Charm: Facilitating Dynamic Analysis of Device Drivers of Mobile Systems

Seyed Mohammadjavad Seyed Talebi, Hamid Tavakoli, Hang Zhang, Zheng Zhang, Ardalan Amiri Sani, Zhiyun Qian

UC Irvine    UC Riverside
What is the problem?

Key ideas to solve the problem

Design
Evaluation
Summary
Security of mobile systems is vital
Mobile systems are diverse

- More than 1,000 Android device manufacturers
- More than 24,000 distinct Android devices
Diverse hardware → many device drivers

Vendors competition → more features
→ more hardwares → more device drivers
Device drivers are a major risk to the security of mobile systems

How to investigate bugs in device drivers of mobile systems?

Bugs found in Android's Kernel

- Rest of kernel
- Drivers

- 15.0%
- 85.0%

Dynamic analysis is useful to find vulnerabilities

- Fuzzing
- Interactive debugging
- Record-and-replay
  - REC
  - Play
- Selective Symbolic Execution

Dynamic taint analysis
Many existing dynamic analysis tools use virtual machines

- kAFL
- Digtool

Interactive debugging
- GDB

Record-and-replay
- QEMU

Selective Symbolic Execution
- S²E

Dynamic taint analysis
- DECAF
Many existing dynamic analysis tools use virtual machines

- Fuzzing
  - kAFL
  - Digtool

- Interactive debugging
  - GDB

- Record-and-replay
  - REC
  - Play
  - QEMU

- Selective Symbolic Execution
  - S²E

- Dynamic taint analysis
  - DECAF
Many existing dynamic analysis tools use virtual machines

- Fuzzing
  - kAFL
  - Digtool

- Interactive debugging
  - GDB

- Record-and-replay
  - QEMU

- Selective Symbolic Execution
  - S\textsuperscript{2}E

- Dynamic taint analysis
  - DECAF
Many existing dynamic analysis tools use virtual machines

- Fuzzing
  - kAFL
  - Digtool

- Interactive debugging
  - GDB

- Record-and-replay
  - QEMU

- Selective Symbolic Execution
  - S²E

- Dynamic taint analysis
  - DECAF
Many existing dynamic analysis tools use virtual machines

- Fuzzing
  - kAFL
  - Digtool

- Interactive debugging
  - GDB

- Record-and-replay
  - QEMU

- Selective Symbolic Execution
  - S²E

Dynamic taint analysis

- DECAF
Applying these tools to device drivers in mobile systems is hard

Hardware assisted
virtual machine

Not available
Applying these tools to device drivers in mobile systems is hard

Hardware assisted virtual machine
Not available

Software only virtual machine
Poor performance
Key ideas to solve the problem
Key idea 1: running device drivers of a mobile system in a virtual machine on a workstation
Key idea 1: running device driver of a mobile system in a virtual machine on a workstation

Virtual machine

Device driver(s) of mobile system

Fails without I/O access
Key idea 2: use the mobile device to serve low-level I/O operations
Design

Evaluation

Summary
Device driver of a mobile system: a closer look
Device driver of a mobile system: a closer look

- Memory mapped register read/writes
- Interrupt
Device driver of a mobile system: a closer look

Mobile system

OS

User space

Kernel

Device driver

Shared modules

I/O device

Shared HW

Clock, power management, GPIO, and pin control
Device driver of a mobile system: a closer look

Function calls to Linux API for shared modules
Move the device driver to a workstation

**Workstation**
- Virtual machine OS
  - User space
  - Kernel
- Hypervisor

**Mobile system**
- OS
  - User space
  - Kernel
- Device driver
- Shared modules
- I/O device
- Shared HW

Move the device driver to a workstation
Move the device driver to a workstation

**Workstation**
- **Virtual machine OS**
  - User space
  - Kernel
- **Device driver**
- **Hypervisor**

**Mobile system**
- **OS**
  - User space
  - Kernel
- **Shared modules**
- **I/O device**
  - Shared HW

Move the device driver to a workstation.
Challenge: cannot move shared modules

Mobile system cannot boot without clock, power management, GPIO, and pin control modules.
Do not move shared modules

**Workstation**

- **Virtual machine OS**
  - **User space**
  - **Kernel**
  - **Device driver**

**Mobile system**

- **OS**
  - **User space**
  - **Kernel**
  - **Shared modules**
  - **I/O device**
  - **Shared HW**
Remote I/O operations

Device driver

Fails without I/O access

Shared modules

I/O device

Shared HW
Low latency USB channel

**Workstation**

- Virtual machine OS
  - User space
  - Kernel
  - Device driver
  - Hypervisor

**Mobile system**

- OS
  - User space
  - Kernel
  - Stub
  - Shared modules
  - I/O device
  - Shared HW

USB channel
Design decision 2: low latency USB channel

Workstation

Virtual machine OS

User space

Kernel

Device driver

Hypervisor

Mobile system

OS

User space

Kernel

Stub

Normal USB channel
~ 2 ms Latency

Charm USB channel

I/O device

Shared modules

Stub

Shared HW

~ 100 us Latency

Design decision 2: low latency USB channel

Normal USB channel
~ 2 ms Latency

Charm USB channel

~ 100 us Latency

Design decision 2: low latency USB channel

Normal USB channel
~ 2 ms Latency

Charm USB channel

~ 100 us Latency
Remote I/O interface 1: remote register read/write

Workstation
- Virtual machine OS
  - User space
  - Kernel
- Hypervisor
- Device driver

Mobile system
- OS
  - User space
  - Kernel
- I/O device
- Stub
- USB channel
- Shared modules
- Shared HW
Remote I/O interface 2: remote interrupt handling

Workstation

Virtual machine OS

Kernel

User space

Hypervisor

Device driver

Mobile system

OS

Kernel

User space

I/O device

Shared modules

USB channel

Stub

Stub

Shared HW
Remote I/O interface 3: Remote Procedure Call (RPC)
Charm supports various drivers and devices

<table>
<thead>
<tr>
<th>Model</th>
<th>Nexus 5X</th>
<th>Nexus 6P</th>
<th>Galaxy S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>LG</td>
<td>Huawei</td>
<td>Samsung</td>
</tr>
<tr>
<td>Supported drivers</td>
<td>Camera, Audio</td>
<td>GPU</td>
<td>IMU Sensors</td>
</tr>
<tr>
<td>Lines of Code Ported</td>
<td>65,000 + 30,000</td>
<td>31,000</td>
<td>3000</td>
</tr>
<tr>
<td>Porting time</td>
<td>-</td>
<td>7 days</td>
<td>2 days</td>
</tr>
</tbody>
</table>
## Time it takes to port a driver to Charm

<table>
<thead>
<tr>
<th>Model</th>
<th>Nexus 5X</th>
<th>Nexus 6P</th>
<th>Galaxy S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>LG</td>
<td>Huawei</td>
<td>Samsung</td>
</tr>
<tr>
<td>Supported drivers</td>
<td>Camera, Audio</td>
<td>GPU</td>
<td>IMU Sensors</td>
</tr>
<tr>
<td>Lines of Code Ported</td>
<td>65,000 + 30,000</td>
<td>31,000</td>
<td>3000</td>
</tr>
<tr>
<td>Porting time</td>
<td>-</td>
<td>7 days</td>
<td>2 days</td>
</tr>
</tbody>
</table>
Charm supports various dynamic analysis techniques

- Fuzzing
- Record-and-replay
- Manual Interactive debugging
How Charm facilitates fuzzing

More hardware support
How Charm facilitates fuzzing

More hardware support

- VT-x
- PT

More software support

- KASAN
- KMSAN
- KTSAN
How Charm facilitates fuzzing

Reliable console access

More hardware support

VT-x
PT

KASAN
KMSAN
KTSAN

More software support

No special hardware
Fuzzing scenarios

Scenario 1
Without Charm

Execute fuzzer on
the phone

Scenario 2
With Charm

Execute fuzzer on
the server
Fuzzing performance on Charm
Low overhead for fuzzing on Charm

Higher performance

Low-level I/O operations

Not frequent
# Bugs found by Charm

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of bugs</td>
<td>25</td>
</tr>
<tr>
<td>New bugs</td>
<td>14</td>
</tr>
<tr>
<td>Bugs found using KASAN</td>
<td>2</td>
</tr>
<tr>
<td>False positive bugs</td>
<td>0</td>
</tr>
</tbody>
</table>
Charm supports various dynamic analysis techniques

- Fuzzing
- Record-and-replay
- Manual Interactive debugging
Charm facilitates record-and-replay

- Not feasible without Charm for mobile device drivers
Record all remote I/O interactions

**Workstation**

- Virtual machine OS
  - User space
  - Kernel
- Hypervisor
- Device driver
- USB channel
- Stub
- REC

**Mobile system**

- OS
  - User space
  - Kernel
- Resident modules
- I/O device
- Resident hw
- Stub
Replay the recorded interactions
Replay the recorded interactions

Mobile system is not needed while replaying
Record-and-replay performance

![Bar chart showing camera initialization time (ms) for Charm, Charm-Record, Charm-Replay, and Phone.](chart)

- Charm: 1700 ms
- Charm-Record: 1800 ms
- Charm-Replay: 300 ms
- Phone: 500 ms
Charm supports various dynamic analysis techniques

- Fuzzing
- Record and Replay
- Manual Interactive debugging

GDB
The GNU Project Debugger
Charm facilitates manual interactive debugging

- Charm enables using GDB for device drivers

Breakpoint  Watchpoint  Single-step execution
Manual interactive debugging results

- We analyzed three known vulnerabilities
  - CVE-2016-3903: use-after-free bug
  - CVE-2016-2501: out-of-bounds access bug
  - CVE-2016-2061: out-of-bounds access bug
- We built an arbitrary kernel code execution exploit using CVE-2016-2061
## Related work

<table>
<thead>
<tr>
<th></th>
<th>Charm</th>
<th>Avatar [NDSS’14]</th>
<th>Surrogate [WOOT’15]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target</strong></td>
<td>Mobile systems, open source device drivers</td>
<td>Embedded systems firmware</td>
<td>Embedded systems firmware</td>
</tr>
<tr>
<td><strong>Forward I/O accesses</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Communication channel</strong></td>
<td>USB</td>
<td>UART and JTAG</td>
<td>PCIe FPGA board/JTAG</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Near native</td>
<td>Poor</td>
<td>Near native</td>
</tr>
</tbody>
</table>
# Limitations and Future work

<table>
<thead>
<tr>
<th>Current Implementation</th>
<th>Future work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual port of drivers</td>
<td>Automatic port of drivers</td>
</tr>
<tr>
<td>No DMA support</td>
<td>DMA support</td>
</tr>
<tr>
<td>Open source drivers support</td>
<td>Binary drivers support</td>
</tr>
</tbody>
</table>
Summary
Summary

- Charm facilitates dynamic analysis of mobile device drivers
- Charm’s performance is on par with actual mobile systems
- Charm supports a broad variety of device drivers with reasonable engineering effort
Summary

- Charm facilitates dynamic analysis of mobile device drivers
- Charm’s performance is on par with actual mobile systems
- Charm supports a broad variety of device drivers with reasonable engineering effort

Charm is open source: http://trusslab.github.io/charm
Backup slides: vulnerable code snippet of CVE-2016-2061

```c
1    int i = stream_cfg_cmd->stream_src;
2    if (i >= VFE_AXI_SRC_MAX) {

3       return -EINVAL;
4  }

...  
5  memset(&axi_data->stream_info[i], 0, sizeof(struct
    msm_vfe_axi_stream));

...  
6  axi_data->stream_info[i].session_id =
    stream_cfg_cmd->session_id;
7  axi_data->stream_info[i].stream_id =
    stream_cfg_cmd->stream_id;
```
Backup slides: vulnerable code snippet of CVE-2016-2061

1     int i = stream_cfg_cmd->stream_src;
2     if (i >= VFE_AXI_SRC_MAX) {
3         return -EINVAL;
4     }

... 
5     memset(&axi_data->stream_info[i], 0, sizeof(struct
        msm_vfe_axi_stream));
... 
6     axi_data->stream_info[i].session_id =
        stream_cfg_cmd->session_id;
7     axi_data->stream_info[i].stream_id =
        stream_cfg_cmd->stream_id;
Backup slides: building exploit

1 int i = stream_cfg_cmd->stream_src;
2 if (i >= VFE_AXI_SRC_MAX) {
3   return -EINVAL;
4 }
5 ...
6 axi_data->stream_info[i].session_id =
7       stream_cfg_cmd->session_id;
8 axi_data->stream_info[i].stream_id =
9       stream_cfg_cmd->stream_id;
Backup slides: building exploit

```c
1 int i = stream_cfg_cmd->stream_src;
2 if (i >= VFE_AXI_SRC_MAX) {
3     ...
4     return -EINVAL;
5 }
...
5 memset(&axi_data->stream_info[i], 0, sizeof(struct
6     msm_vfe_axi_stream));
...
6 axi_data->stream_info[i].session_id =
7     stream_cfg_cmd->session_id;
8 axi_data->stream_info[i].stream_id =
7     stream_cfg_cmd->stream_id;
```
Backup slides: building exploit

Target object

Vulnerable object

offset
## Dynamic analysis is very useful

<table>
<thead>
<tr>
<th></th>
<th>Static analysis</th>
<th>Dynamic analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>False positives rate</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Compiler/linker bugs</td>
<td>Cannot find</td>
<td>Can find</td>
</tr>
<tr>
<td>Code obfuscation</td>
<td>Vulnerable</td>
<td>Not vulnerable</td>
</tr>
<tr>
<td>Unknown types of bugs</td>
<td>Cannot find</td>
<td>Can find</td>
</tr>
<tr>
<td>Code coverage</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
CVE-2016-3903

```c
/* in msm_csid_cmd(): */
1 for (i = 0; i < csid_params.lut_params.num_cid; i++) {
   ...
2   if (copy_from_user(vc_cfg, (void *))
       csid_params.lut_params.vc_cfg[i], sizeof(struct
       msm_camera_csid_vc_cfg)) {
       ...
3      for (i--; i >= 0; i--)
4      kfree(csid_params.lut_params.vc_cfg[i]);
5      rc = -EFAULT;
6      break;
7  }
8  csid_params.lut_params.vc_cfg[i] = vc_cfg;
9 }
10 ...
11 rc = msm_csid_config(csid_dev, &csid_params);

/* in msm_csid_cid_lut(): */
...
11 if (csid_lut_params->vc_cfg[i]->cid >=
    csid_lut_params->num_cid ||
    csid_lut_params->vc_cfg[i]->cid < 0) {
   ...
12 }
```

Is it out-of-bound access?
CVE-2016-3903

/* in msm_csid_cmd(): */
1 for (i = 0; i < csid_params.lut_params.num_cid; i++) {
   ... 
2 if (copy_from_user(vc_cfg, (void *))
        csid_params.lut_params.vc_cfg[i], sizeof(struct
        msm_camera_csid_vc_cfg)) {
   ... 
3     for (i--; i >= 0; i--)
4           kfree(csid_params.lut_params.vc_cfg[i]);
5     rc = -EFAULT;
6     break;
7 }
8     csid_params.lut_params.vc_cfg[i] = vc_cfg;
9 }
... 
10 rc = msm_csid_config(csid_dev, &csid_params);

/* in msm_csid_cid_lut(): */
... 
11 if (csid_lut_params->vc_cfg[i]->cid >=
     csid_lut_params->num_cid ||
     csid_lut_params->vc_cfg[i]->cid < 0) {
   ... 
12 }

Is it out-of-bound access?
CVE-2016-3903

Watch points

Use after free

```c
/* in msm_csid_cmd(): */
for (i = 0; i < csid_params.lut_params.num_cid; i++) {
    ...
    if (copy_from_user(vc_cfg, (void *))
        csid_params.lut_params.vc_cfg[i], sizeof(struct
        msm_camera_csid_vc_cfg)) {
        ...
        for (i , i > 0, i )
            kfree(csid_params.lut_params.vc_cfg[i]);
        rc = EFAULT;
        break;
    }
    csid_params.lut_params.vc_cfg[i] = vc_cfg;
}
...,
rc = msm_csid_config(csid_dev, &csid_params);

/* in msm_csid_cid_lut(): */
...,
if (csid_lut_params->vc_cfg[i] > cid >=
    csid_lut_params->num_cid ||
    csid_lut_params->vc_cfg[i] > cid < 0) {
    ...
}
```