Understanding and Detecting Disordered Error Handling with Precise Function Pairing

Qiushi Wu, Aditya Pakki, Navid Emamdoost, Stephen McCamant, and Kangjie Lu
Two parts of error related code
Error related code == cleaning up + error handling

- Cleanup operations
  - In the error code, programs often perform “cleanup” operations

```c
/* drivers/media/platform/s5p-g2d/g2d.c */
static int g2d_probe (struct platform_device *pdev) {
    ...
    ret = v4l2_device_register(&pdev->dev, &dev->v4l2_dev);
    if (ret)
        goto unprep_clk_gate;
    vfd = video_device_alloc();
    if (!vfd) {
        ret = -ENOMEM;
        goto unreg_v4l2_dev;
    }
    ...
    unreg_v4l2_dev:
    v4l2_device_unregister(&dev->v4l2_dev);
    unprep_clk_gate:
    clk_unprepare(dev->gate);
    return ret;
}
```
Error related code == cleaning up + error handling

- **Cleanup operations**
- **Handling the errors**
  - Besides cleanup operations, the program will handle the error through return error, print out error-message, stop the execution ...

```c
/* drivers/media/platform/s5p-g2d/g2d.c */
static int g2d_probe (struct platform_device *pdev) {
  ...
  ret = v4l2_device_register(&pdev->dev, &dev->v4l2_dev);
  if (ret)
    goto unprep_clk_gate;
  vfd = video_device_alloc();
  if (!vfd) {
    ret = -ENOMEM;
    goto unreg_v4l2_dev;
  }
  ...
unreg_v4l2_dev:
  v4l2_device_unregister(&dev->v4l2_dev);
unprep_clk_gate:
  clk_unprepare(dev->gate);
  return ret;
}
```

Handling the errors
Previous works

- Most research focused on “handling” but not “cleanup”
  - Error-propagation related bugs
  - Error check related bugs
  - Error-handling severity level
Previous works

- Most research focused on “handling” but not “cleanup”

- Some attempted to detect only missing cleanup operations
  - Missing release, missing refcount decrease, ...
Previous works

- Most research focus on “handling” but not “cleanup”

- Some research attempted to detect only missing cleanup operations

Limitations:
1. Could only handle obvious cleanup operations
   - Commonly used release functions
   - Function name based approach
2. None of them can comprehensively and systematically detect wrong cleanup operations.
Critical issues with cleanup operations

- Calling cleanup operations inadequately
  - Missing release, missing refcount decrease ...
Critical issues with cleanup operations

- Calling cleanup operations inadequately

- Calling cleanup operations redundantly
  - Double-free, double-unlock, ...
Critical issues with cleanup operations

- Calling cleanup operations inadequately
- Calling cleanup operations redundantly

- Calling cleanup operations in an incorrect order
  - Use-after-free, double-free, ...
Critical issues with cleanup operations

- Calling cleanup operations inadequately
- Calling cleanup operations redundantly
- Calling cleanup operations in an incorrect order

We refer to such bugs that caused by incorrect cleanup operations as *Disordered Error Handling* (DiEH).
Our goal:

Systematically study and detect the DiEH bugs
How to clean up correctly?
Clean up correctly == Correctly use function pairs

Function pair = leader function + follower function

- **Leader function**
  - initiates an operation against some resources

- **Follower function: typically used as cleanup operations**
  - recovers the resources that are initiated by leader functions
Cleanup correctly == Correctly use function pairs

```c
1 static int foo() {
2     // allocate memory
3     mem = kmalloc(...);
4     if (!mem)
5         return -1;
6 ...
7     if (err) {
8         // free previously allocated memory
9         kfree(mem);
10        ...
11     }
12 ...
13 }
```

Function pairs

<table>
<thead>
<tr>
<th>Leader</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmalloc</td>
<td>kfree</td>
</tr>
</tbody>
</table>
Cleanup correctly == Correctly use function pairs

```
static int foo() {
  // allocate memory
  mem = kmalloc(...);
  if (!mem)
    return -1;
  ...
  if (err) {
    // free previously allocated memory
    kfree(mem);
    ...
  }
}
```

The follower functions typically perform the cleanup operation.

<table>
<thead>
<tr>
<th>Leader</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmalloc</td>
<td>kfree</td>
</tr>
</tbody>
</table>
The first step and the key step of identifying DiEH bugs is correctly pairing function.
How to collect function pairs?
Intuitive approaches

● Name-based approach
  ○ e.g., "alloc" --- ‘free’, “inc” -- “dec”

● NLP based approach
  ○ Checking the function description

● Pattern-mining based approach

Limitations:
1. High FPs & FNs
2. Cannot handling customized functions well
The incremental error-handling structure --- EH stacks

Function pairs

<table>
<thead>
<tr>
<th>Leader</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>v4l2_device_register</td>
<td>v4l2_device_unregister</td>
</tr>
<tr>
<td>video_device_alloc</td>
<td>video_device_release</td>
</tr>
<tr>
<td>video_register_device</td>
<td>video_unregister_device</td>
</tr>
</tbody>
</table>

A stack-like (LIFO) structure: The first called leader function will be handled at last.
The incremental error-handling structure --- EH stacks

1 /* drivers/media/platform/s5p-g2d/g2d.c */
2 static int g2d_probe (struct platform_device *pdev) {
3    ...
4    ret = v4l2_device_register(&pdev->dev, &dev->v4l2_dev);
5    if (ret)
6      goto unprep_clk_gate;
7    vfd = video_device_alloc();
8    if (!vfd) {
9      ret = -ENOMEM;
10     goto unreg_v4l2_dev;
11    }
12    ...
13    ret = video_register_device(vfd, VFL_TYPE_VID, 0);
14    if (ret)
15      goto rel_vdev;
16    ...
17    dev->m2m_dev = v4l2_m2m_init(&g2d_m2m_ops);
18    if (IS_ERR(dev->m2m_dev))
19      goto unreg_video_dev;
20    ...
21    unreg_video_dev:
22    video_unregister_device(dev->vfd);
23    rel_vdev:
24    video_device_release(vfd);
25    unreg_v4l2_dev:
26    v4l2_device_unregister(&dev->v4l2_dev);
27    unprep_clk_gate:
28    clk_unprepare(dev->gate);
29    ...
30 }

Error handling(EH) stack: represent the functions in one path.

E.g.,:

4 v4l2_device_register
7 video_device_alloc
26 v4l2_device_unregister

The functions in line 4 -> 7 -> 26 -> … form a EH stack
Collecting function pairs based on the EH deltas

Consider two execution paths

<table>
<thead>
<tr>
<th>EH Stack</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4-&gt;7-&gt;26-&gt;...</td>
<td>#1</td>
</tr>
</tbody>
</table>

25 `unreg_v4l2_dev:`
26 `v4l2_device_unregister(&dev->v4l2_dev);`
27 `unprep_clk_gate:`
28 `clk_unprepare(dev->gate);`
29 `...`
30 `}`
Collecting function pairs based on the EH deltas

1 /* drivers/media/platform/s5p-g2d/g2d.c */
2 static int g2d_probe (struct platform_device *pdev) {
3     ...
4     ret = v4l2_device_register(&pdev->dev, &dev->v4l2_dev);
5     if (ret)
6         goto unprep_clk_gate;
7     vfd = video_device_alloc();
8     if (!vfd) {
9         ret = -ENOMEM;
10        goto unreg_v4l2_dev;
11    }
12    ...
13    ret = video_register_device(vfd, VFL_TYPE_VID, 0);
14    if (ret)
15        goto rel_vdev;
16 }
17 ...
18
19 rel_vdev:
20    video_device_release(vfd);
21 unreg_v4l2_dev:
22    v4l2_device_unregister(&dev->v4l2_dev);
23 unprep_clk_gate:
24    clk_unprepare(dev->gate);
25    ...
26 }

Consider two execution paths

<table>
<thead>
<tr>
<th>EH Stack</th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-&gt;7-&gt;26-&gt;...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-&gt;7-&gt;13-&gt;24-&gt;26-&gt;...</td>
<td>#2</td>
<td>#2</td>
</tr>
</tbody>
</table>
Collecting function pairs based on the EH deltas

Consider two execution paths

**EH Stack**

| #1 | 4->7->26->... |
| #2 | 4->7->13->24->26->... |

**EH delta**: Subtract two EHs

(only consider successfully executed functions)

7(video_device_alloc) - 24(video_device_release)
Collecting function pairs based on the EH deltas

Consider two execution paths

<table>
<thead>
<tr>
<th>EH Stack</th>
<th>#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-7-26-...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EH Stack</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-7-13-24-26-...</td>
<td></td>
</tr>
</tbody>
</table>

**EH delta**: Subtract two EHs
(only consider successfully executed functions)

7(video_device_alloc)-24(video_device_release)

EH deltas = Function pairs
Detecting DiEH bugs based on function pairs
Detecting DiEH bugs based on function pairs

- **Detecting DiEH cases**
  - Check leader and follower functions in every EH stacks
  - Detecting the disordered usages of the function pairs
  - E.g., [alloc1, alloc2, free1, free2]

- **Detecting DiEH bugs from DiEH cases**
  - Eliminating infeasible paths
  - Eliminating harmless DiEH cases by dependency reasoning
  - Ranking reported bugs through cross-validation
Evaluation
## Performance

<table>
<thead>
<tr>
<th>Target programs</th>
<th>Lines of code</th>
<th>Num of IR files</th>
<th>Time for pairing</th>
<th>Time for detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel</td>
<td>17.7M</td>
<td>18071</td>
<td>48 min</td>
<td>10h16min</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>4.8M</td>
<td>1483</td>
<td>10 min</td>
<td>2h28min</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>450K</td>
<td>1903</td>
<td>53 sec</td>
<td>11min</td>
</tr>
</tbody>
</table>
Detected function pairs and DiEH bugs

<table>
<thead>
<tr>
<th>Target programs</th>
<th>Num of func pairs</th>
<th>Num of DiEH bugs</th>
<th>Security impacts of DiEH bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel</td>
<td>7.5K (precision &gt; 90%, recall &gt; 60%)</td>
<td>234 (48% FPR)</td>
<td>Double-free, use-after-free, refcount leak, memory leak, double unlock</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>416</td>
<td>2</td>
<td>Memory leak</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>323</td>
<td>3</td>
<td>Double-free, memory leak</td>
</tr>
</tbody>
</table>
Conclusions

● Cleanup operations are commonly misused.

● We proposed DiEH bugs that are caused by improper cleanup operations.

● We proposed an EH-delta based pairing mechanism.

● Identified a large number of function pairs, critical DiEH bugs (cause double-free, UAF, memory leak, …)