中國科學院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

UNIVERSITY OF CALIFORNIA
UCRIVERSIDE

北京郵電大學
Beijing University of Posts and Telecommunications

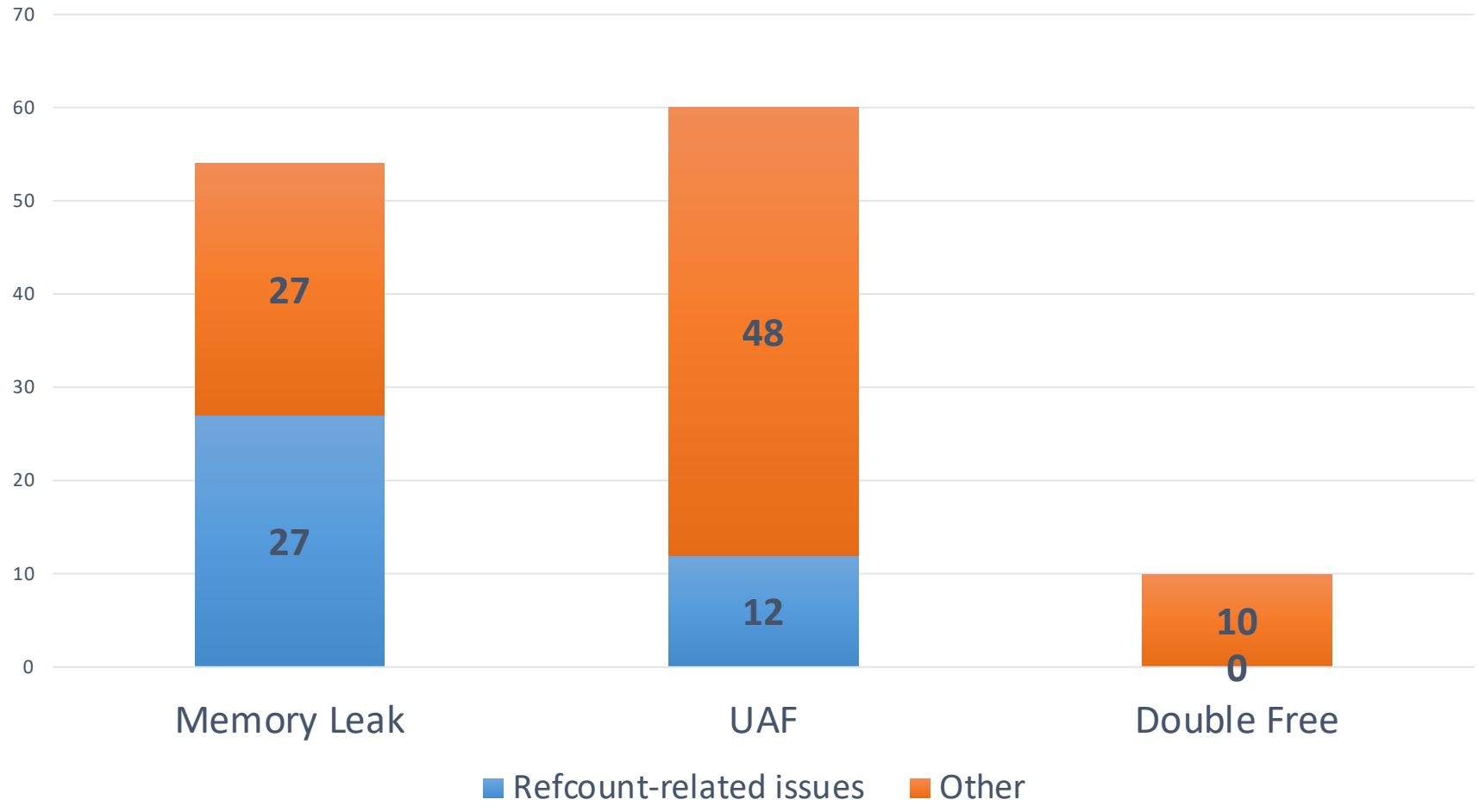
LinKRID: Vetting Imbalance Reference Counting in Linux Kernel with Symbolic Execution

Jian Liu, Lin Yi, Weiteng Chen, Chenyu Song,
Zhiyun Qian, Qiuping Yi

Boston, USA

31ST USENIX SECURITY SYMPOSIUM

Kernel Memory Management Bugs



A statistic on memory management-related issues in Linux 4.14

Reference Counter in Kernels

- Key idea: each object includes a ref_count
  - Assume obj * p = NULL;
  - p = obj1; // obj1->ref_count++
  - p = obj2; // obj1->ref_count--, obj2->ref_count++
- If an object's ref_count == 0, it is garbage
  - No pointers target that object
  - thus, it can be safely freed
- Manually managed → **error prone**

Refcount Bugs

- Two types of bugs
  - Failure to **increase** the counter when a new reference is created -> dangling pointer(s) -> use-after-free (UAF)
  - Failure to **decrement** the counter when a reference has been removed -> memory leaks

```
@@ -2030,7 +2030,6 @@ static int replace_map_fd_with_map_ptr
                        if (IS_ERR(map)) {
                                verbose("fd %d is not pointi
                                        insn->imm);
-                               fdput(f);
                                return PTR_ERR(map);
                        }
```

CVE-2016-4557

```
@@ -794,6 +794,7 @@ long join_session_keyring(const
                ret = PTR_ERR(keyring);
                goto error2;
        } else if (keyring == new->session_keyring)
+               key_put(keyring);
                ret = 0;
                goto error2;
        }
```

CVE-2014-0728

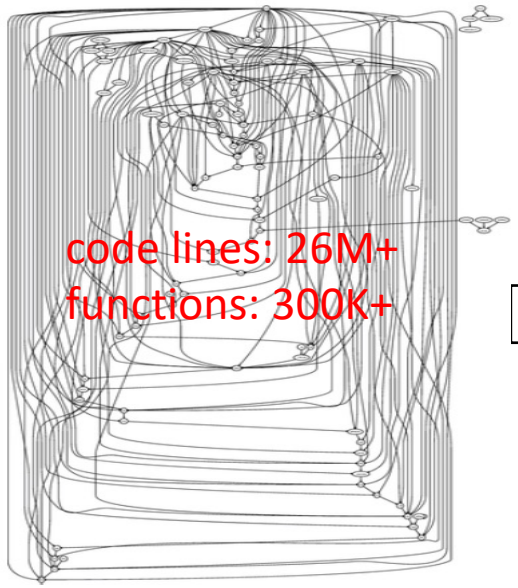
Challenge-1: Refcount Bug Modeling

- Existing stratigies to model refcount bugs
  1. RID[1] and CID[2] : IPP-based (Inconsistent Path Pair) modeling
     - Check the inconsistency between path pair which have same arguments and return value
     - Limited checking scenario, e.g., ***inconsistency within a single path*** (CVE-2016-0728)
  2. Pungi[3]: Affine abstraction modeling
     - Check refcount changes equal to the number of escaped references
     - Not special to Linux Kernel, e.g., Linux refcount wrappers
     - High False Positives Rate in Kernel

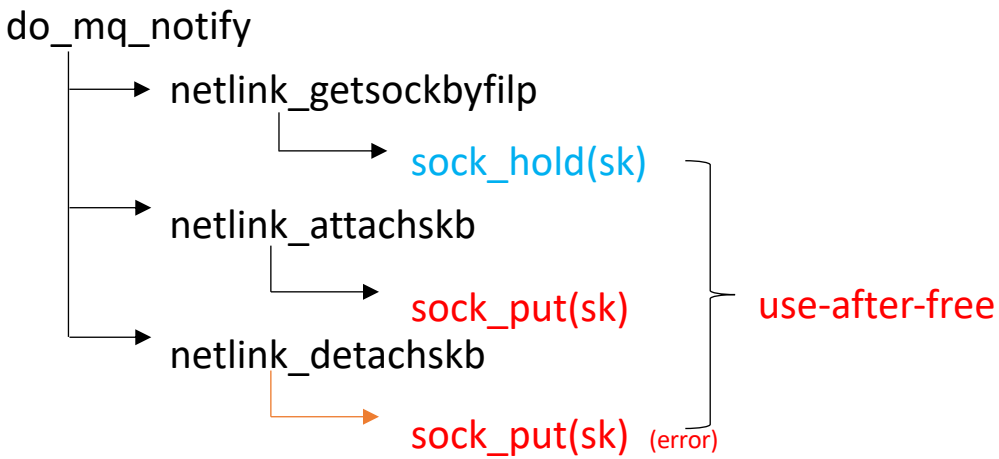
[1] Junjie Mao, Yu Chen, Qixue Xiao, and Yuanchun Shi. RID: Finding Reference Count Bugs with Inconsistent Path Pair Checking. ASPLOS 2016.
[2] Xin Tan, et al. "Detecting Kernel Refcount Bugs with {Two-Dimensional} Consistency Checking. USENIX Security 2021.
[3] Siliang Li and Gang Tan. Finding reference-counting errors in Python/C programs with affine analysis. ECOOP 2014.

Challenge-2: Refcount Bug Checking

- How to scale up the path-sensitive analyse Linux kernel codes, eg. CVE-2017-11176



localizing



1. RID[1]: symbolic execution
    - without details on how to scale up
2. CID[2] and Pungi[3]: static analysis

Challenge-3: Kernel Refcount Conventions

- Existing works without considering kernel refcount conventions[1] (e.g., CVE-2016-4805)

1. External reference:
   - strictly abide the reference consistency invariant

2. **Internal reference** (or weak reference):
   - It simply means the object "exists"
   - Violates the refcount consistency invariant
   - Used inside a software cache, such as a radix tree, a double linked list...

```
1   static void amp_mgr_destroy(struct kref *kref) {
2     struct amp_mgr *mgr = container_of(kref, struct
       amp_mgr, kref);
3     mutex_lock(&amp_mgr_list_lock);
4     list_del(&mgr->list);
5     mutex_unlock(&amp_mgr_list_lock);
6     kfree(mgr);
7   }
8   int amp_mgr_put(struct amp_mgr *mgr) {
9     return kref_put(&mgr->kref,
       &amp_mgr_destroy);
10  }
11
```

An example of internal reference

[1]Neil Brown. Linux kernel design patterns - part 1. https://lwn.net/Articles/336224/, 2009.

Our Key Observations

- **Bug Oracle**: the refcount and corresponding reference changes should be consistent
- **Invariant**: Δ#(reference) == #escaped - #released

```
1  func_1(){ // local reference scope begins
2    p = kmalloc(sizeof( *p ));
3    refcnt_init(p); // init refcnt = 1
4    list_insert(p, list); // save to heap, escaped
       reference += 1
5  } // local reference scope ends
6  // #escape = 1, #release = 0,  Δrefcnt = 1.
7  // Δrefcnt == (#escape - #release) == 1. correct!
```

(a) Creation

```
9  func_2(){ // local reference scope begins
10   p = list_lookup("name", list);
11   refcnt_get(p); // refcnt += 1
12   use(p);
13   refcnt_put(p); // refcnt -= 1
14 } // local reference scope ends
15 // #escape = 0, #release = 0, Δrefcnt = 0.
16 // Δrefcnt == (#escape - #release) == 0. correct!
```

(b) Usage

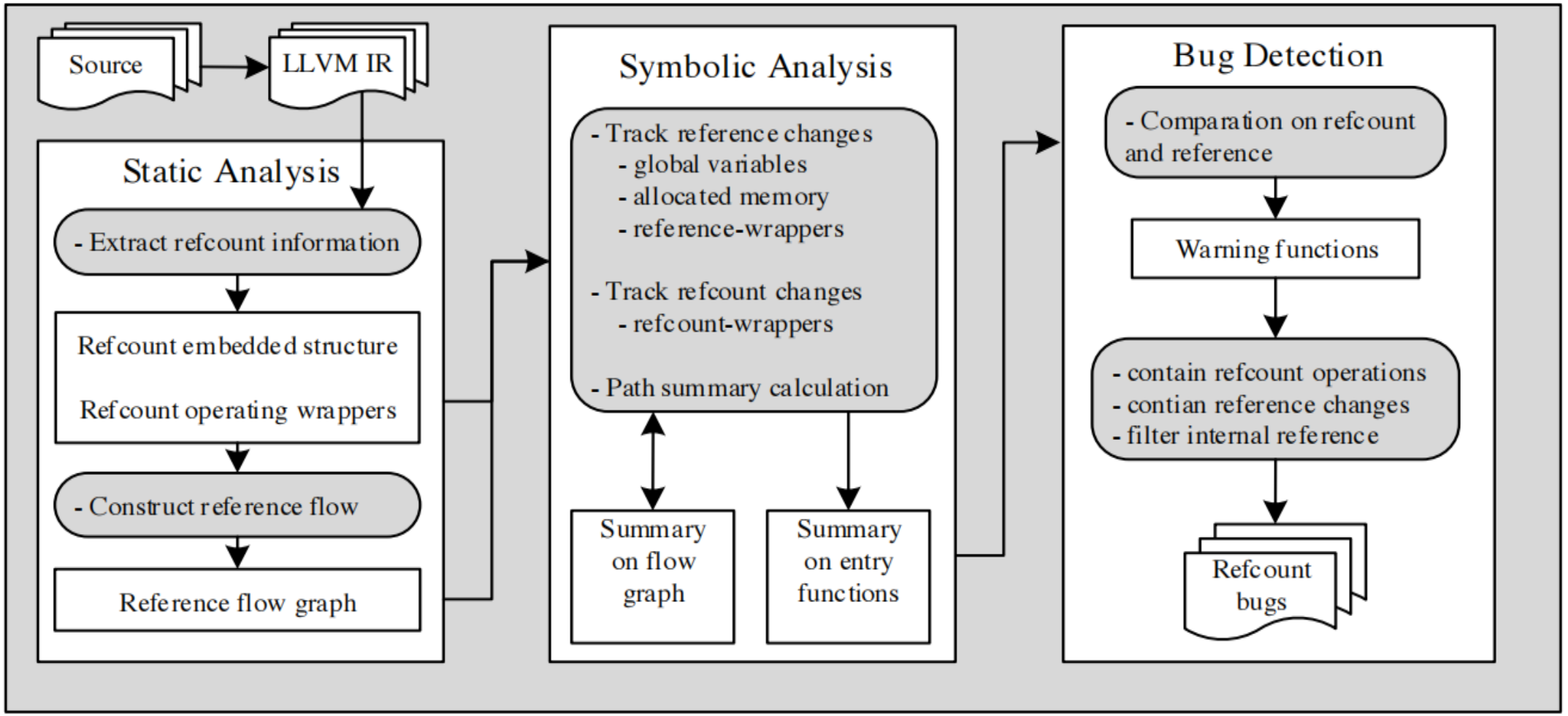
```
18  func_3(){ // local reference scope begins
19    p = list_lookup("name", list);
20    refcnt_get(p); // refcnt += 1
21    insert(p, list2);  // escaped reference += 1
22  } // local reference scope ends
23  // #escape = 1, #release = 0,  Δrefcnt = 1.
24  // Δrefcnt == (#escape - #release) == 1. correct!
```

(c) Escape

```
26 func_4(){ // local reference scope begins
27   p = list_lookup("name", list);
28   list_remove(p, list); // released reference += 1
29   refcnt_put(p); // refcnt -= 1
30 } // local reference scope ends
31 // #escape = 0, #release = 1, Δrefcnt = -1.
32 // Δrefcnt == (#escape - #release) == -1. correct!
```

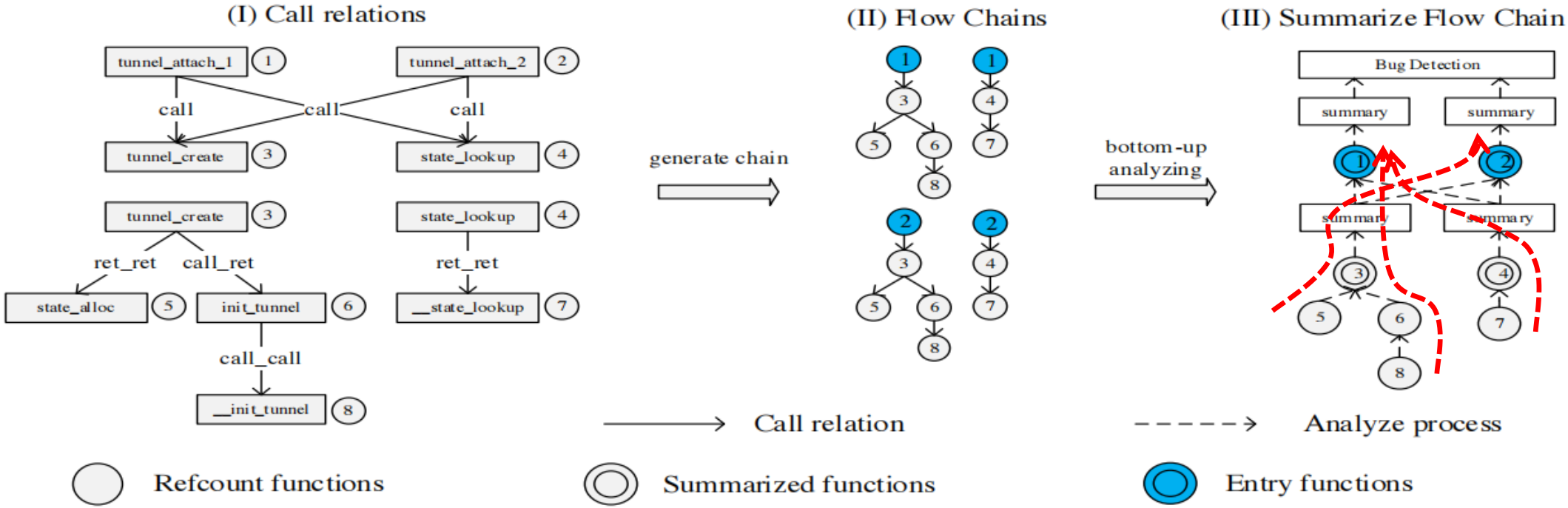
(d) Release

LinkRID Workflow



Summary-based Chain Analysis

- Ideas: using under-constrained symbolic execution to analyze flow chains symbolically to identify potential buggy paths
  - scalable: bottom-up summary-based computing
  - precious: on-the-fly tracking reference and refcount changes



Bug Detection (1)

- Given a flow chain, LinkRID reports a potential bug whenever there exists a local reference $|r$ satisfying the following condition

$$\Delta refcount_{/r} \neq |escape_{/r}| - |release_{/r}|$$

- Detailed bug reporting
  - branch direction, refcount get/put, source code line...

```
join_session_keyring path record:
Error Path:
security/keys/process_keys.c L780, get keyring refcnt +
  1 by find_keyring_by_name
security/keys/process_keys.c L781, if condition is false
security/keys/process_keys.c L791, if condition is false
security/keys/process_keys.c L794, if condition is true
...
Summary: <keyring, escape=∅, release=∅, Δrefcnt=1, return=0>
Result : refcnt error in keyring
```

Bug Detection (2)

- Identifying Internal References: two internal reference patterns
  - (1) separate/additional references (released automatically when the refcount reaches zero)
  - (2) released in domain-specific manners, eg.

Considering as Internal References

- Association of Refcount and Reference Changes: heuristic-based aliasing inferring
  - (1) if two local references (e.g., gdm and tty_port) happen to trigger two warnings in the same function
  - (2) one local reference is “embedded” in another

Considering as related refcount and reference changes

Implementation: LinkRID

- Static anlaysis (multiple LLVM Passes with version 9.0.0)
  - Call graph construction from KENALI
  - Data flow analysis
- Under-constraint symoblic execution engine (based on KLEE 2.0)
  - Lazy initializtion of Kernel Objects
  - Summary-based symbolic execution
- Bug detector
  - Convention filters
- Totoal: 3.5k lines of C++ code and 2.7k line of python code

Evaluation Setup

- Target: Linux 4.14.0 compiled with LLVM
  - git commit *bebc608*
  - Compiled with allyesconfig
  - LLVM 3.9 with `make defconfig` and `make allyesconfig`

- Experimental configuration
  - (Virtual machines each with Intel Xeon E3-1220 CPU and 32GB RAM) X 2

Performance

- Time to analyze the whole kernel (with allyesconfig)
  - Static analysis: 1 hours
    - Identify 445 refcounted structures, 685 refcounted functions and 54731 related functions
    - generate 12075 flow chains
  - Symoblic execution: 192 hours
    - gnerate 9419 summaries

Phase	Description	Data	
		defconfig	allyesconfig
Static Analysis	refcounted structures	85	445
	refcount wrapper	149	685
	related functions	16155	54731
	flow chains	4063	12075
Symbolic Execution	summaries	2964	9419
Bug Detection	BUGs	31	118

Effectiveness

- Found 118 new refount bugs in Linux Kernel 4.14.0
  - **82** are new bugs, including **30** 1-day bugs and **52** 0-day bugs.
  - **20** patches have already been merged into the Linux upstream.

Conclusion

- LinkRID, a scalable and practical refcount bug discovery tool
  - scale up the path-sensitive analysis, with a lightweight static analysis and an under-constrained symbolic execution
  - a set of conventions seemingly erroneous but benign usage patterns

Thank you for listening!

Q&A