level 1: 49/76 (64.5%)
  Level 2: 22/76 (28.9%)
  Level 3: 5/76 (6.6%)
KARMA Patch Example

```c
if (requeue_pi) {
    /*
    * Requeue PI only works on two distinct uaddrs. This
    * check is only valid for private futexes. See below.
    */
    if (uaddr1 == uaddr2)
        return -EINVAL;

    /*
    * requeue_pi requires a pi_state, try to allocate it now
    * without any locks in case it fails.
    */
}
```

Part of the official patch of CVE-2014-3153 (Towelroot)
```python
function kpatcher(patchID, sp, cpsr, r0, r1, 
    r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, 
    r12, r14)
    if patchID == 0xca5269db50f4 then 
        uaddr1 = r0 
        uaddr2 = r2 
        if uaddr1 == uaddr2 then 
            return -22 
        else 
            return 0 
        end 
    end 
end 
kpatch.hook(0xca5269db50f4, "futex_requeue")
```

More complex examples in the paper
<table>
<thead>
<tr>
<th>API</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>hook</td>
<td>Hook a function for live patching</td>
</tr>
<tr>
<td>subhook</td>
<td>Hook the calls to sub-functions for live patching</td>
</tr>
<tr>
<td>alloc_mem</td>
<td>Allocate memory for live patching</td>
</tr>
<tr>
<td>free_mem</td>
<td>Free the allocated memory for live patching</td>
</tr>
<tr>
<td>get_callee</td>
<td>Locate a callee that can be hooked</td>
</tr>
<tr>
<td>search_symbol</td>
<td>Get the kernel symbol address</td>
</tr>
<tr>
<td>current_thread</td>
<td>Get the current thread context</td>
</tr>
<tr>
<td>read_buf</td>
<td>Read raw bytes from memory with the given size</td>
</tr>
<tr>
<td>read_int_8</td>
<td>Read 8 bits from memory as an integer</td>
</tr>
<tr>
<td>read_int_16</td>
<td>Read 16 bits from memory as an integer</td>
</tr>
<tr>
<td>read_int_32</td>
<td>Read 32 bits from memory as an integer</td>
</tr>
<tr>
<td>read_int_64</td>
<td>Read 64 bits from memory as an integer</td>
</tr>
</tbody>
</table>
## KARMA API

<table>
<thead>
<tr>
<th>API</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>hook</td>
<td>Hook a function for live patching</td>
</tr>
<tr>
<td>subhook</td>
<td>Hook the calls to sub-functions for live patching</td>
</tr>
<tr>
<td>alloc_mem</td>
<td>Allocate memory for live patching</td>
</tr>
<tr>
<td>free_mem</td>
<td>Free the allocated memory for live patching</td>
</tr>
<tr>
<td>get_callee</td>
<td>Locate a callee that can be hooked</td>
</tr>
<tr>
<td>search_symbol</td>
<td>Get the kernel symbol address</td>
</tr>
<tr>
<td>current_thread</td>
<td>Get the current thread context</td>
</tr>
<tr>
<td>read_buf</td>
<td>Read raw bytes from memory with the given size</td>
</tr>
<tr>
<td>read_int_8</td>
<td>Read 8 bits from memory as an integer</td>
</tr>
<tr>
<td>read_int_16</td>
<td>Read 16 bits from memory as an integer</td>
</tr>
<tr>
<td>read_int_32</td>
<td>Read 32 bits from memory as an integer</td>
</tr>
<tr>
<td>read_int_64</td>
<td>Read 64 bits from memory as an integer</td>
</tr>
</tbody>
</table>

Available to patches
KARMA Architecture

1. Offline Patch Generation and Verification
   - Vulnerable Function Identification
   - Semantic Matching
   - Signed Patches for Target Kernels

2. Online Live Patching by KARMA Client
   - Download & Verify Patch
   - Apply Patch
Offline Patch Adaptation
Offline Patch Adaptation

Three steps:

1. **Identify** the vulnerable functions in the target kernel
   - Same function but different names
   - Inlined

2. **Check** if the reference patch works for the target kernel
   - Same function but different semantics

3. **Adapt** the reference patch for the target kernel
Vulnerable Function Identification Example
CVE-2015-3636 (PingPong Root)

Device A: ping_unhash

Device B: ping_v4_unhash

Call graph based similarity comparison
Semantic Matching

- Check if two functions are semantically equivalent
- If so, adapt the reference patch to the target kernel
- Syntactic matching is too strict
  - Different compilers can generate different code with same semantics
    - Instruction order, register allocation, instruction selection, code layout
Semantic Matching

Same semantics with different syntax
Semantic Matching

Same semantics with different syntax
Semantic Matching

Same semantics with different syntax
Semantic Matching

Same semantics with different syntax
Semantic Matching

Same semantics with different syntax
Semantic Matching

• Check if two functions are semantically equivalent
• If so, adapt the reference patch to the target kernel
• Syntactic matching is too strict
  – Different compilers can generate different code with same semantics
    • Instruction order, register allocation, instruction selection, code layout
• Use **symbolic execution** to abstract these differences and adapt patches
  – Use approximation to improve scalability (details in the paper)
Online Patch Application

Function entry point hooking
Prototype Implementation

• Lua engine in kernel (11K SLOC)
  – Simple
  – Memory-safe
  – Easy to embed and extend
  – 24 years of development

• Semantic matching
  – angr
Evaluation: Applicability

- Evaluated **76** critical vulnerabilities in the last three years
- Patch level:
  - Level-1: 49
  - Level-2: 22
  - Level-3: 5
## Evaluation: Adaptability

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>CVE ID</th>
<th># of Opcode Clusters</th>
<th>% of the Largest Opcode Cluster</th>
<th># of Syntax Clusters</th>
<th>% of the Largest Syntax Cluster</th>
<th># of Semantic Clusters</th>
<th>% of the Largest Semantic Cluster</th>
<th>Semantic Matching Time Cost</th>
<th># of Instructions</th>
<th># of Basic Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock_diag_recv_msg</td>
<td>2013-1763</td>
<td>35</td>
<td>25.0%</td>
<td>7</td>
<td>73.5%</td>
<td>3</td>
<td>75.5%</td>
<td>10.5s</td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>perf_swevent_init</td>
<td>2013-2094</td>
<td>9</td>
<td>55.9%</td>
<td>5</td>
<td>55.9%</td>
<td>2</td>
<td>96.3%</td>
<td>24.6s</td>
<td>81</td>
<td>22</td>
</tr>
<tr>
<td>fb_mmap</td>
<td>2013-2596</td>
<td>26</td>
<td>20.2%</td>
<td>7</td>
<td>44.4%</td>
<td>5</td>
<td>66.9%</td>
<td>12.2s</td>
<td>102</td>
<td>15</td>
</tr>
<tr>
<td>__get_user_1</td>
<td>2013-6282</td>
<td>3</td>
<td>92.4%</td>
<td>2</td>
<td>92.4%</td>
<td>2</td>
<td>98.0%</td>
<td>3.2s</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>futex_requeue</td>
<td>2014-3153</td>
<td>54</td>
<td>14.8%</td>
<td>9</td>
<td>71.0%</td>
<td>3</td>
<td>99.3%</td>
<td>35.8s</td>
<td>459</td>
<td>107</td>
</tr>
<tr>
<td>msm_isp_proc_cmd</td>
<td>2014-4321</td>
<td>42</td>
<td>22.0%</td>
<td>5</td>
<td>66.5%</td>
<td>3</td>
<td>42.8%</td>
<td>8.8s</td>
<td>385</td>
<td>68</td>
</tr>
<tr>
<td>send_write Packing_test_read</td>
<td>2014-9878</td>
<td>12</td>
<td>57.6%</td>
<td>4</td>
<td>61.2%</td>
<td>1</td>
<td>100%</td>
<td>4.9s</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>msm_cci_validate_queue</td>
<td>2014-9890</td>
<td>6</td>
<td>59.5%</td>
<td>4</td>
<td>84.9%</td>
<td>2</td>
<td>72.4%</td>
<td>6.7s</td>
<td>77</td>
<td>8</td>
</tr>
<tr>
<td>ping_unhash</td>
<td>2015-3636</td>
<td>36</td>
<td>12.5%</td>
<td>5</td>
<td>75.7%</td>
<td>3</td>
<td>50.5%</td>
<td>4.6s</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>q61sm_snd_model_buf_alloc</td>
<td>2015-8940</td>
<td>29</td>
<td>34.0%</td>
<td>9</td>
<td>36.6%</td>
<td>5</td>
<td>44.2%</td>
<td>9.9s</td>
<td>104</td>
<td>20</td>
</tr>
<tr>
<td>sys_perf_event_open</td>
<td>2016-0819</td>
<td>22</td>
<td>36.3%</td>
<td>6</td>
<td>46.9%</td>
<td>6</td>
<td>84.2%</td>
<td>34.6s</td>
<td>569</td>
<td>118</td>
</tr>
<tr>
<td>ksgl_ioctl_gpmem_alloc</td>
<td>2016-3842</td>
<td>16</td>
<td>35.4%</td>
<td>3</td>
<td>88.8%</td>
<td>4</td>
<td>46.0%</td>
<td>4.7s</td>
<td>79</td>
<td>11</td>
</tr>
<tr>
<td>is_ashmem_file</td>
<td>2016-5340</td>
<td>6</td>
<td>89.6%</td>
<td>2</td>
<td>93.9%</td>
<td>2</td>
<td>98.1%</td>
<td>0.8s</td>
<td>23</td>
<td>3</td>
</tr>
</tbody>
</table>
# Evaluation: Adaptability

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>CVE ID</th>
<th># of Opcode Clusters</th>
<th>% of the Largest Opcode Cluster</th>
<th># of Syntax Clusters</th>
<th>% of the Largest Syntax Cluster</th>
<th># of Semantic Clusters</th>
<th>% of the Largest Semantic Cluster</th>
<th>Semantic Matching Time Cost</th>
<th># of Instructions</th>
<th># of Basic Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock_diag_rcv_msg</td>
<td>2013-1763</td>
<td>35</td>
<td>25.0%</td>
<td>7</td>
<td>73.5%</td>
<td>3</td>
<td>75.5%</td>
<td>10.5s</td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>perf_swevent_init</td>
<td>2013-2094</td>
<td>9</td>
<td>55.9%</td>
<td>5</td>
<td>55.9%</td>
<td>2</td>
<td>96.3%</td>
<td>24.6s</td>
<td>81</td>
<td>22</td>
</tr>
<tr>
<td>fb_mmap</td>
<td>2013-2596</td>
<td>26</td>
<td>20.2%</td>
<td>7</td>
<td>44.4%</td>
<td>5</td>
<td>66.9%</td>
<td>12.2s</td>
<td>102</td>
<td>15</td>
</tr>
<tr>
<td>__get_user_1</td>
<td>2013-6282</td>
<td>3</td>
<td>92.4%</td>
<td>2</td>
<td>92.4%</td>
<td>2</td>
<td>98.0%</td>
<td>3.2s</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>futex_requeue</td>
<td>2014-3153</td>
<td>54</td>
<td>14.8%</td>
<td>9</td>
<td>71.0%</td>
<td>3</td>
<td>99.3%</td>
<td>35.8s</td>
<td>459</td>
<td>107</td>
</tr>
<tr>
<td>msm_isp_proc_cmd</td>
<td>2014-4321</td>
<td>42</td>
<td>22.0%</td>
<td>5</td>
<td>66.5%</td>
<td>3</td>
<td>42.8%</td>
<td>8.8s</td>
<td>385</td>
<td>68</td>
</tr>
<tr>
<td>send_write packing test read</td>
<td>2014-9878</td>
<td>12</td>
<td>57.6%</td>
<td>4</td>
<td>61.2%</td>
<td>1</td>
<td>100%</td>
<td>4.9s</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>msm_cci_validate_queue</td>
<td>2014-9890</td>
<td>6</td>
<td>59.5%</td>
<td>4</td>
<td>84.9%</td>
<td>2</td>
<td>72.4%</td>
<td>6.7s</td>
<td>77</td>
<td>8</td>
</tr>
<tr>
<td>ping_unhash</td>
<td>2015-3636</td>
<td>36</td>
<td>12.5%</td>
<td>5</td>
<td>75.7%</td>
<td>3</td>
<td>50.5%</td>
<td>4.6s</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>q61sm_snd_model_buf_alloc</td>
<td>2015-8940</td>
<td>29</td>
<td>34.0%</td>
<td>9</td>
<td>36.6%</td>
<td>5</td>
<td>44.2%</td>
<td>9.9s</td>
<td>104</td>
<td>20</td>
</tr>
<tr>
<td>sys_perf_event_open</td>
<td>2016-0819</td>
<td>22</td>
<td>36.3%</td>
<td>6</td>
<td>46.9%</td>
<td>6</td>
<td>84.2%</td>
<td>34.6s</td>
<td>569</td>
<td>118</td>
</tr>
<tr>
<td>kgsl_ioctl_spumem_alloc</td>
<td>2016-3842</td>
<td>16</td>
<td>35.4%</td>
<td>3</td>
<td>88.8%</td>
<td>4</td>
<td>46.0%</td>
<td>4.7s</td>
<td>79</td>
<td>11</td>
</tr>
<tr>
<td>is_ashmem_file</td>
<td>2016-5340</td>
<td>6</td>
<td>89.6%</td>
<td>2</td>
<td>93.9%</td>
<td>2</td>
<td>98.1%</td>
<td>0.8s</td>
<td>23</td>
<td>3</td>
</tr>
</tbody>
</table>

Types and frequencies of instruction opcodes
## Evaluation: Adaptability

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>CVE ID</th>
<th># of Opcode Clusters</th>
<th>% of the Largest Opcode Cluster</th>
<th># of Syntax Clusters</th>
<th>% of the Largest Syntax Cluster</th>
<th># of Semantic Clusters</th>
<th>Semantic Matching Time Cost</th>
<th># of Instructions</th>
<th># of Basic Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock_diag_rcv_msg</td>
<td>2013-1763</td>
<td>35</td>
<td>25.0%</td>
<td>7</td>
<td>73.5%</td>
<td>3</td>
<td>10.5s</td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>perf_swevent_init</td>
<td>2013-2094</td>
<td>9</td>
<td>55.9%</td>
<td>5</td>
<td>55.9%</td>
<td>2</td>
<td>24.6s</td>
<td>81</td>
<td>22</td>
</tr>
<tr>
<td>fb_mmap</td>
<td>2013-2596</td>
<td>26</td>
<td>20.2%</td>
<td>7</td>
<td>44.4%</td>
<td>5</td>
<td>12.2s</td>
<td>102</td>
<td>15</td>
</tr>
<tr>
<td>__get_user_1</td>
<td>2013-6282</td>
<td>3</td>
<td>92.4%</td>
<td>2</td>
<td>92.4%</td>
<td>2</td>
<td>3.2s</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>futex_requeue</td>
<td>2014-3153</td>
<td>54</td>
<td>14.8%</td>
<td>9</td>
<td>71.0%</td>
<td>3</td>
<td>35.8s</td>
<td>459</td>
<td>107</td>
</tr>
<tr>
<td>msm_isp_proc_cmd</td>
<td>2014-4321</td>
<td>42</td>
<td>22.0%</td>
<td>5</td>
<td>66.5%</td>
<td>3</td>
<td>8.8s</td>
<td>385</td>
<td>68</td>
</tr>
<tr>
<td>send_write_packing_test_read</td>
<td>2014-9878</td>
<td>12</td>
<td>57.6%</td>
<td>4</td>
<td>61.2%</td>
<td>1</td>
<td>4.9s</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>msm_cci_validate_queue</td>
<td>2014-9890</td>
<td>6</td>
<td>59.5%</td>
<td>4</td>
<td>84.9%</td>
<td>2</td>
<td>6.7s</td>
<td>77</td>
<td>8</td>
</tr>
<tr>
<td>ping_unhash</td>
<td>2015-3636</td>
<td>36</td>
<td>12.5%</td>
<td>5</td>
<td>75.7%</td>
<td>3</td>
<td>4.6s</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>q61sm_snd_model_buf_alloc</td>
<td>2015-8940</td>
<td>29</td>
<td>34.0%</td>
<td>9</td>
<td>36.6%</td>
<td>5</td>
<td>9.9s</td>
<td>104</td>
<td>20</td>
</tr>
<tr>
<td>sys_perf_event_open</td>
<td>2016-0819</td>
<td>22</td>
<td>36.3%</td>
<td>6</td>
<td>46.9%</td>
<td>6</td>
<td>34.6s</td>
<td>569</td>
<td>118</td>
</tr>
<tr>
<td>ksml_ioctl_spunem_alloc</td>
<td>2016-3842</td>
<td>16</td>
<td>35.4%</td>
<td>3</td>
<td>88.8%</td>
<td>4</td>
<td>4.7s</td>
<td>79</td>
<td>11</td>
</tr>
<tr>
<td>is_ashmem_file</td>
<td>2016-5340</td>
<td>6</td>
<td>89.6%</td>
<td>2</td>
<td>93.9%</td>
<td>2</td>
<td>0.8s</td>
<td>23</td>
<td>3</td>
</tr>
</tbody>
</table>

Number of function calls and conditional branches (to abstract CFG)
## Evaluation: Adaptability

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>CVE ID</th>
<th># of Opcode Clusters</th>
<th>% of Opcode Clusters</th>
<th># of Syntax Clusters</th>
<th>% of Syntax Clusters</th>
<th># of Semantic Clusters</th>
<th>% of Semantic Clusters</th>
<th>Semantic Matching Time Cost</th>
<th># of Instructions</th>
<th># of Basic Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock_diag_rcv_msg</td>
<td>2013-1763</td>
<td>35</td>
<td>25.0%</td>
<td>7</td>
<td>73.5%</td>
<td>3</td>
<td>75.5%</td>
<td>10.5s</td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>perf_swevent_init</td>
<td>2013-2094</td>
<td>9</td>
<td>55.9%</td>
<td>5</td>
<td>55.9%</td>
<td>2</td>
<td>96.3%</td>
<td>24.6s</td>
<td>81</td>
<td>22</td>
</tr>
<tr>
<td>fb_mmap</td>
<td>2013-2596</td>
<td>26</td>
<td>20.2%</td>
<td>7</td>
<td>44.4%</td>
<td>5</td>
<td>66.9%</td>
<td>12.2s</td>
<td>102</td>
<td>15</td>
</tr>
<tr>
<td>__get_user_l</td>
<td>2013-6282</td>
<td>3</td>
<td>92.4%</td>
<td>2</td>
<td>92.4%</td>
<td>2</td>
<td>98.0%</td>
<td>3.2s</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>futex_requeue</td>
<td>2014-3153</td>
<td>54</td>
<td>14.8%</td>
<td>9</td>
<td>71.0%</td>
<td>3</td>
<td>99.3%</td>
<td>35.8s</td>
<td>459</td>
<td>107</td>
</tr>
<tr>
<td>msm_isp_proc_cmd</td>
<td>2014-4321</td>
<td>42</td>
<td>22.0%</td>
<td>5</td>
<td>66.5%</td>
<td>3</td>
<td>42.8%</td>
<td>8.8s</td>
<td>385</td>
<td>68</td>
</tr>
<tr>
<td>send_write_packing_test_read</td>
<td>2014-9878</td>
<td>12</td>
<td>57.6%</td>
<td>4</td>
<td>61.2%</td>
<td>1</td>
<td>100%</td>
<td>4.9s</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>msm_cci_validate_queue</td>
<td>2014-9890</td>
<td>6</td>
<td>59.5%</td>
<td>4</td>
<td>84.9%</td>
<td>2</td>
<td>72.4%</td>
<td>6.7s</td>
<td>77</td>
<td>8</td>
</tr>
<tr>
<td>ping_unhash</td>
<td>2015-3636</td>
<td>36</td>
<td>12.5%</td>
<td>5</td>
<td>75.7%</td>
<td>3</td>
<td>50.5%</td>
<td>4.6s</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>q61sm_snd_model_buf_alloc</td>
<td>2015-8940</td>
<td>29</td>
<td>34.0%</td>
<td>9</td>
<td>36.6%</td>
<td>5</td>
<td>44.2%</td>
<td>9.9s</td>
<td>104</td>
<td>20</td>
</tr>
<tr>
<td>sys_perf_event_open</td>
<td>2016-0819</td>
<td>22</td>
<td>36.3%</td>
<td>6</td>
<td>46.9%</td>
<td>6</td>
<td>84.2%</td>
<td>34.6s</td>
<td>569</td>
<td>118</td>
</tr>
<tr>
<td>kgs1_ioctl_spumem_alloc</td>
<td>2016-3842</td>
<td>16</td>
<td>35.4%</td>
<td>3</td>
<td>88.8%</td>
<td>4</td>
<td>46.0%</td>
<td>4.7s</td>
<td>79</td>
<td>11</td>
</tr>
<tr>
<td>is_ashmem_file</td>
<td>2016-5340</td>
<td>6</td>
<td>89.6%</td>
<td>2</td>
<td>93.9%</td>
<td>2</td>
<td>98.1%</td>
<td>0.8s</td>
<td>23</td>
<td>3</td>
</tr>
</tbody>
</table>

**KARMA's semantic matching**
Evaluation: Performance

CF-Bench results with different patches
Evaluation: Performance

Execution time of `chmod` with different patches
Future Work

• User-space vulnerability protection
  – Project Treble → only partially solve the problem

• Lua engine in the kernel (11K SLOC)
  – Alternative execution engines, like BPF or sandboxed binary patches

• Error handling code could be vulnerable
  – Error injection to detect vulnerable error-handling code

• Improve semantic matching
Q & A

Adaptive Android Kernel Live Patching

www.YueChen.me
Backup Slides
Attack TrustZone from Kernel

• Example:
  – [Downgrade Attack on TrustZone](#) (see its references)
Observations

- Most kernel functions are **stable** across devices and Android releases.

![Graph showing the number of syntax clusters for each function]

- About 40% of the shared functions have only one cluster, and about 80% of them have 4 clusters or less.
Observations

- Most kernel functions are stable across devices and Android releases.

For about 60% of shared functions, the largest cluster contains more than 80% of all the kernels that have this function.
Symbolic Execution

• Challenges
  – Avoid path explosion
  – Impact to the environment

• Practical Solution
  – Non-local memory writes
  – Function calls (and their arguments)
  – Function return values

• Adaptation (e.g., mutate constants or offsets)
  – foo(symbol_A + 4, 36) ➔ foo(symbol_A + 8, 36)
Evaluation: Overall Performance

- Complex patch for most frequent syscall (gettimeofday) during web browsing
- Overall system performance overhead in this extreme situation: 0.9%
Example: CVE-2013-6123

```c
1 static long msm_ioctl_server(struct file * file, void *fh, bool valid_prio, int cmd, void *arg)
2 {
3     ...
4     if (copy_from_user(&u_isp_event,
5             (void __user *)(ioctl_ptr->ioctl_ptr,
6             sizeof(struct msm_isp_event_ctrl))) {
7         ...
8     }
9     ...
10    +    if(u_isp_event.isp_data.ctrl.queue_idx<0
11    +    || u_isp_event.isp_data.ctrl.queue_idx >=
12    +    MAX_NUM_ACTIVE_CAMERA) {
13    +    pr_err("%s: Invalid index %d\n",
14    +    __func__, u_isp_event.isp_data.
15    +    ctrl.queue_idx);
16    +    rc = -EINVAL;
17    +    return rc;
18    +    }
19 }
```
Example: CVE-2013-6123

```
static long msm_ioctl_server(struct file *file, void *fh, bool valid_prio, int cmd, void *arg)
{
    ...
    if (copy_from_user(&u_isp_event, (void *)__user *)ioctl_ptr->ioctl_ptr, sizeof(struct msm_isp_event_ctrl)) {
        ...
    }
    ...
    if(u_isp_event.isp_data.ctrl.queue_idx < 0 || u_isp_event.isp_data.ctrl.queue_idx >= MAX_NUM_ACTIVE_CAMERA) {
        pr_err("%s: Invalid index %d\n", __func__, u_isp_event.isp_data.ctrl.queue_idx);
        rc = -EINVAL;
        return rc;
    }
    ...
}
```
Example: CVE-2013-6123

static long msm_ioctl_server(struct file *file, void *fh, bool valid_prio, int cmd, void *arg)
{
  ...
  if (copy_from_user(&u_isp_event, (void *)__user *)(ioctl_ptr->ioctl_ptr, sizeof(struct msm_isp_event_ctrl))) {
    ...
  }
  ...
  if(u_isp_event isp_data.ctrl.queue_idx<0 || u_isp_event.isp_data.ctrl.queue_idx >= MAX_NUM_ACTIVE_CAMERA) {
    pr_err("%s: Invalid index %d\n", __func__, u_isp_event.isp_data.ctrl.queue_idx);
    rc = -EINVAL;
    return rc;
  }
  ...
}
Example: CVE-2013-6123

```c
static long msm_ioctl_server(struct file *
    file, void *fh, bool valid_prio, int cmd,
    void *arg)
{
    ...
    if (copy_from_user(&u_isp_event,
        (void *)__user *)(ioctl_ptr->ioctl_ptr,
        sizeof(struct msm_isp_event_ctrl))) {
        ...
    }
    ...
    if(u_isp_event.isp_data.ctrl.queue_idx<0
        || u_isp_event.isp_data.ctrl.queue_idx >=
        MAX_NUM_ACTIVE_CAMERA) {
        pr_err("%s: Invalid index %d\n",
            __func__, u_isp_event.isp_data.
            ctrl.queue_idx);
        rc = -EINVAL;
        return rc;
    }
    ...
}
```
Example: CVE-2013-6123

```c
static long msm_ioctl_server(struct file *file, void *fh, bool valid_prio, int cmd, void *arg)
{
  ...
  if (copy_from_user(&u_isp_event, (void __user *)ioctl_ptr->ioctl_ptr, sizeof(struct msm_isp_event_ctrl))) {
    ...
  }
  ...
  if(u_isp_event.isp_data.ctrl.queue_idx<0 || u_isp_event.isp_data.ctrl.queue_idx >= MAX_NUM_ACTIVE_CAMERA) {
    pr_err("%s: Invalid index %d\n", __func__, u_isp_event.isp_data.ctrl.queue_idx);
    rc = -EINVAL;
    return rc;
  }
  ...
}
```
Example: CVE-2013-6123

```c
1 static long msm_ioctl_server(struct file *file, void *fh, bool valid_prio, int cmd, void *arg)
2 {
3     ...
4     if (copy_from_user(&u_isp_event,
5         (void *)__user *)ioctl_ptr->ioctl_ptr,
6         sizeof(struct msm_isp_event_ctrl))) {
7         ...
8     }
9     ...
10 +    if(u_isp_event.isp_data.ctrl.queue_idx<0
11 +    || u_isp_event.isp_data.ctrl.queue_idx »
12 +    MAX_NUM_ACTIVE_CAMERA) {
13 +        pr_err("%s: Invalid index %d\n",
14 +            __func__, u_isp_event.isp_data.
15 +            ctrl.queue_idx);
16 +        rc = -EINVAL;
17 +    }
18 +    return rc;
19 +}
```
You May Also Like

• A time machine to locate vulnerabilities:
  – Pinpointing Vulnerabilities

• Protect your computer by encrypting memory all the time:
  – Secure In-Cache Execution

• Fine-grained dynamic ASLR during runtime:
  – Remix: On-demand Live Randomization