InstaGuard: Instantly Deployable Hot-patches for Vulnerable System Programs on Android

Yaohui Chen, Yuping Li, Long Lu, Yueh-Hsun Lin, Hayawardh Vijayakumar, Zhi Wang, Xinming Ou
Prolonged Android System Security Updates

• Require lengthy tests from various parties
  • Security
  • Compatibility

• Done by each go-to-market partners
  • It is justified, but too time-consuming
To Accelerate the Process

- Monthly Security Maintenance Release (SMR)
- Project Treble
  - Isolate SoC vendors from Google and OEMs when preparing new OS updates
  - OS updates still need to be done and tested by OEMs and carriers
- Hot patches
  - KARMA [Chen et al, Usenix Security’17]
  - Patchdroid [Mulliner et al, ACSAC’13]
  - InstaGuard
Background – Code update v.s Policy update

• Matured and commercialized **code update based** hot patch solutions
  • Microsoft Hotfix
  • Ubuntu Livepatch
  • Not carrier-passthrough

• Policy updates
  • Less dangerous, more restrictive than code update
  • Takes much shorter time to go through
  • Infrastructure for policy update is in-place and well established
Introducing InstaGuard

• Approach
  • Utilize **policy-driven update** to timely mitigate critical **user-level system** vulnerabilities

• Key difference to traditional hot patches
  • Non-code update
  • Restrictiveness (fail-safe)
Threat Model

**Trusted**
- Kernel
- Hardware

**Benign but vulnerable**
- User-level system daemons
- User-level system libraries

**Can’t handle**
- Zero-day vulnerabilities
- Compromised system daemons
Unique Challenges

Policy language

- Expressive to describe various kinds of vulnerabilities
- Restrictive to be fail-safe

Policy generation

- Automated policy generation
System Overview

| InstaGuard (online) | RuleMaker (offline) |
InstaGuard

- Zoom in to the on-device InstaGuard
GuardRule Basic Format

GuardRule-Header

Rule ID
- hash
- vuln Type

Target Module
- binary path
- target name

Alert Decision
- BLOCK
- ALERT

GuardRule-Body

Breakpoints
- install first
- address
- action

Watchpoints
- install first
- address
- size
- action

Assertions
- relation op
- data constraints
- action
GuardRule

Expressive: precisely capture control and data properties

• Precisely describe various kinds of vulnerabilities
  • logic bug, integer overflow, buffer overflow, out-of-bound access, use-after-free and race conditions

Fail-safe: buggy (or malicious) rules can only cause DoS

• The InstaGuard mechanism simply do not support any intrusive operations.
InstaGuard in Action

• CVE-2016-3861 (logic bug)

```c
ssize_t utf16_to_utf8_length(const char16_t *src, size_t src_len) {

  //sanity checks
  size_t ret = 0;
  const char16_t * const end = src + src_len;
  while (src < end) {
    if (((*src & 0xFC00) == 0xD800 || (*src + 1) & 0xFC00) == 0xDC00) {
      // surrogate pairs are always 4 bytes.
      ret += 4;
      src++;
    } else {
      ret += utf32_codepoint_utf8_length((char32_t)*src);
      src++;
    }
  }

  return ret;
}
```

```xml
<rules>
  <rule cve="CVE-2016-3861">
    <module_name>libutils.so</module_name>
    <decision>BLOCK</decision>
    <binary_path>/system/bin/mediaserver</binary_path>
    <break_points>
      <break_point first=true, id=0>
        <!--binary address corresponding to line 9 in Listing 1-->
        <address>0x08055000</address>
        <!--next action: activating assertion primitive with id 0-->
        <action>VERIFY AS#0</action>
      </break_point>
    </break_points>
    <assertions>
      <!--if assertion evaluate to true InstaGuard BLOCK the execution as node ?decision? specify-->
      <assertion id=0, action=decision>
        <data_constraints>
          <data_constraint>
            <ops>NE</ops>
            <left_exp>
              <!--retrieval rule for *src-->  
              <node id=0>reg_2_32</node>
            </left_exp>
            <right_exp>
              <node>const_0xFC00</node>
            </right_exp>
          </data_constraint>
        </data_constraints>
      </assertion>
    </assertions>
  </rule>
</rules>
```
InstaGuard in Action Cont.

- **CVE-2016-7911** (race condition)

```c
// src/block/blk-ioc.c
void exit_io_context(struct task_struct *task)
{
    struct io_context *ioc;
    task_lock(task);
    ioc = task->io_context;
    task->io_context = NULL;
    task_unlock(task);
    atomic_dec(&ioc->nr_tasks);
    put_io_context_active(ioc);
}

// src/block/ioprio.c
static int get_task_ioprio(struct task_struct *p)
{
    int ret;
    ret = security_task_getioprio(p);
    if (ret)
        goto out;
    ret = IOPRIO_PRIO_VALUE(IOPRIO_CLASS_NONE,
                            IOPRIO_NORM);
    if (p->io_context)
        ret = p->io_context->ioprio;
out:
    return ret;
}
```

- **Con**

- **BP**

- **BP**

- **BP**

- **BP**

- **BP**

- **BP**

- **BP**

- **AS**

```c
<rules>
  <rule name="CVE-2016-7911">
    <module_name>get_task_ioprio | 154
    <binary_path>/system/bin/dummy</binary_path>
    <assertion>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>iocblkblk-ioc.c | exit_io_context | 204</address>
      <action>
        return 0;
      </action>
      <break_point>
      <first>true</first>
      <address>/system/bin/dummy</address>
      <data_constraints>
        <data_constraint>
          <left_exp>nr_tasks</left_exp>
          <right_exp>io_context</right_exp>
          <id>5</id>
        </data_constraint>
      </data_constraints>
      <actions>
        <action>
          task_unlock(task);
        </action>
      </actions>
      <assertions>
        <assertion>
          <left_exp>io_context_active</left_exp>
          <right_exp>iocblkblk-ioc.c | exit_io_context | 204</right_exp>
          <id>6</id>
        </assertion>
      </assertions>
      </assertion>
  </rule>
</rules>
```
GuardSpec and RuleMaker

- **GuardSpec**
  - High-level **vulnerability description**
  - Hides the details about InstaGuard primitives
- **RuleMaker**
  - **Automatically** synthesizes GuardRules
  - Conversion based on empirical experiences
  - Several implementation challenges in paper
# GuardSpec Showcases

## CVE-2016-3861 (logic bug)

<table>
<thead>
<tr>
<th>ID</th>
<th>CVE-2016-3861</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Path</td>
<td>/system/bin/mediaserver</td>
</tr>
<tr>
<td>Module Name</td>
<td>libutils.so</td>
</tr>
<tr>
<td>Decision</td>
<td>BLOCK</td>
</tr>
<tr>
<td>Vul Location</td>
<td>system/core/libutils/Unicode.cpp</td>
</tr>
<tr>
<td>Operation</td>
<td>NE</td>
</tr>
</tbody>
</table>

- Vulnerable Location: `utf16_to_utf8_length` (411)

## CVE-2016-7911 (race condition)

<table>
<thead>
<tr>
<th>ID</th>
<th>CVE-2016-7911</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Path</td>
<td>/system/bin/dummy</td>
</tr>
<tr>
<td>Module Name</td>
<td>kernel</td>
</tr>
<tr>
<td>Decision</td>
<td>BLOCK</td>
</tr>
<tr>
<td>Race Condition</td>
<td>TASK</td>
</tr>
<tr>
<td>Vul Location</td>
<td>/block/blk-ioc.c</td>
</tr>
<tr>
<td>Operation</td>
<td>true</td>
</tr>
</tbody>
</table>

- Race Location 1: `exit_io_context` (204)
- Race Location 2: `get_task_ioprio` (151)

## CVE-2016-3871 (buffer overflow)

<table>
<thead>
<tr>
<th>ID</th>
<th>CVE-2016-3871</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Path</td>
<td>/system/bin/mediaserver</td>
</tr>
<tr>
<td>Module Name</td>
<td>libstagefright_soft_avcenc.so</td>
</tr>
<tr>
<td>Decision</td>
<td>BLOCK</td>
</tr>
<tr>
<td>Vul Location</td>
<td>libstagefright/codecs/mp3dec/SoftMP3.cpp</td>
</tr>
<tr>
<td>Operation</td>
<td>NE</td>
</tr>
</tbody>
</table>

- Vulnerable Location: `libstagefright/codecs/mp3dec/SoftMP3.cpp`

## CVE-2016-3895 (integer overflow)

<table>
<thead>
<tr>
<th>ID</th>
<th>CVE-2016-3895</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Path</td>
<td>/system/bin/surfaceflinger</td>
</tr>
<tr>
<td>Module Name</td>
<td>libui.so</td>
</tr>
<tr>
<td>Decision</td>
<td>BLOCK</td>
</tr>
<tr>
<td>Integer Condition</td>
<td>MAX</td>
</tr>
<tr>
<td>Vul Location</td>
<td>frameworks/native/libs/ul/Region.cpp</td>
</tr>
<tr>
<td>Operation</td>
<td>true</td>
</tr>
</tbody>
</table>

- Vulnerable Location: `frameworks/native/libs/ul/Region.cpp`

---

**CVE-2016-3861** (logic bug)

```c
[common]
ID = CVE-2016-3861
binary_path = /system/bin/mediaserver
module_name = libutils.so
decision = BLOCK

[logic bug]
vul_location = system/core/libutils/Unicode.cpp |
  utf16_to_utf8_length | 411
lexp = *src & 0xRFC00
rexp = 0xDC00
relation_op = NE
```

**CVE-2016-7911** (race condition)

```c
[common]
ID = CVE-2016-7911
binary_path = /system/bin/dummy
module_name = kernel
decision = BLOCK

[race condition]
racer_location1 = task : /block/blk-ioc.c |
  exit_io_context | 204 : /block/blk-ioc.c |
  get_task_ioprio | 151 : /block/ioprio.c |
  get_task_ioprio | 154
```

**CVE-2016-3871** (buffer overflow)

```c
[common]
ID = CVE-2016-3871
binary_path = /system/bin/mediaserver
module_name = libstagefright_soft_avcenc.so
decision = BLOCK

[buffer overflow]
buf_name = outHeader->pBuffer
buf_size = outHeader->nAllocLen
vul_location =
  libstagefright/codecs/mp3dec/SoftMP3.cpp
  |SoftMP3::internalGetParameter | 303
```

**CVE-2016-3895** (integer overflow)

```c
[common]
ID = CVE-2016-3895
binary_path = /system/bin/surfaceflinger
module_name = libui.so
decision = BLOCK

[integer overflow]
involved_vars = numRects, Rect
overflow_exp = numRects * Rect
overflow_dir = MAX
trigger_value = 0xffffffff
vul_location = frameworks/native/libs/ul/Region.cpp
  |Region::flatten | 794
```
Evaluation

• Expressiveness
  • Can InstaGuard generically block different kinds of real-world vulnerabilities?

• Ease of use
  • How easy is it to make use of the InstaGuard framework?

• Overhead
  • What are the runtime and memory overheads of generated policies?
Expressiveness Evaluation

- 30 critical framework vulnerabilities from Android security bulletin from 2016
  - 28/30 GuardSpec are less than 10 lines
  - Info leak is not easy to mitigate without risking availability lost
  - UaF and Race conditions mostly find their root causes as logical bugs
Ease of Use

- Facilitated by security researchers from Samsung
- Tested 4 different categories of bugs
- Task: write GuardSpec and then synthesize GuardRule using RuleMaker

<table>
<thead>
<tr>
<th>Vulnerability type</th>
<th>GuardSpec compose time (mins)</th>
<th>GuardSpec Line#</th>
<th>Synthesized GuardRule Line#</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-3895</td>
<td>Integer Overflow</td>
<td>40</td>
<td>9</td>
</tr>
<tr>
<td>CVE-2016-0836</td>
<td>Out-of-Bound</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>CVE 2016-3861</td>
<td>Logic Bug</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>CVE-2015-1474</td>
<td>Buffer Overflow</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>Avg.</td>
<td></td>
<td>22.5</td>
<td>7.75</td>
</tr>
</tbody>
</table>
## Runtime and Memory Overheads

- Unit tests
- Stacked GuardRules

<table>
<thead>
<tr>
<th></th>
<th>Used Primitives</th>
<th>Vulnerability type</th>
<th>Memory Overhead(%)</th>
<th>Runtime Slowdown(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-3895</td>
<td>(1x)BP, (1x)AS</td>
<td>Integer Overflow</td>
<td>0.37%</td>
<td>2.89%</td>
</tr>
<tr>
<td>CVE-2016-0836</td>
<td>(1x)BP, (1x)AS</td>
<td>Out-of-Bound</td>
<td>4.11%</td>
<td>3.27%</td>
</tr>
<tr>
<td>CVE 2016-3861</td>
<td>(1x)BP, (1x)AS</td>
<td>Logic Bug</td>
<td>1.08%</td>
<td>2.70%</td>
</tr>
<tr>
<td>CVE-2015-1474</td>
<td>(2x)BP, (1x)WP, (1x)AS</td>
<td>Buffer Overflow</td>
<td>1.19%</td>
<td>1.94%</td>
</tr>
<tr>
<td>Avg.</td>
<td></td>
<td></td>
<td>1.69%</td>
<td>2.70%</td>
</tr>
<tr>
<td><strong>Wrapper service</strong></td>
<td><strong>All of the above</strong></td>
<td><strong>All of the above</strong></td>
<td><strong>0.48%</strong></td>
<td><strong>9.73%</strong></td>
</tr>
</tbody>
</table>
Conclusion

• InstaGuard aims to make timely hot-patches available to users
  • Leverage **policy update** instead of code update
  • GuardRules are **expressive** yet **restrictive**

• RuleMaker synthesizes GuardRule from GuardSpec
  • GuardSpecs are high-level **vulnerability description**
  • Hides low-level InstaGuard primitives from user

• Evaluation on critical system vulnerabilities reveals minor overheads
Thank You!

Yaohui Chen
chen.yaoh@husky.neu.edu
More Details about Data constraints
Necessary Data Constraint Operators

• logical and, or operators: &&, ||

• relational operators: ==, !=, <, >, <=, >=

• binary operators: +, -, *, /, %, &, |, ^

• unary operators: +, -, !(logical negation), ~(bitwise not)
Data Constraint Representation

- **primaryExpression:**
  - basicOperands
  - **unaryOps** basicOperands [-,!,~]
  - primaryExpression **binaryOps** primaryExpression [+,-,*,/,%,&,|,^]

- **conditionalExpression:**
  - primaryExpression **relationalOps** primaryExpression [==,!=,>,<,>=,<=]

- Conditional expression format: (type), (ops), (leftExp), (rightExp);
Data constraints format

- `<data_constraints>`
  - `<type>int64</type>`
  - `<relational_operator>=</relational_operator>`
  - `<left_exp>`
    - `<ops>*,+</ops>`
    - `<var_categories>global,local,local</var_categories>`
    - `<operands>x,y,z</operands>`
  - `</left_exp>`
  - `<right_exp>`
    - `<ops></ops>`
    - `<var_categories>const</var_categories>`
    - `<operands>2</operands>`
  - `</right_exp>`
- `<data_constraints>`