Shattered Trust: When Replacement Smartphone Components Attack

Omer Shwartz (omershv@post.bgu.ac.il)
Amir Cohen (amir3@post.bgu.ac.il)
Dr. Asaf Shabtaei (shabtaia@bgu.ac.il)
Dr. Yossi Oren (yos@bgu.ac.il)

Ben-Gurion University of the Negev, Israel
Overview

- Motivation
- Attack surface
- Vulnerability discovery and demo
- Discussion, conclusions and questions
Motivation
Motivation

- Smartphone components often break.
- Many replacements are counterfeit.
- “Street corner phone repair shops” available everywhere.

Image from: Wikimedia
Research Question

What if a smartphone peripheral was malicious?

Could it attack the stock driver?

Would it affect the user’s privacy?
Attack model
Attack surface survey

Attack surface survey

- We started by doing a semi-automated analysis of the source code of 26 android smartphones.
- Drivers were catalogued to vendor and version.
- 89 different driver versions were evaluated.
## Peripheral diversity

<table>
<thead>
<tr>
<th>Phone model</th>
<th>Touchscreen</th>
<th>NFC</th>
<th>Charger IC</th>
<th>Battery</th>
<th>Wireless Charger</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC Desire 626</td>
<td>Syn4, Atm4</td>
<td>NXP_n1</td>
<td>Syn4</td>
<td>Max4</td>
<td>---</td>
</tr>
<tr>
<td>HTC One A9</td>
<td>Syn2</td>
<td>NXP_n2</td>
<td>Syn2</td>
<td>HTCA1</td>
<td>---</td>
</tr>
<tr>
<td>HTC One M9+</td>
<td>Syn3, Max3</td>
<td>NXP_n3</td>
<td>Syn3</td>
<td>HTCA2</td>
<td>---</td>
</tr>
<tr>
<td>Huawei Honor 5X</td>
<td>Syn4</td>
<td>NXP_n4</td>
<td>Hua1</td>
<td>TI3</td>
<td>---</td>
</tr>
<tr>
<td>Huawei Nexus 6P</td>
<td>Syn5</td>
<td>*NXP_n5</td>
<td>Qua1</td>
<td>Qua4</td>
<td>---</td>
</tr>
<tr>
<td>Lenovo K3 Note</td>
<td>Med1</td>
<td>---</td>
<td>Med1</td>
<td>Med4</td>
<td>---</td>
</tr>
<tr>
<td>Lenovo K5 Note</td>
<td>Med2</td>
<td>Qua1</td>
<td>Med4</td>
<td>Med4</td>
<td>---</td>
</tr>
<tr>
<td>Lenovo Vibe K4 Note</td>
<td>Med3</td>
<td>Med4</td>
<td>TEx1</td>
<td>Med4</td>
<td>---</td>
</tr>
<tr>
<td>LG Nexus 4 E960</td>
<td>Syn6</td>
<td>BrO1</td>
<td>Intc1</td>
<td>TI6</td>
<td>Ti4</td>
</tr>
<tr>
<td>LG Nexus 5</td>
<td>Syn7</td>
<td>Bro2</td>
<td>Sum5</td>
<td>TI6</td>
<td>Ti5</td>
</tr>
<tr>
<td>LG G3</td>
<td>Atm2</td>
<td>NXP_n6</td>
<td>Sum6</td>
<td>*TI4, Max3</td>
<td>Ti6</td>
</tr>
<tr>
<td>LG G4</td>
<td>Syn8</td>
<td>NXP_n7</td>
<td>Sum7</td>
<td>Max4</td>
<td>---</td>
</tr>
<tr>
<td>LG Nexus 5X</td>
<td>Syn9</td>
<td>*NXP_n5</td>
<td>Qua2</td>
<td>Qua4</td>
<td>---</td>
</tr>
<tr>
<td>Motorola Nexus 6</td>
<td>Atm3</td>
<td>Bro3</td>
<td>Sum8</td>
<td>TI6, Max4</td>
<td>Mot4</td>
</tr>
<tr>
<td>Samsung Galaxy S5</td>
<td>Atm4</td>
<td>NXP, BrO4</td>
<td>Sum9</td>
<td>TI6, Max4</td>
<td>---</td>
</tr>
<tr>
<td>Samsung Galaxy S6</td>
<td>*STM, 1</td>
<td>Sam1</td>
<td>*Max4</td>
<td>*Max6</td>
<td>TEx4</td>
</tr>
<tr>
<td>Samsung Galaxy S6 edge</td>
<td>*STM, 1</td>
<td>Sam2</td>
<td>*Max4</td>
<td>*Max6</td>
<td>TEx5</td>
</tr>
<tr>
<td>Samsung Galaxy S6 edge+</td>
<td>STM, 2</td>
<td>Sam3</td>
<td>*Max4</td>
<td>*Max6</td>
<td>TEx6</td>
</tr>
<tr>
<td>Samsung Galaxy S7</td>
<td>Sam1</td>
<td>NXP_n9, Sam4</td>
<td>Max2</td>
<td>Max4</td>
<td>IDT6, 1</td>
</tr>
<tr>
<td>Samsung Galaxy S7 edge</td>
<td>Sam2</td>
<td>NXP_n10, Sam5</td>
<td>Max3</td>
<td>Max4</td>
<td>IDT6, 2</td>
</tr>
<tr>
<td>Sony Xperia Z2</td>
<td>Max2</td>
<td>NXP_n11</td>
<td>Qua3</td>
<td>*TI4, Max9</td>
<td>---</td>
</tr>
<tr>
<td>Sony Xperia M4 Aqua</td>
<td>Syn, 10</td>
<td>NXP_n12</td>
<td>Qua4</td>
<td>Sum4</td>
<td>---</td>
</tr>
<tr>
<td>Sony Xperia M5</td>
<td>Syn, 11</td>
<td>NXP_n13</td>
<td>Med2</td>
<td>Med4</td>
<td>---</td>
</tr>
<tr>
<td>Sony Xperia Z5</td>
<td>Syn, 12, Atm5</td>
<td>NXP_n14</td>
<td>Sum, 10</td>
<td>TI6</td>
<td>---</td>
</tr>
<tr>
<td>Sony Xperia E5</td>
<td>Med4</td>
<td>Med2</td>
<td>Med3</td>
<td>Med5</td>
<td>---</td>
</tr>
<tr>
<td>Sony Xperia XA</td>
<td>Med5</td>
<td>Med3</td>
<td>Qua1</td>
<td>Med6</td>
<td>---</td>
</tr>
</tbody>
</table>
Attack surface survey results

- Only three drivers were used in two phone models.
- Only two drivers were used on three or more phone models.
- Most of the drivers were unique to their respective device.

Driver Diversity

- Shared drivers
- Unique drivers

94.4%
Percentage of Driver related android CVEs is on the rise.
Attack surface survey insights

- A codebase that diverse is ought to contain bugs and vulnerabilities.
- Bugs are easy to find (more ahead...)

Image: A toy frog holding a smartphone.
Bug Hunting

- We started looking for device driver bugs that may be exploited by the component.
- In less than two hours (!), we found exploitable buffer and heap overflows in a touchscreen module.
- Further analysis revealed dozens more potential issues in multiple drivers.
- The issues found were reported, acknowledged (CVE-2017-0650) and patched by Google.
Actual Exploitation
Workbench
static void synaptics_rmi4_sensor_report(struct synaptics_rmi4_data *rmi4_data,
    bool report)
{
    int retval;
    unsigned char data[MAX_INTR_REGS + 1];
    unsigned char *intr = &data[1];
    struct synaptics_rmi4_f01_device_status status;
    struct synaptics_rmi4_fn *fhandler;
    struct synaptics_rmi4_exp_fhandler *exp_fhandler;
    struct synaptics_rmi4_device_info *rmi;

    rmi = &(rmi4_data->rmi4_mod_info);

    /*
     * Get interrupt status information from F01 Data1 register to
     * determine the source(s) that are flagging the interrupt.
     */
    retval = synaptics_rmi4_reg_read(rmi4_data,
        rmi4_data->f01_data_base_addr,
        data,
        rmi4_data->num_of_intr_regs + 1);
CVE-2017-0650

Get data from device...

....

```c
static int synaptics_rmi4_query_device(struct synaptics_rmi4_data *rmi4_data) {
    if (!list_empty(&rmi->support_fn_list)) {
        list_for_each_entry(fhandler, &rmi->support_fn_list, link) {
            if (fhandler->num_of_data_sources) {
                rmi4_data->intr_mask[fhandler->intr_reg_num] =
                    fhandler->intr_mask;
            }

            unsigned char no_sleep_setting;
            unsigned char intr_mask[0x00000008];
            unsigned char *button_fxxr_mapping;
            unsigned short num_of_intr_regs;
            unsigned short f01_query_base_addr;
            unsigned short f01_cmd_base_addr;
            unsigned short f01_ctrl_base_addr;
            unsigned short f01_data_base_addr;
            unsigned int firmware_id;
            unsigned int config_id;
            int irq;
        }
    }
}
```
ARM64 ROP Attack

Return flow

Original stack

Overwritten stack

Malicious flow

Local Variables of CF

Padding

Addr. of memory word to change

Padding

Buffer overflow

Local Variables of CF-1

Padding

Addr. of do_exit()
**ARM64 ROP Attack**

**Assembly**

**Gadget 1**
- `ldp x19, x20, [sp, #0x10]
- `ldp x29, x30, [sp], #0x20
- `ret`

**Gadget 2**
- `mov x2, x19`
- `mov x0, x2`
- `ldp x19, x20, [sp, #0x10]
- `ldp x29, x30, [sp], #0x30`
- `ret`

**Gadget 3**
- `mov x0, x19`
- `mov x1, x20`
- `blr x2`
- `ldp x19, x20, [sp, #0x10]
- `ldr x21, [sp, #0x20]
- `ldp x29, x30, [sp], #0x30`
- `ret`

**Pseudo code***

Load SP+0x10 to X19 and X20
Set return address to SP+0x08
Increment SP by 0x20

Assign X2 := X19
Load SP+0x10 to X19 and X20
Set return address to SP+0x08
Increment SP by 0x30

Assign X0 := X19
Assign X1 := X20
Call X2(X0, X1)
Set return address to SP+0x08
Increment SP by 0x30

* The code omits non relevant instructions
Vulnerabilities

- Vulnerabilities such as CVE-2017-0650 are easy to find!
- Another vulnerability was found in a different touchscreen driver by another manufacturer. This vulnerability was proved in a POC level. (Pending Responsible Disclosure)
Touch Logging & Touch Injection

- In addition to exploiting the kernel, the touchscreen can abuse its known capabilities.
- The touchscreen or a component on the touchscreen bus can record user touches.
- Injection of touch events can also be done without any user interaction.
Proofs of Concept

- Malicious Software Installation
  https://youtu.be/83VMVrcEOCM
- Take Picture and Send Via Email
  https://youtu.be/WS4NChPjaaY
- Replace URL with Phishing URL
  https://youtu.be/XZujd42eYek
- Log and Exfiltrate Screen Unlock Pattern
  https://youtu.be/fY58zoadqMA
- Complete Phone Compromise
  https://youtu.be/sDfD5fJfiNc
Discussion

- In smartphones, device drivers are very diverse, leading to many potential vulnerabilities.
- The device drivers are trusted by the kernel, and the peripherals are trusted by the drivers.
- We showed how a peripheral may exploit that trust.
- Detection of attacks may be impossible with current tools and design.
Conclusions

● Attacks of this sort are practical and may be highly effective.
● Consumers have no tools to help them detect such attacks.
● Countermeasures, such as a physical interface firewall may prove effective.
● There is much more left to explore in the field of malicious hardware components.
Thank you

- Omer Shwartz (omershv@post.bgu.ac.il)
- Amir Cohen (amir3@post.bgu.ac.il)
- Dr. Asaf Shabtai (shabtaia@bgu.ac.il)
- Dr. Yossi Oren (vos@bgu.ac.il)