Detecting API Post-Handling Bugs Using Code and Description in Patches

Miaoqian Lin, Kai Chen, Yang Xiao

INSTITUTE OF INFORMATION ENGINEERING, CAS
API Post-Handling (APH)

@@ -255,8 +255,8 @@ static int intel_rapi_tpmi_probe(struct aur
    }
    trp->base = devm_ioremap_resource(rdev->auxdev->dev, res);
-   if (!trp->base) {
-     ret = -ENOMEM;
+   if (IS_ERR(trp->base)) {
+     ret = PTR_ERR(trp->base);
        goto err;
    }

@@ -4152,8 +4152,10 @@ static struct usb_hcd *oxu_create
       ret = usb_add_hcd(hcd, irq, IRQF_SHARED);
-      if (ret < 0)
+      if (ret < 0) {
+        usb_put_hcd(hcd);
+        return ERR_PTR(ret);
+      }
    }

    device_wakeup_enable(hcd->self.controller);
    return hcd;

API Post-handling is error-prone and detecting APH bugs is vital
How to detect?

Document of **kobject_init_and_add** in Linux kernel:

“If this function returns an error, **kobject_put()** must be called to properly clean up the memory associated with the object”

APH Specifications are the key for detecting APH bugs
Limitations of Previous Work

- Incomplete or Unclear API Documents
- Wrong API Usages

Failure to extract specifications leads to **uncovered bugs!**
APH Bug Patches

APH bug patches are good source for APH specifications

```c
@@ -255,8 +255,8 @@ static int intel_rapi_tomi_probe(struct au
                ret = -ENOMEM;
            } if (IS_ERR(trp->base)) {  
            ret = PTR_ERR(trp->base);
                goto err;
            }

@@ -4152,8 +4152,10 @@ static struct usb_hcd *oxu_cre
                   ret = usb_add_hcd(hcd, lrq, IRQF_SHARED);
                } if (ret < 0) {  
                usb_out_hcd(hcd);
            return ERR_PTR(ret);
        }

    device_wakeup_enable(hcd->self.controller);
    return hcd;
```
Insights

Document of `kobject_init_and_add` in Linux kernel

“If this function returns an error, `kobject_put()` must be called to properly clean up the memory associated with the object.”

- **Target API** requires post-operation
- **Post-operation** handles target API’s effects
- **Critical variable** affected by target API
- **Path condition** indicates when to apply post-operation

Define APH specifications as **four-tuples** with key elements
Motivating Example

Target API: `usb_create_hcd`

Critical variable: `hcd`

Post-operation: `usb_put_hcd`

Path conditions

Patch contains key elements defined in APH specification
"usb: oxu210hp-hcd: Fix memory leak in oxu_create
usb_create_hcd will alloc memory for hcd, and we should call usb_put_hcd to free it when adding fails to prevent memory leak."

Extract specifications using **code and description** in patches
Overview of APHP: APH bugs detector using patches
"usb: oxu210hp-hcd: Fix memory leak …

*usb_create_hcd* will alloc memory for *hcd*, and we should call *usb_put_hcd* to free it when adding fails to prevent memory leak."

```c
01 @ -4152,8 +4152,10 @@ static struct oxu_create(...)
02     struct usb_hcd *hcd;
03 
04     hcd = usb_create_hcd(&oxu_hc_driver, ...);
05     if (!hcd)
06         return ERR_PTR(-ENOMEM);
07     oxu = hcd_to_oxu(hcd);
08 
09     ret = usb_add_hcd(hcd, irq, IRQF_SHARED);
10     if (ret < 0)
11         if (ret < 0) {
12             usb_put_hcd(hcd);
13             return ERR_PTR(ret);
14         }
15     return hcd;
```

**Post-operation**

```
usb_put_hcd
```

**Target API**

```
usb_create_hcd
```

**Critical variable**

```
hcd (return value)
```

**Path conditions**

**AST difference**

**Path-level difference**

**Combine code and textual semantics**
Bug Detection: Partial path-sensitive analysis

APH specification

Target API
usb_create_hcd

Path conditions
get callers of target API

get paths to be checked

Critical variable
hcd(return value)

focus on the variable

Post-operation
usb_put_hcd

check if the operation exist

If not exist, report it

APH specification-based graph (ASG)

Extensive code with numerous paths

Analyze partial paths

ASG generation

Path verification

ASG of function dwc2_hcd_init
Evaluation Results: APHP Effectiveness

• Dataset
  ◦ Four popular open-source programs: Linux kernel, QEMU, Git and Redis

• Results
  ◦ Detected 410 new bugs, 216 confirmed by developers
  ◦ Bugs exist for a long time, on average more than 5 years
  ◦ Various security impacts such as resource leaks, NULL pointer dereference.
Evaluation Results: Comparisons with SOTAs

- Comparators
  - Patch-based: VUDDY[S&P’17], MVP[Security’20]
  - Document-based: Advance[CCS’20]
  - Source code-based: IPPO[CCS’21]

These tools fail to detect most APH bugs found by APHP
Evaluation Results: Ablation study

- Contribution of patch descriptions

<table>
<thead>
<tr>
<th>Approach</th>
<th>Specification extraction</th>
<th>Bug detection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>APHP</td>
<td>89%</td>
<td>89%</td>
</tr>
<tr>
<td>APHP−</td>
<td>26.5%</td>
<td>94%</td>
</tr>
</tbody>
</table>

Patch descriptions enhance the precision

- Contribution of APH specification-based graph (ASG)

<table>
<thead>
<tr>
<th></th>
<th>Num of nodes</th>
<th>Num of paths</th>
<th>Avg. path length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASG</td>
<td>14.4</td>
<td>45.4</td>
<td>8.7</td>
</tr>
<tr>
<td>CFG</td>
<td>106.0</td>
<td>2942.2</td>
<td>61.6</td>
</tr>
<tr>
<td>% Reduction</td>
<td>86.4%</td>
<td>98.5%</td>
<td>85.9%</td>
</tr>
</tbody>
</table>

ASG reduce the amount of code analyzed
Key Findings from Detected APH Bugs

- Error-prone APIs 😞
- Implicit APH specifications 😞
- Specifications deviating from default conventions 😞

<table>
<thead>
<tr>
<th>API Description</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF device node getter</td>
<td>of_find_node_by_name&lt;br&gt;of_find_node_by_path&lt;br&gt;of_find_node_by_phandle&lt;br&gt;of_get_child_by_name&lt;br&gt;of_find_matching_node_and_match&lt;br&gt;of_get_next_parent&lt;br&gt;of_graph_get_remote_node&lt;br&gt;of_get_next_child&lt;br&gt;of_cpu_device_node_get</td>
</tr>
<tr>
<td>API Description</td>
<td>API</td>
</tr>
<tr>
<td></td>
<td>of_parse_phandle&lt;br&gt;of_find_matching_node&lt;br&gt;of_find_compatible_node</td>
</tr>
</tbody>
</table>
Conclusion: APHP

- Novel approach to detect APH bugs using code and descriptions in patches
- Detect 410 new bugs in popular programs such as Linux Kernel, Qemu
- Valuable knowledge gain for bug hunters and developers
- https://github.com/Yuuoniy/APHP
Thank You

Q&A

linmiaoqian@iie.ac.cn