浙江大学 Zhejiang University

UNIVERSITY OF MINNESOTA Driven to Discover®

浙江大学 NESA Lab

Detecting Missed Security Operations Through Differential Checking of Object-based Similar Paths

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

CCS 2021

Background

Background

Reference count operation

Security check

Initialization

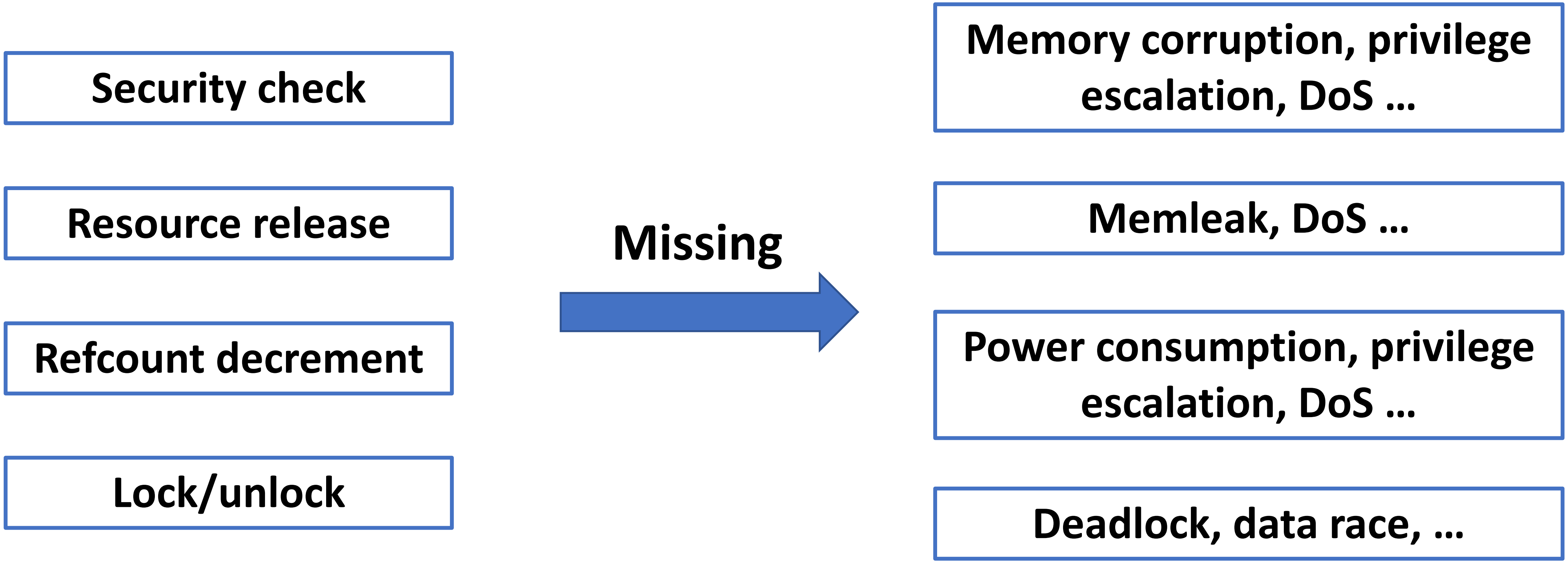
Lock

Security operations are widely used in large-scale programs

Resource release

Background

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



Security check

Resource release

Refcount decrement

Lock/unlock

Missing

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!

Background

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.

Insight

- **A security operation usually focuses on one critical object.**

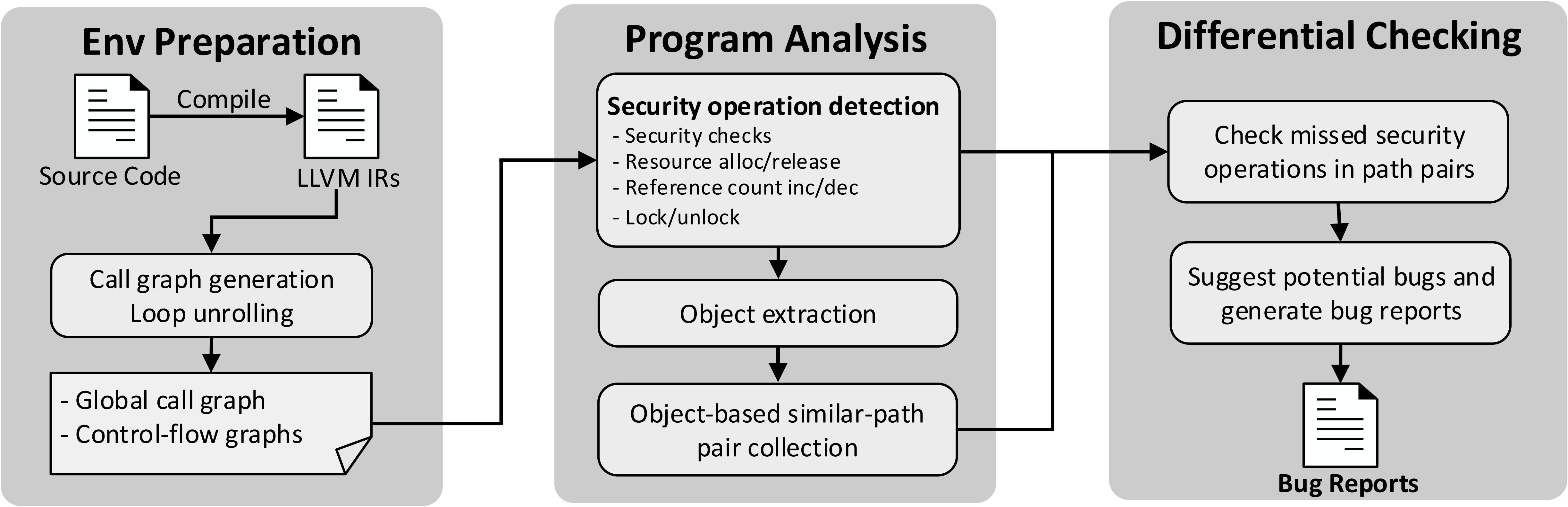
- **The similarity of code pieces should be based on the particular object.**
  - Object-based similar path pair.
  - It takes only 2 paths to enable inconsistency analysis and bug detection.
  - Fine-grained and robust.

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.



Security Operation Detection

Security check

FILE: drivers/dma/dma-jz4780.c

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

Extracting Objects

Security check

FILE: drivers/dma/dma-jz4780.c

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

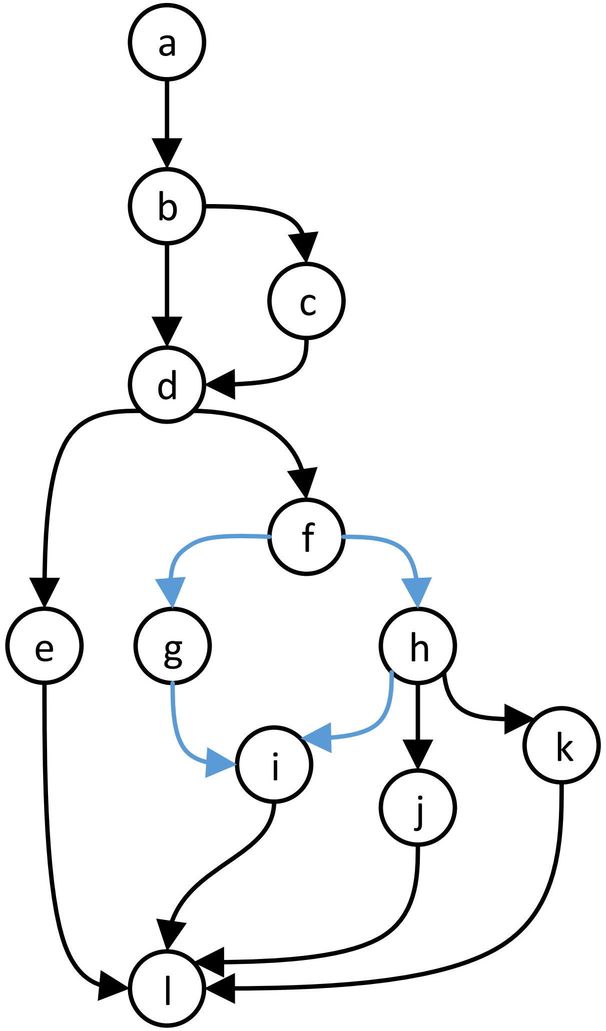
```
…
124.   buffer = kzalloc(buffer_size, GFP_KERNEL);
…
141.   kfree(buffer);
…
```

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**



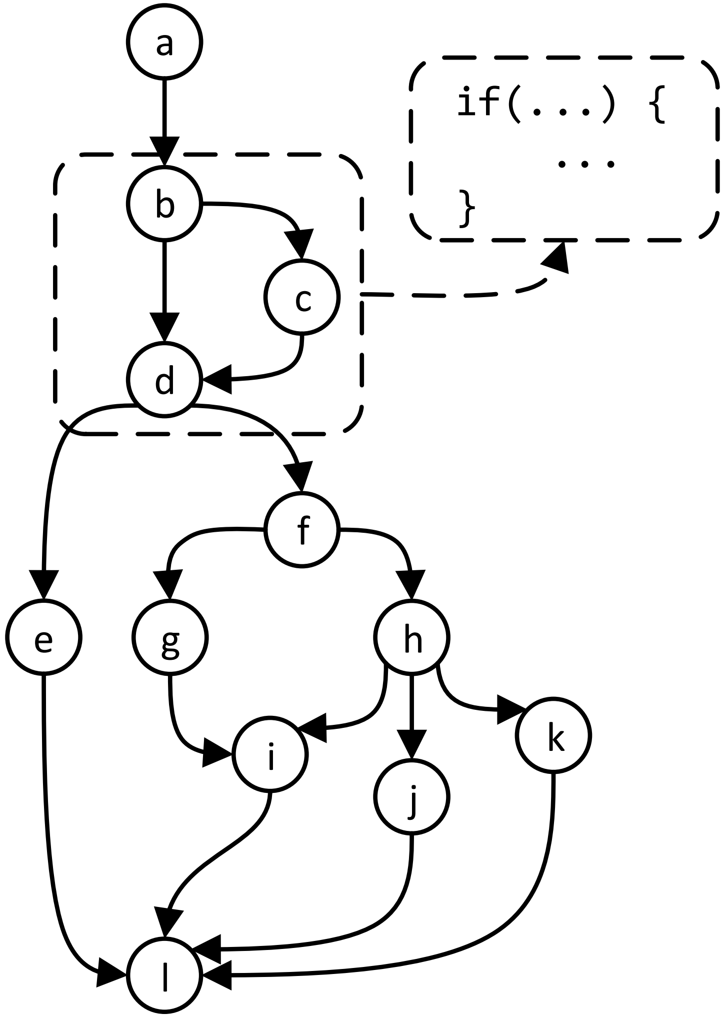
Object-based Similar Path Pair

- **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



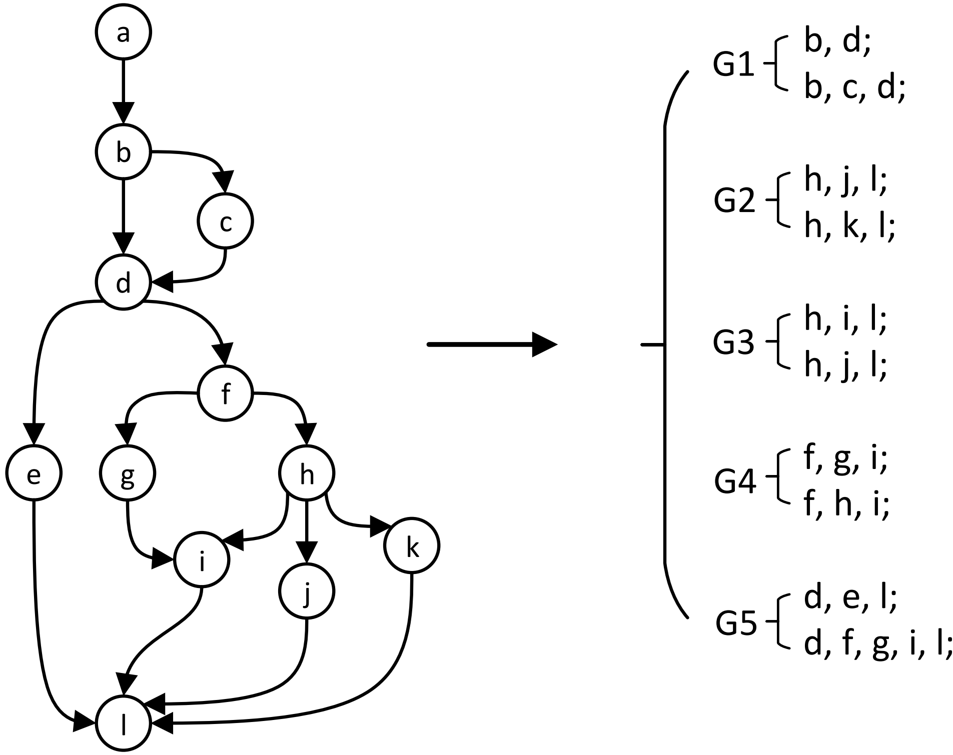
Object-based Similar Path Pair

- 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.



G1 { b, d; b, c, d;

G2 { h, j, l; h, k, l;

G3 { h, i, l; h, j, l;

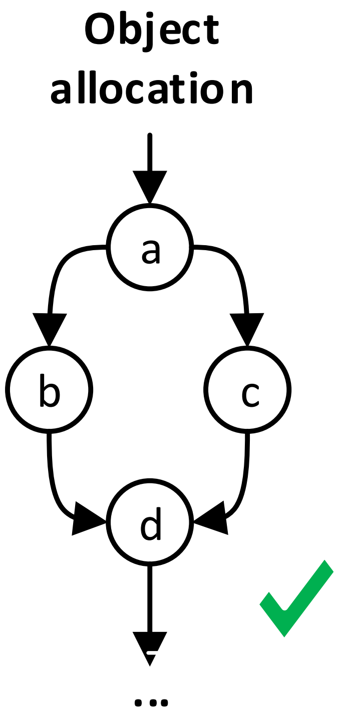
G4 { f, g, i; f, h, i;

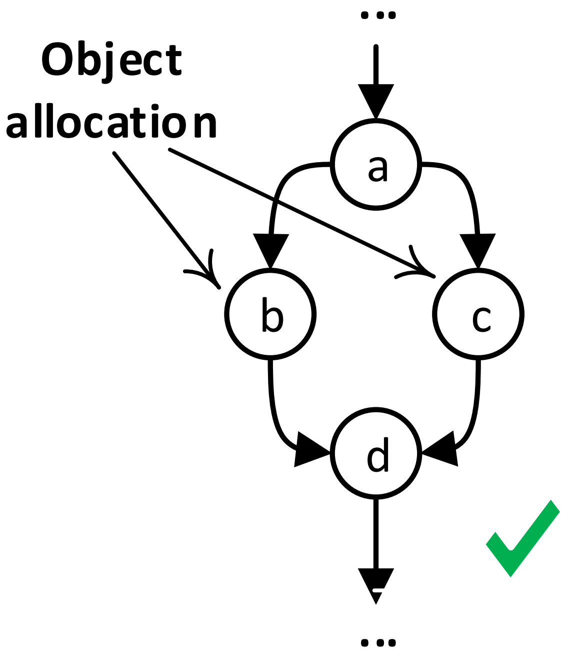
G5 { d, e, l; d, f, g, i, l;

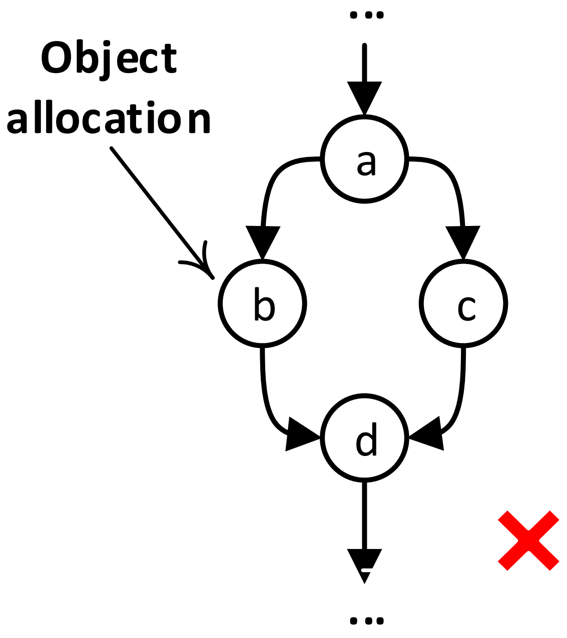
Object-based Similar Path Pair

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

- **Rule 2**
  - The object has the same state in two paths.





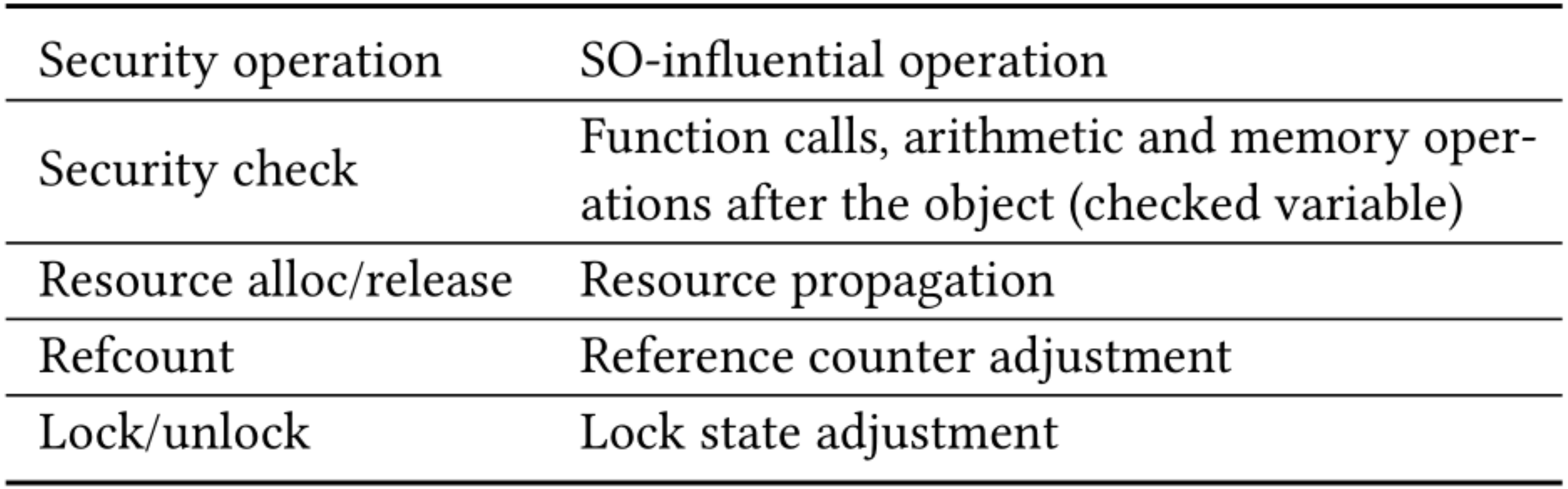


Object-based Similar Path Pair

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

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

Table 1: SO-influential operations.



Security operation	SO-influential operation
Security check	Function calls, arithmetic and memory operations after the object (checked variable)
Resource alloc/release	Resource propagation
Refcount	Reference counter adjustment
Lock/unlock	Lock state adjustment

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. → *Must be object irrelevant*

Post-condition:

The return values of a path pair. → *A pair of normal paths or a pair of error handing paths*

Object-based Similar Path Pair

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

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

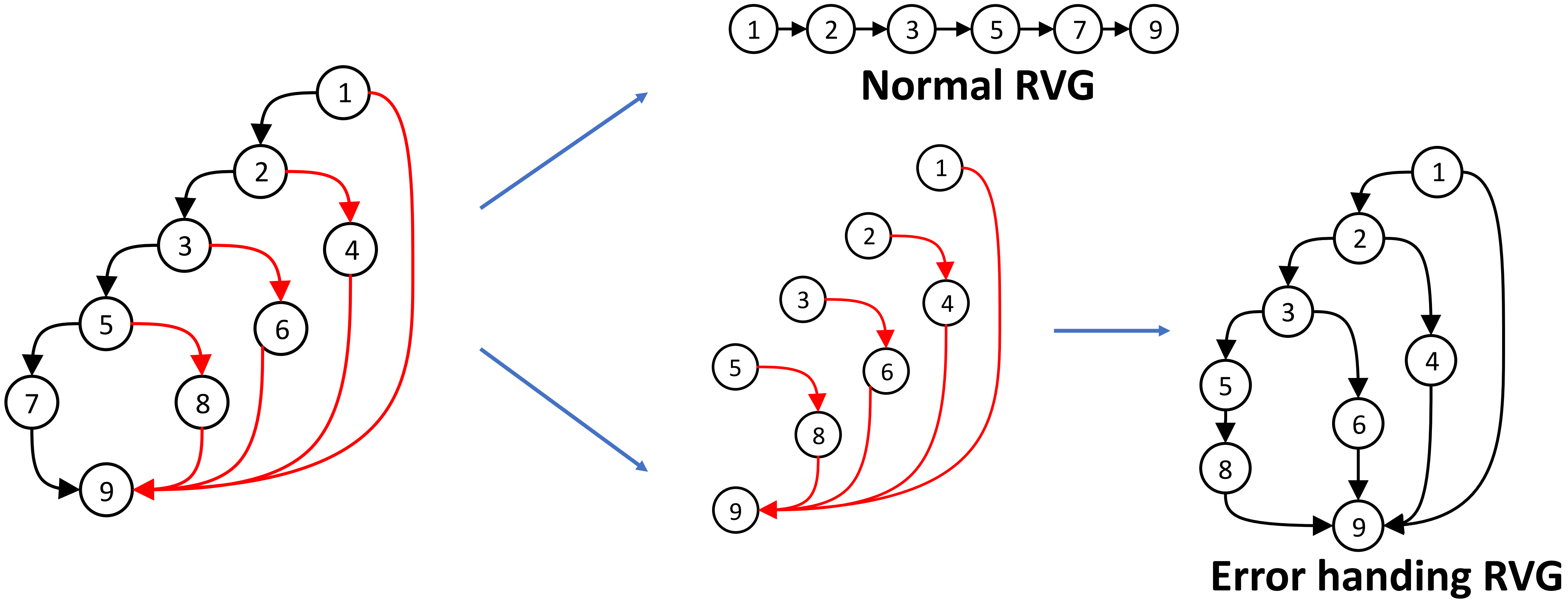
- **Our solution:** **graph partitioning**
  - Divide the CFG into 2 sub-CFGs:
  - Paths in each sub-CFG share the same return value

➢ Return value-based graphs (RVGs)

Object-based Similar Path Pair

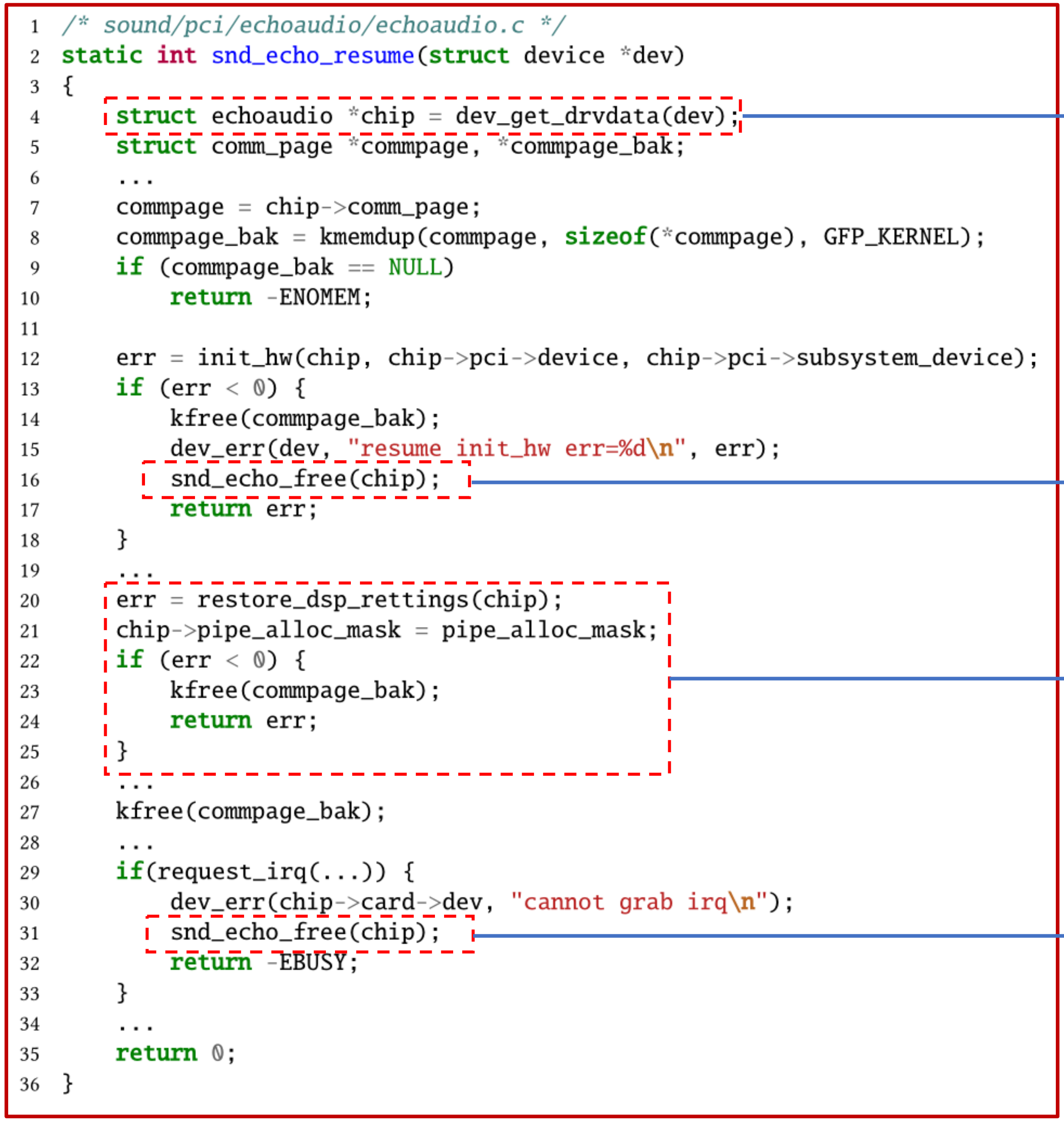
- Rules for constructing object-based similar path pair (OSPP)

  - Generating return value-based graphs



Case Study

A Double-free Bug Found by IPPO



```c
/* 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_dsp_rettings(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*

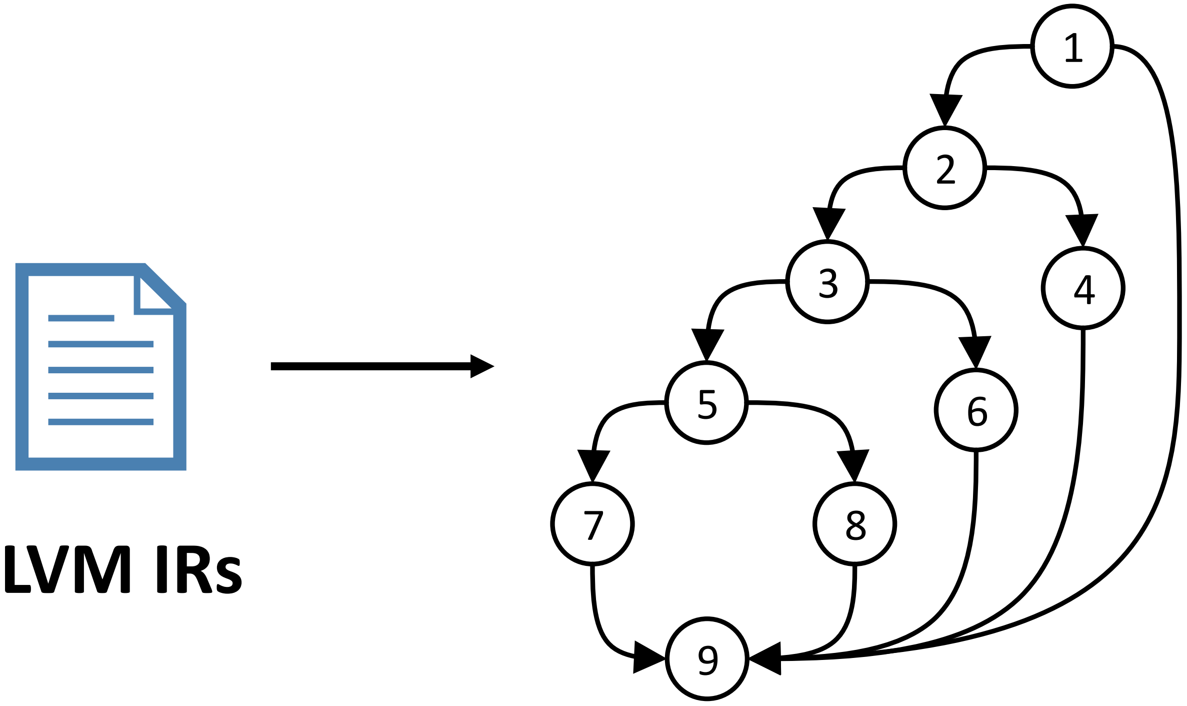
Resource release of object *chip*

Missing release against *chip* in the error handing path

Resource release of object *chip*

Workflow of IPPO

- **Security operation detection & error edges identification**

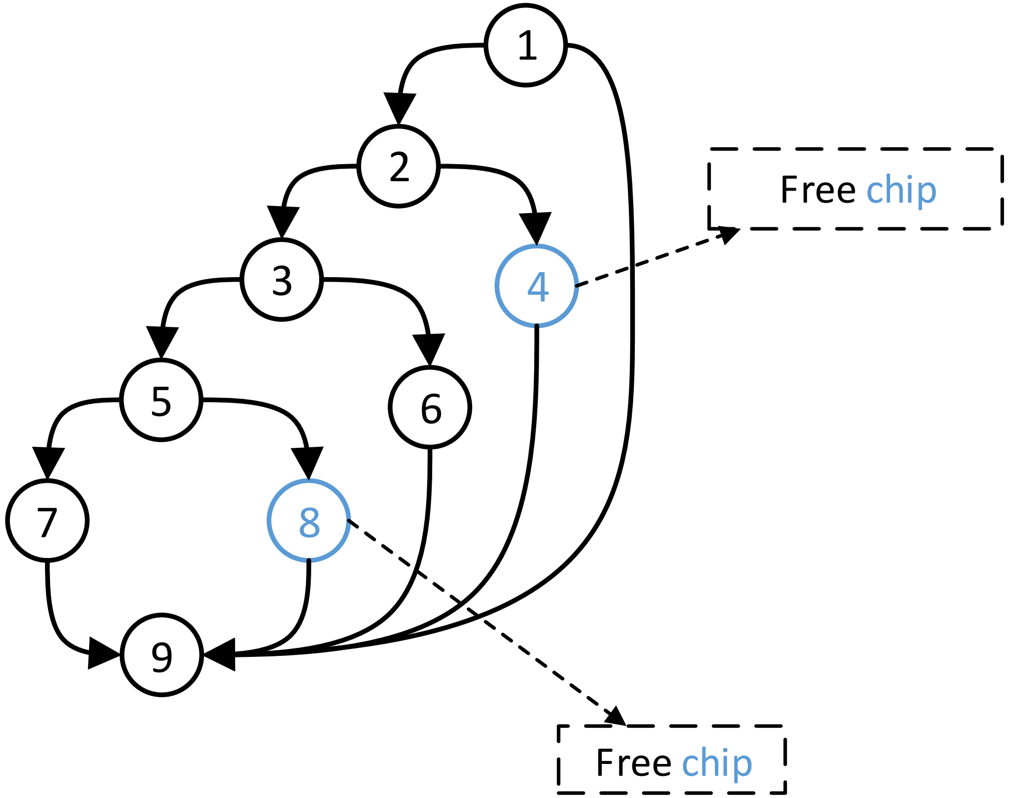


Control flow graph

Workflow of IPPO

- **Security operation detection & error edges identification**

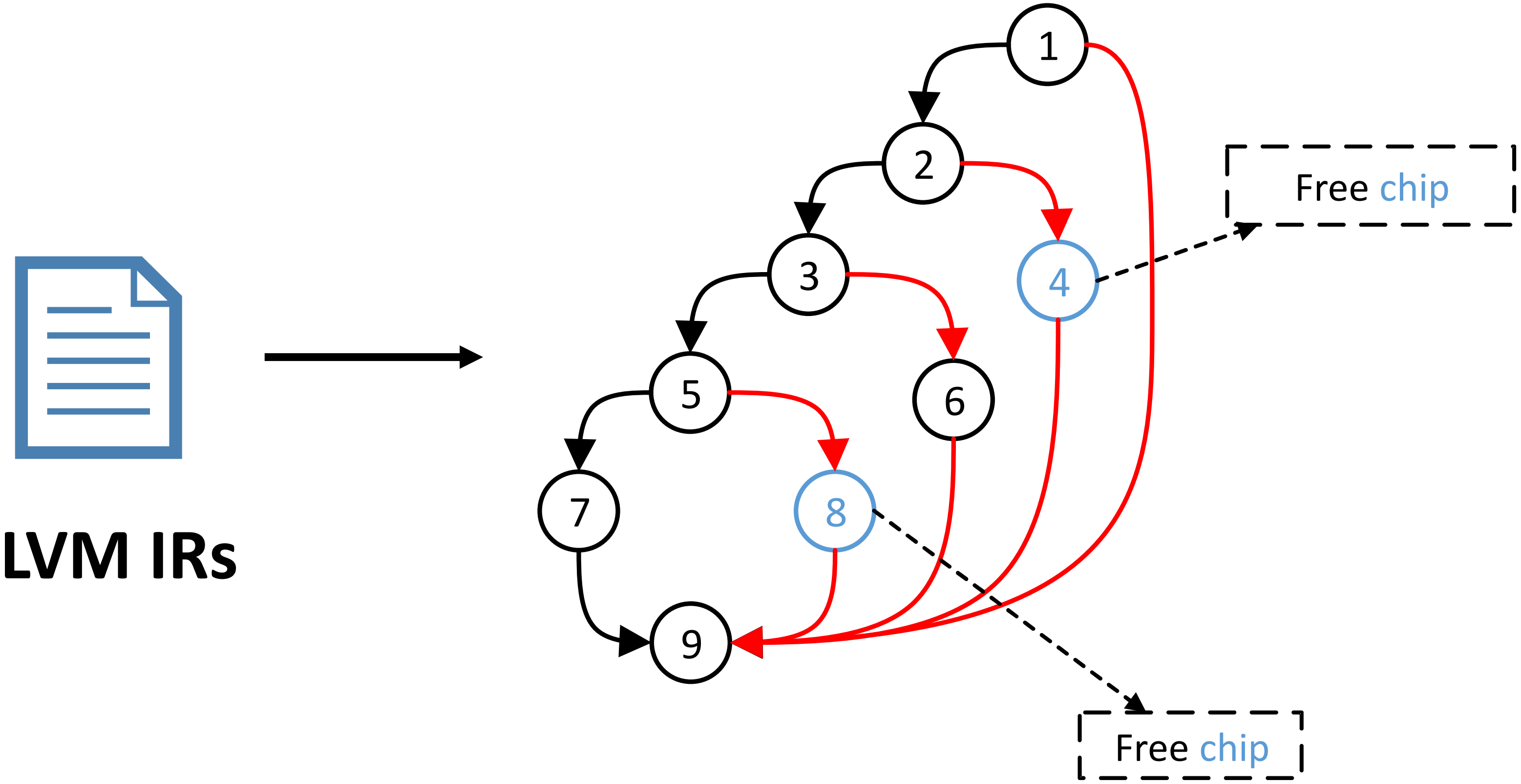
LLVM IRs



Control flow graph

Workflow of IPPO

- **Identify error edges**

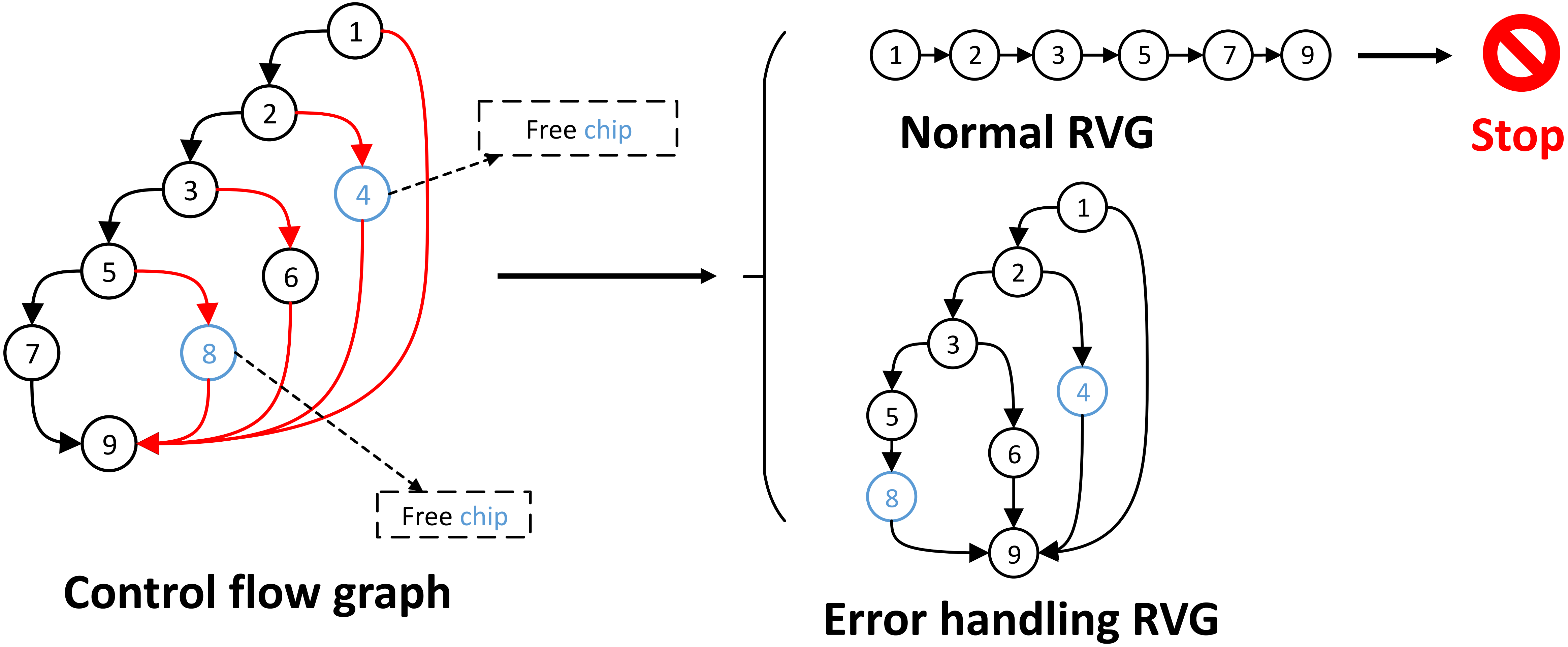


LLVM IRs

Control flow graph

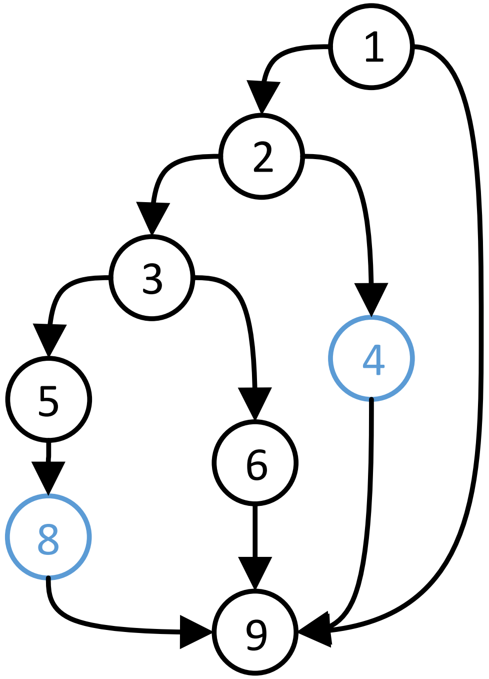
Workflow of IPPO

- **Generate return value-based graphs**



Workflow of IPPO

- **Collect reduced similar paths (RSPs)**



RSP 1
③ - ⑤ - ⑧ - ⑨
③ - ⑥ - ⑨

RSP 2
② - ③ - ⑤ - ⑧ - ⑨
② - ④ - ⑨

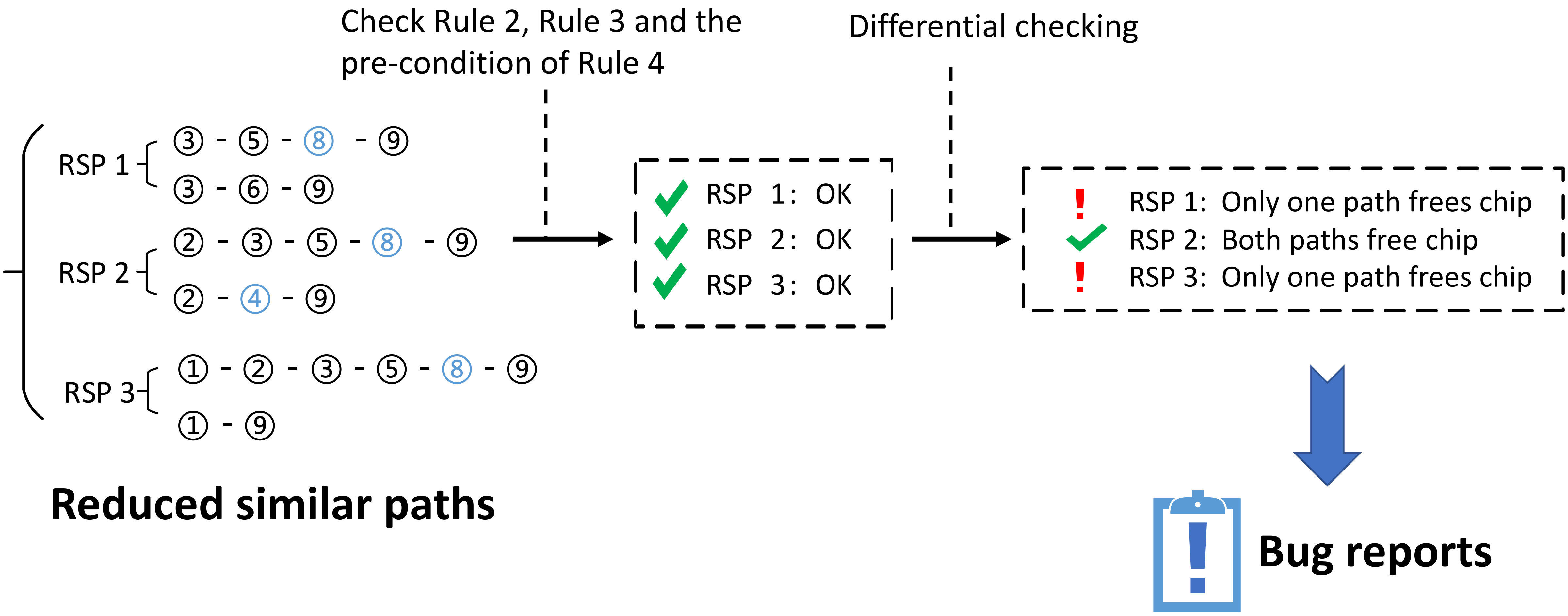
RSP 3
① - ② - ③ - ⑤ - ⑧ - ⑨
① - ⑨

Error handling RVG

Reduced similar paths

Workflow of IPPO

➢ OSPP rules checking & differential checking



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

Differential checking

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

✓ RSP 1: OK
✓ RSP 2: OK
✓ RSP 3: OK

! RSP 1: Only one path frees chip
✓ RSP 2: Both paths free chip
! RSP 3: Only one path frees chip

Reduced similar paths

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.**

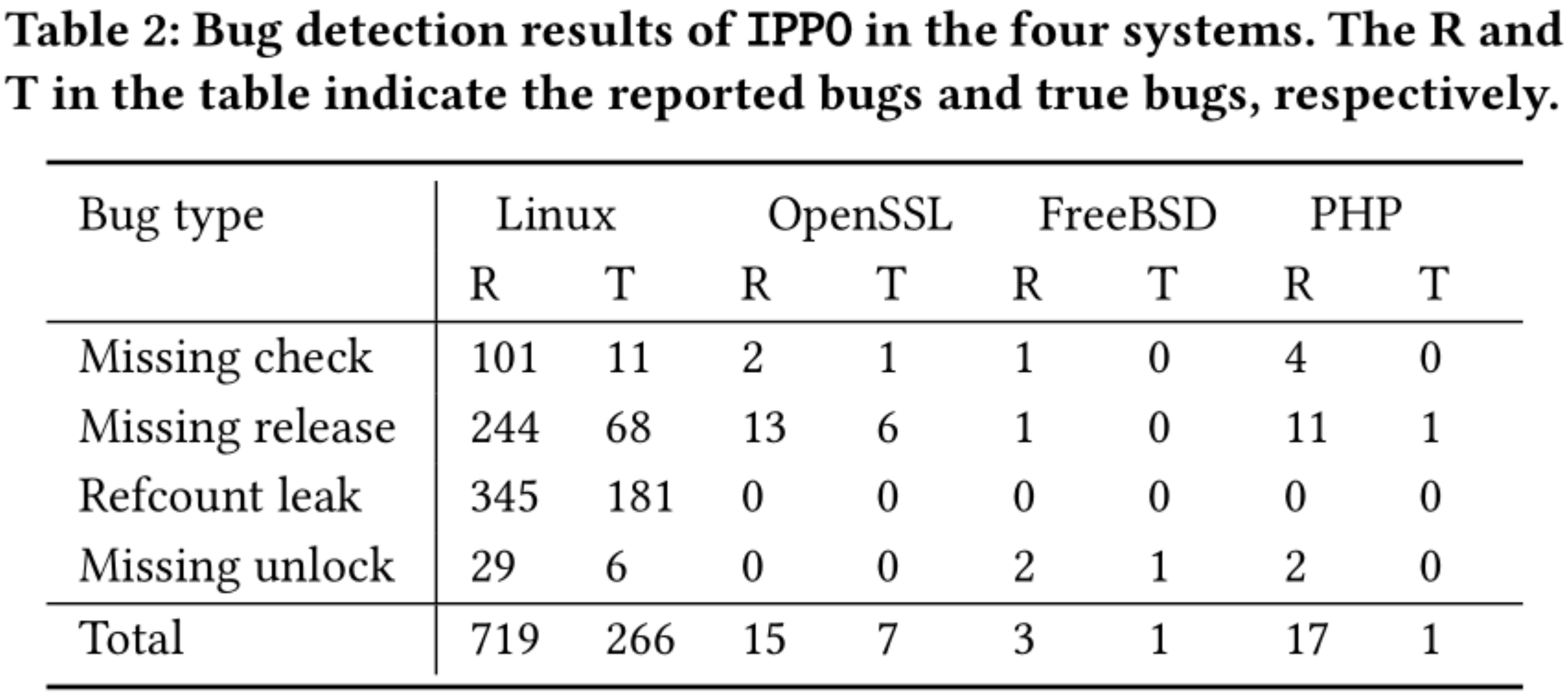


Table 2: Bug detection results of IPPO in the four systems. The R and T in the table indicate the reported bugs and true bugs, respectively.

Bug type	Linux		OpenSSL		FreeBSD		PHP	
	R	T	R	T	R	T	R	T
Missing check	101	11	2	1	1	0	4	0
Missing release	244	68	13	6	1	0	11	1
Refcount leak	345	181	0	0	0	0	0	0
Missing unlock	29	6	0	0	2	1	2	0
Total	719	266	15	7	3	1	17	1

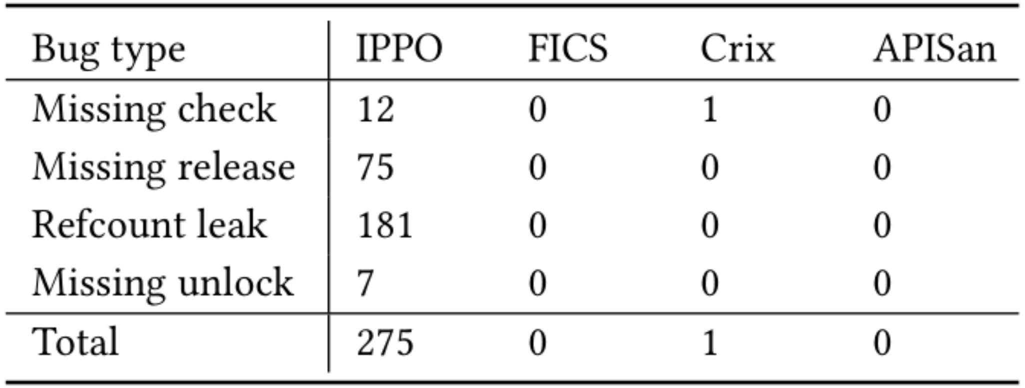
275 valid bugs.

161 are previous unknown.

136 have been fixed by our patches or reports.

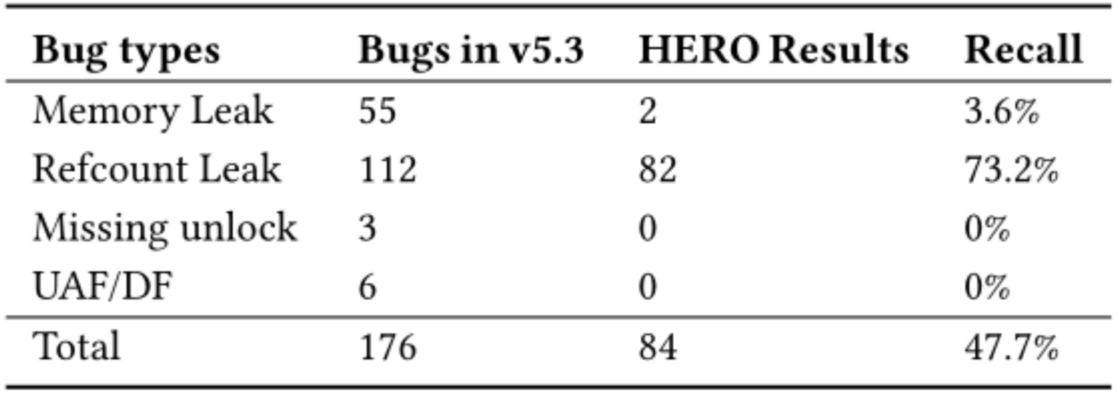
Comparison with Other Tools

- **Comparison with cross-checking tools**



Bug type	IPPO	FICS	Crix	APISan
Missing check	12	0	1	0
Missing release	75	0	0	0
Refcount leak	181	0	0	0
Missing unlock	7	0	0	0
Total	275	0	1	0

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



Bug types	**Bugs in v5.3**	**HERO Results**	**Recall**
Memory Leak	55	2	3.6%
Refcount Leak	112	82	73.2%
Missing unlock	3	0	0%
UAF/DF	6	0	0%
Total	176	84	47.7%

IPPO is a promising complementation with existing tools.

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.

Conclusion

- **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

Q&A

dinghao.liu@zju.edu.cn