
Dinghao Liu  Qiushi Wu  Shouling Ji  Kangjie Lu
Zhenguang Liu  Jianhai Chen  Qinming He

CCS 2021
Background
Security operations are widely used in large-scale programs.
Background

- **Missing security operations could lead to many security issues**

  - Security check
  - Resource release
  - Refcount decrement
  - Lock/unlock

  - Memory corruption, privilege escalation, DoS ...
  - Memleak, DoS ...
  - Power consumption, privilege escalation, DoS ...
  - Deadlock, data race, ...

61% vulnerabilities in the NVD are caused by missing security operations!
How to determine whether the missed security operations are indeed necessary?
Cross-checking

• High level idea
  • Collect a substantial number of similar code pieces.
  • Check the behaviors of security operations across the similar code pieces.
  • The majority is correct.

• Limitations
  • Sufficient code pieces are required to enable cross-checking.
  • The granularity of code piece is hard to control.
  • The majority is not always correct.
• A security operation usually focuses on one critical object.

• The similarity of code pieces should be based on the particular object.
  • Object-based similar path pair.
  • It takes only 2 paths to enable inconsistency analysis and bug detection.
  • Fine-grained and robust.
System Design
Overview

**IPPO (Inconsistent Path Pairs as a bug Oracle)**

- Statically detect bugs caused by missed security operations.
- LLVM-based intra-procedural static analyzer.

---

**Env Preparation**

- Source Code → Compile → LLVM IRs
- Call graph generation
  - Loop unrolling
  - Global call graph
  - Control-flow graphs

**Program Analysis**

- **Security operation detection**
  - Security checks
  - Resource alloc/release
  - Reference count inc/dec
  - Lock/unlock
- Object extraction
- Object-based similar-path pair collection

**Differential Checking**

- Check missed security operations in path pairs
- Suggest potential bugs and generate bug reports
- Bug Reports
## Security Operation Detection

### Security check

**FILE:** drivers/dma/dma-jz4780.c

---

854. jzdma = devm_kzalloc(dev, struct_size(jzdma, chan, soc_data->nb_channels), GFP_KERNEL);
855. if (!jzdma)
856. return -ENOMEM;
857. return -ENOMEM;

---

### Lock/unlock

**FILE:** arch/x86/platform/uv/uv_irq.c

---

161. mutex_lock(&uv_lock);
175. mutex_unlock(&uv_lock);

---

### Refcount inc/dec

**FILE:** drivers/net/ethernet/intel/e1000e/ethtool.c

---

161. pm_runtime_get_sync(netdev->dev.parent);
175. pm_runtime_put_sync(netdev->dev.parent);

---

### Resource alloc/release

**FILE:** drivers/platform/x86/dell/dell-wmi-sysman/biosattr-interface.c

---

124. buffer = kzalloc(buffer_size, GFP_KERNEL);
141. kfree(buffer);
Security check

<table>
<thead>
<tr>
<th>FILE: drivers/dma/dma-jz4780.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>854. jzdma = devm_kzalloc(dev, struct_size(jzdma, chan, soc_data-&gt;nb_channels), GFP_KERNEL);</td>
</tr>
<tr>
<td>856. if (!jzdma) {</td>
</tr>
<tr>
<td>857. return -ENOMEM;</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Lock/unlock

<table>
<thead>
<tr>
<th>FILE: arch/x86/platform/uv/uv_irq.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>161. mutex_lock(&amp;uv_lock);</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>175. mutex_unlock(&amp;uv_lock);</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Recount inc/dec

<table>
<thead>
<tr>
<th>FILE: drivers/net/ethernet/intel/e1000e/ethtool.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>161. pm_runtime_get_sync((netdev-&gt;dev.parent);</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>175. pm_runtime_put_sync((netdev-&gt;dev.parent);</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Resource alloc/release

<table>
<thead>
<tr>
<th>FILE: drivers/platform/x86/dell/dell-wmi-sysman/biosattr-interface.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>124. buffer = kzalloc(buffer_size, GFP_KERNEL);</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>141. kfree(buffer);</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Object-based Similar Path Pair

- **Rules for constructing object-based similar path pair (OSPP)**
  - **Rule 1**
    - The two paths start at the same block and end at the same block in CFG.
  - **Challenge**: path explosion in large functions
Object-based Similar Path Pair

- Rules for constructing object-based similar path pair (OSPP)

  - **Rule 1**
    - The two paths start at the same block and end at the same block in CFG.

  - **Challenge:** path explosion in large functions

Root cause: The redundant common messages
Object-based Similar Path Pair

- **Rules for constructing object-based similar path pair (OSPP)**

  - **Rule 1**
    - The two paths start at the same block and end at the same block in CFG.

  - **Challenge:** path explosion in large functions

  - **Our solution:** reduced similar path (RSP)
    - Only collect paths that share no common basic blocks besides the start block and the end block.
Rules for constructing object-based similar path pair (OSPP)

• Rule 2
  • The object has the same state in two paths.
Object-based Similar Path Pair

- Rules for constructing object-based similar path pair (OSPP)

- Rule 3
  - The object has the same *security operation-influential operations* against the object.

<table>
<thead>
<tr>
<th>Security operation</th>
<th>SO-influential operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security check</td>
<td>Function calls, arithmetic and memory operations after the object (checked variable)</td>
</tr>
<tr>
<td>Resource alloc/release</td>
<td>Resource propagation</td>
</tr>
<tr>
<td>Refcount</td>
<td>Reference counter adjustment</td>
</tr>
<tr>
<td>Lock/unlock</td>
<td>Lock state adjustment</td>
</tr>
</tbody>
</table>
Object-based Similar Path Pair

- Rules for constructing object-based similar path pair (OSPP)

- Rule 4
  - The two paths have the same set of pre- and post-conditions against the object.

Pre-condition:
The branch condition of a path pair.

Post-condition:
The return values of a path pair.

Must be object irrelevant

A pair of normal paths or a pair of error handing paths
Rules for constructing object-based similar path pair (OSPP)

- **Challenge:** how to efficiently collect path pairs that satisfy the post-condition of Rule 4?

- **Our solution:** graph partitioning
  - Divide the CFG into 2 sub-CFGs:
  - Paths in each sub-CFG share the same return value

- Return value-based graphs (RVGs)
Rules for constructing object-based similar path pair (OSPP)

- Generating return value-based graphs

![Diagram showing normal and error handing RVG graphs]
Case Study
A Double-free Bug Found by IPPO

```
/* sound/pci/echoaudio/echoaudio.c */
static int snd_echo_resume(struct device *dev)
{
    struct echoaudio *chip = dev_get_drvdata(dev);
    struct comm_page *commpage, *commpage_bak;
    ...
    commpage = chip->comm_page;
    commpage_bak = kmemdup(commpage, sizeof(*commpage), GFP_KERNEL);
    if (commpage_bak == NULL)
        return -ENOMEM;
    err = init_hw(chip, chip->pci->device, chip->pci->subsystem_device);
    if (err < 0)
    {
        kfree(commpage_bak);
        dev_err(dev, "resume_init_hw err=%d\n", err);
        snd_echo_free(chip);
        return err;
    }
    err = restore_dspettings(chip);
    chip->pipe_alloc_mask = pipe_alloc_mask;
    if (err < 0)
    {
        kfree(commpage_bak);
        return err;
    }
    kfree(commpage_bak);
    ...
    if (request_irq(...))
    {
        dev_err(chip->card->dev, "cannot grab irq\n");
        snd_echo_free(chip);
        return -EBUSY;
    }
    ...
    return 0;
}
```

Resource allocation of object `chip`

Resource release of object `chip`

Missing release against `chip` in the error handing path

Resource release of object `chip`
Workflow of IPPO

- Security operation detection & error edges identification
Workflow of IPPO

- Security operation detection & error edges identification

LLVM IRs

Control flow graph
Workflow of IPPO

- Identify error edges

LLVM IRs

Control flow graph
Workflow of IPPO

- Generate return value-based graphs

Control flow graph

Normal RVG

Error handling RVG

Stop
Workflow of IPPO

- Collect reduced similar paths (RSPs)

Error handling RVG

Reduced similar paths:
- RSP 1: 3 → 5 → 8 → 9
  - 3 → 6 → 9
- RSP 2: 2 → 3 → 5 → 8 → 9
  - 2 → 4 → 9
- RSP 3: 1 → 2 → 3 → 5 → 8 → 9
  - 1 → 9
Workflow of IPPO

- OSPP rules checking & differential checking

Reduced similar paths

Check Rule 2, Rule 3 and the pre-condition of Rule 4

Differential checking

RSP 1:
- Only one path frees chip

RSP 2:
- Both paths free chip

RSP 3:
- Only one path frees chip

Bug reports
Evaluation
Experimental Setting

Environment

• Use a laptop with 16 GB RAM and Intel Core i7 CPU with six cores
• Use Clang-9.0

Targets

• Linux kernel v5.8
• FreeBSD 12
• OpenSSL 3.0.0-alpha6
• PHP 8.0.8
Bug Findings

- Only focus on missed return value checks, refcount decrement, resource release, and unlock.
- Complete the whole analysis in 2 hours.

Table 2: Bug detection results of IPP0 in the four systems. The R and T in the table indicate the reported bugs and true bugs, respectively.

<table>
<thead>
<tr>
<th>Bug type</th>
<th>Linux</th>
<th>OpenSSL</th>
<th>FreeBSD</th>
<th>PHP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>T</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>Missing check</td>
<td>101</td>
<td>11</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Missing release</td>
<td>244</td>
<td>68</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Refcount leak</td>
<td>345</td>
<td>181</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Missing unlock</td>
<td>29</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>719</td>
<td>266</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

- 275 valid bugs.
- 161 are previous unknown.
- 136 have been fixed by our patches or reports.
Comparison with Other Tools

- **Comparison with cross-checking tools**

<table>
<thead>
<tr>
<th>Bug type</th>
<th>IPPO</th>
<th>FICS</th>
<th>Crix</th>
<th>APISan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing check</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Missing release</td>
<td>75</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Refcount leak</td>
<td>181</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Missing unlock</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>275</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

IPPO is a promising complementation with existing tools.

- **Comparison with pairing analysis tool: HERO**

<table>
<thead>
<tr>
<th>Bug types</th>
<th>Bugs in v5.3</th>
<th>HERO Results</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Leak</td>
<td>55</td>
<td>2</td>
<td>3.6%</td>
</tr>
<tr>
<td>Refcount Leak</td>
<td>112</td>
<td>82</td>
<td>73.2%</td>
</tr>
<tr>
<td>Missing unlock</td>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>UAF/DF</td>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>176</td>
<td>84</td>
<td>47.7%</td>
</tr>
</tbody>
</table>
Limitation & Discussion

- **False positives**
  - Unexpected pre-condition.
  - Imprecise data-flow analysis.
  - Imperfect error path analysis.
  - Imperfect security operation detection.
  - ......

- **False negatives**
  - Imperfect security operation detection.
  - ......

- **Supporting inter-procedural analysis**
  - Model inter-procedural object-based similar paths.
Missing security operations is common in real-world programs, and could cause various security issues.

We presented IPPO: a framework to detect missed security operations.

- Object-based similar path pairs.
- Reduced similar path.
- Return value-based sub-CFG.

We evaluated IPPO on 4 real-world programs.

- Find 161 new bugs.
- IPPO could effectively detect bugs that missed by existing tools.