Automatically Diagnosing Use-after-free Bugs via Reference Miscounting Detection on Binaries

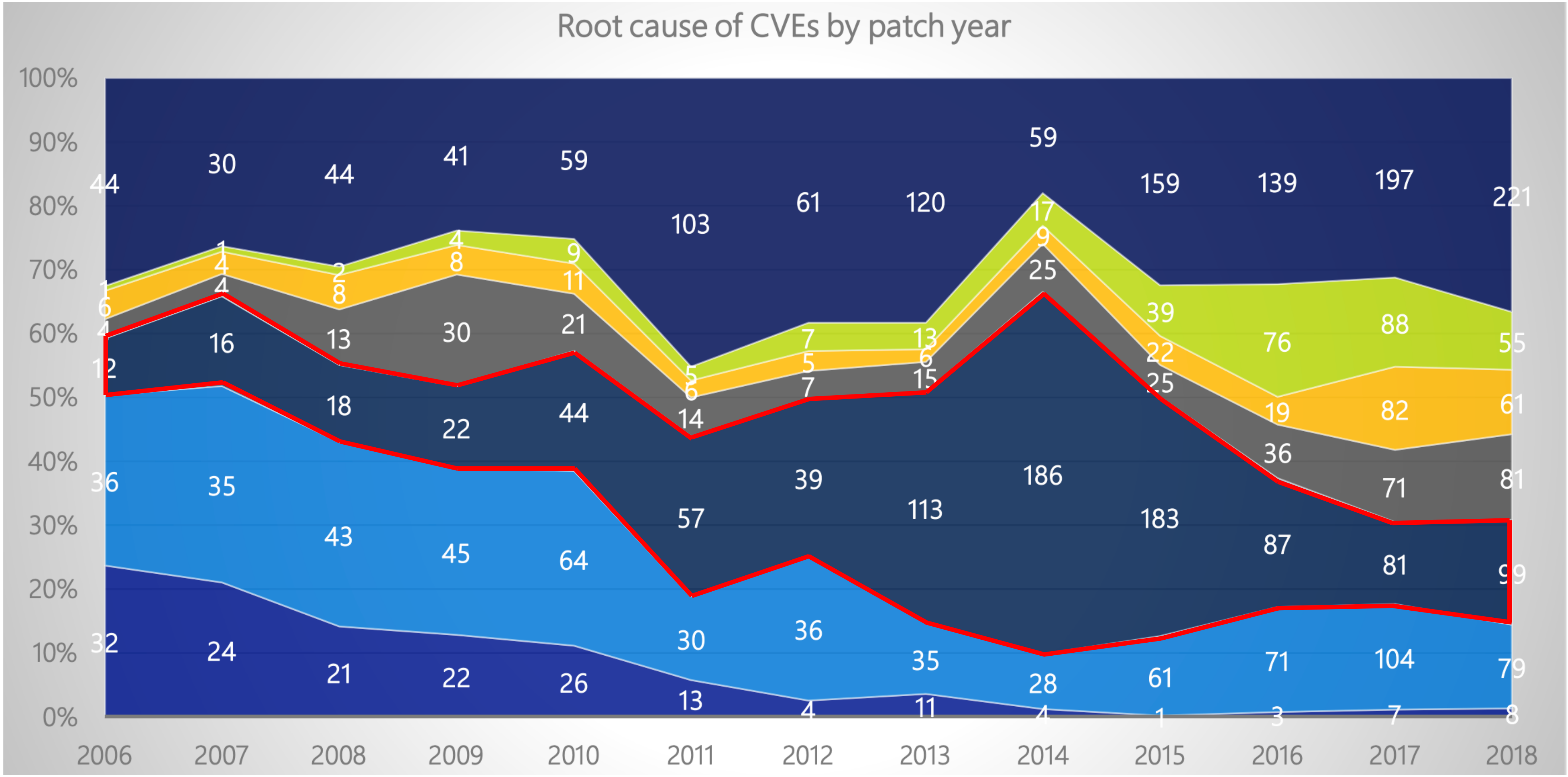
Liang He Hong Hu Purui Su Yan Cai Zhenkai Liang

ISCAS
中國科學院軟件研究所
Institute of Software Chinese Academy of Sciences

PennState

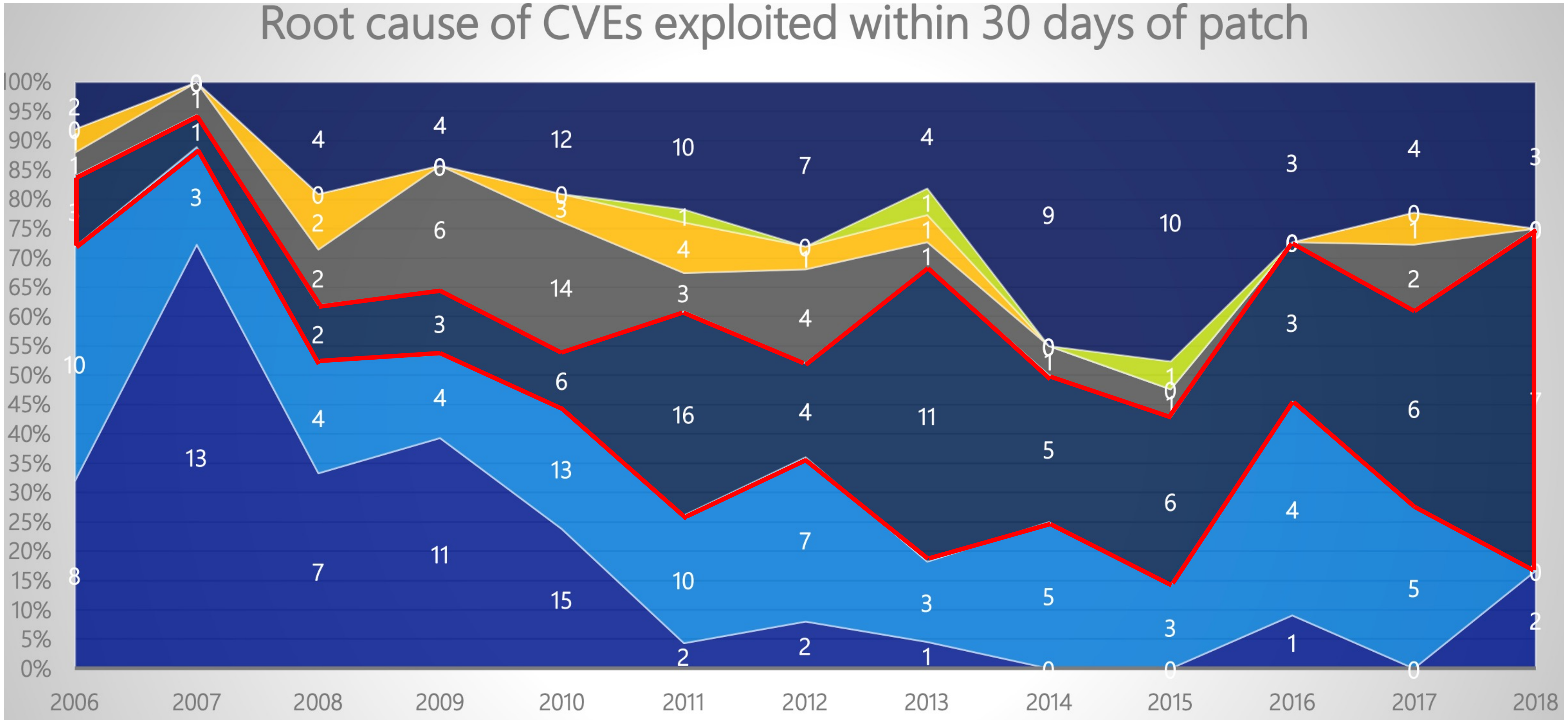
NUS
National University of Singapore

Use-after-free (UAF): **consistently popular**



Matt Miller. Trends, Challenges, And Strategic Shifts In The Software Vulnerability Mitigation Landscape. BluHat IL. 2019

Use-after-free (UAF): **highly exploitable**



Matt Miller. Trends, Challenges, And Strategic Shifts In The Software Vulnerability Mitigation Landscape. BluHat IL. 2019

Root causes of UAF bugs

- UAF: use a dangling pointer after the referred object is freed
- (Illegally use) a dangling pointer after the referred object is freed
  - dangling use
  - *DangNull [NDSS'15], pSweeper [CCS'18], ASAN [ATC'12], …*
- Use a dangling pointer after the referred object is (illegally freed)
  - premature free
  - no existing work

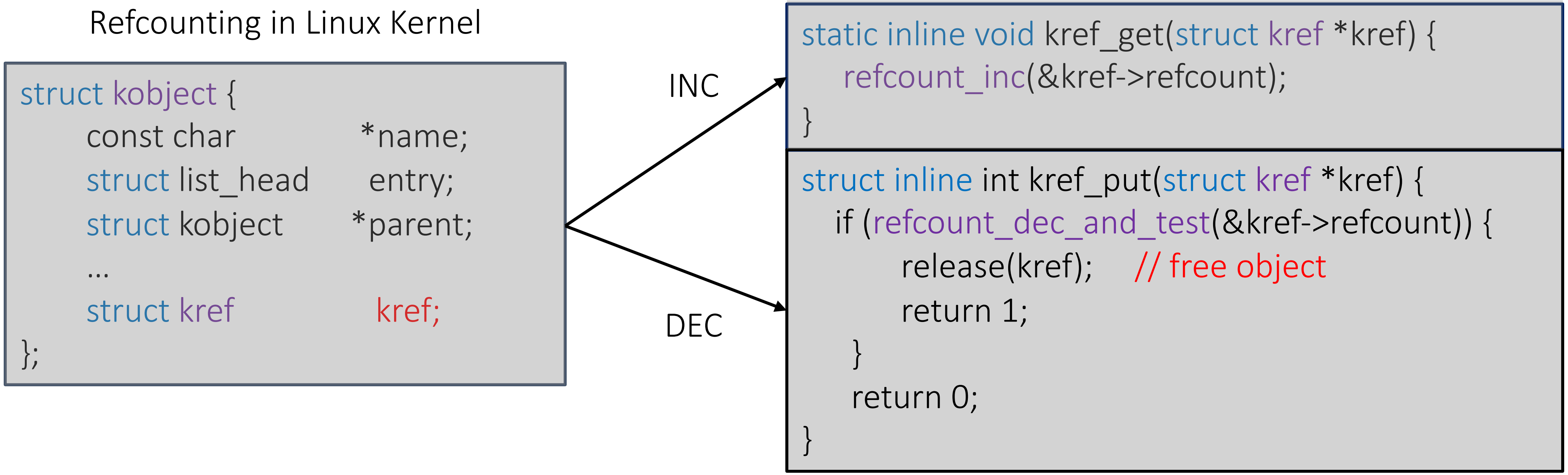
Our work: FreeWill

- Automatic diagnose premature-free caused UAF bugs
    - identify *reference miscounting* as a common reason
    - detect reference miscounting operations
    - suggest possible patches
- Evaluations on large programs/systems
    - 76 bugs from Linux/MacOS, Python/PHP, Chrome/Firefox/IE
        - confirm 48 miscouting, 16 dangling usage
    - complete analysis within 15 minutes
    - 56 patch suggestions

Reference counting for memory management

- Associate a counter for each heap object
- Create a new reference => increase the counter
- Destroy an existing reference => decrease the counter
  - if counter reaches 0, free the object

Refcounting in Linux Kernel



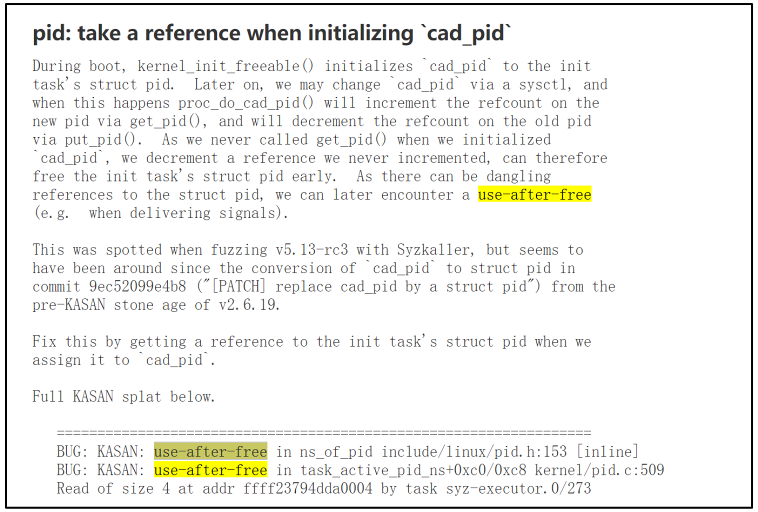
```
struct kobject {
        const char            *name;
        struct list_head       entry;
        struct kobject        *parent;
        …
        struct kref            kref;
};
```

```
static inline void kref_get(struct kref *kref) {
    refcount_inc(&kref->refcount);
}
```

```
struct inline int kref_put(struct kref *kref) {
   if (refcount_dec_and_test(&kref->refcount)) {
         release(kref);     // free object
         return 1;
    }
    return 0;
}
```

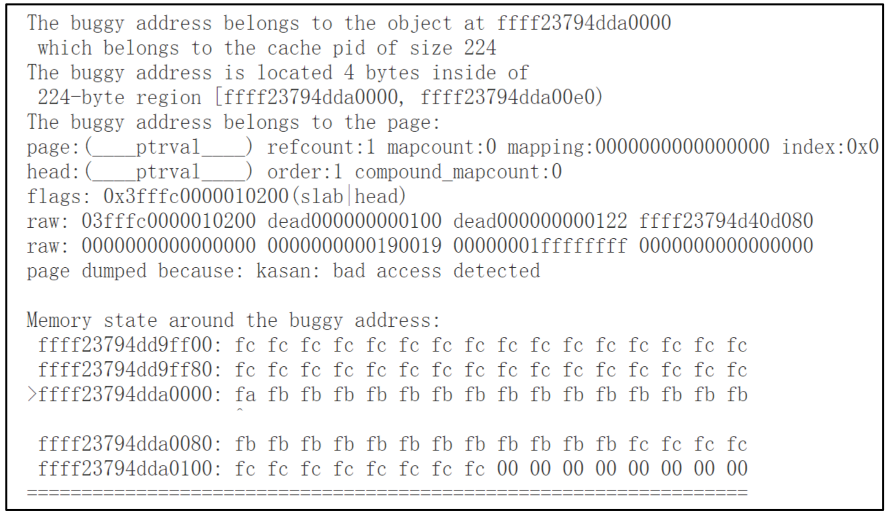
Reference miscounting

- Miss decrements for destroyed references
  - never free (memory leak)
- Miss increments for newly created references
  - premature free (finally use-after-free)

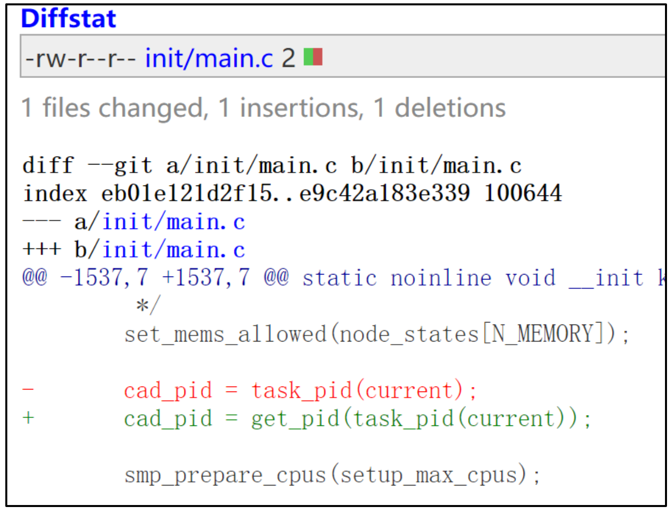


pid: take a reference when initializing `cad_pid`

```
During boot, kernel_init_freeable() initializes `cad_pid` to the init
task's struct pid.  Later on, we may change `cad_pid` via a sysctl, and
when this happens proc_do_cad_pid() will increment the refcount on the
new pid via get_pid(), and will decrement the refcount on the old pid
via put_pid().  As we never called get_pid() when we initialized
`cad_pid`, we decrement a reference we never incremented, can therefore
free the init task's struct pid early.  As there can be dangling
references to the struct pid, we can later encounter a use-after-free
(e.g.  when delivering signals).

This was spotted when fuzzing v5.13-rc3 with Syzkaller, but seems to
have been around since the conversion of `cad_pid` to struct pid in
commit 9ec52099e4b8 ("[PATCH] replace cad_pid by a struct pid") from the
pre-KASAN stone age of v2.6.19.

Fix this by getting a reference to the init task's struct pid when we
assign it to `cad_pid`.

Full KASAN splat below.

  ==================================================================
  BUG: KASAN: use-after-free in ns_of_pid include/linux/pid.h:153 [inline]
  BUG: KASAN: use-after-free in task_active_pid_ns+0xc0/0xc8 kernel/pid.c:509
  Read of size 4 at addr ffff23794dda0004 by task syz-executor.0/273
```



```
The buggy address belongs to the object at ffff23794dda0000
 which belongs to the cache pid of size 224
The buggy address is located 4 bytes inside of
 224-byte region [ffff23794dda0000, ffff23794dda00e0)
The buggy address belongs to the page:
page:(____ptrval____) refcount:1 mapcount:0 mapping:0000000000000000 index:0x0
head:(____ptrval____) order:1 compound_mapcount:0
flags: 0x3fffc0000010200(slab|head)
raw: 03fffc0000010200 dead000000000100 dead000000000122 ffff23794d40d080
raw: 0000000000000000 0000000000190019 00000001ffffffff 0000000000000000
page dumped because: kasan: bad access detected

Memory state around the buggy address:
 ffff23794dd9ff00: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff23794dd9ff80: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
>ffff23794dda0000: fa fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
                   ^
 ffff23794dda0080: fb fb fb fb fb fb fb fb fb fb fb fb fc fc fc fc
 ffff23794dda0100: fc fc fc fc fc fc fc fc 00 00 00 00 00 00 00 00
==================================================================
```



Diffstat

-rw-r--r-- init/main.c 2 🟩🟥

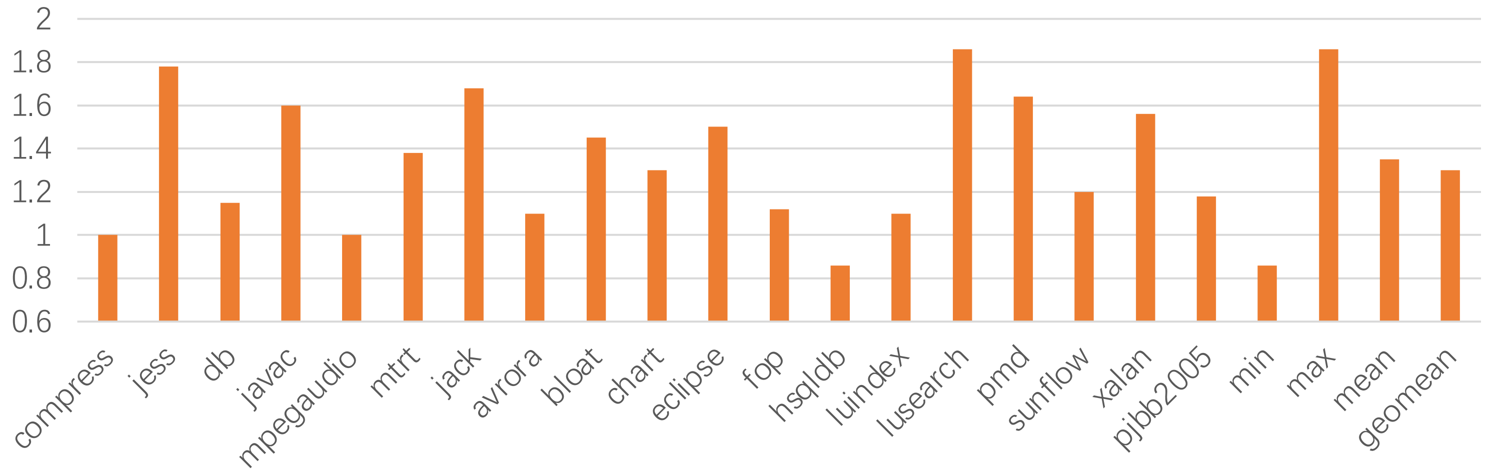
1 files changed, 1 insertions, 1 deletions

```
diff --git a/init/main.c b/init/main.c
index eb01e121d2f15..e9c42a183e339 100644
--- a/init/main.c
+++ b/init/main.c
@@ -1537,7 +1537,7 @@ static noinline void __init k
 	 */
 	set_mems_allowed(node_states[N_MEMORY]);

-	cad_pid = task_pid(current);
+	cad_pid = get_pid(task_pid(current));

 	smp_prepare_cpus(setup_max_cpus);
```

Linux Kernel Commit-0711F0D7050
(2021-06-04)

Challenge of detection: not all missed refcounting are bad

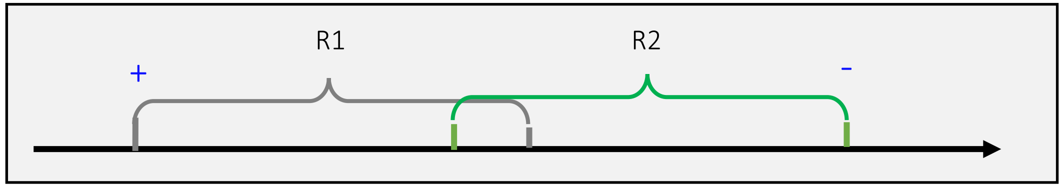
- Refcounting brings ***performance*** overhead (up to 30%)



Rifat Shahriyar, Stephen M. Blackburn, and Daniel Frampton. Down For The Count? Getting Reference Counting Back In The Ring. In ACM SIGPLAN Notices, volume 47, pages 73–84. ACM, 2012.

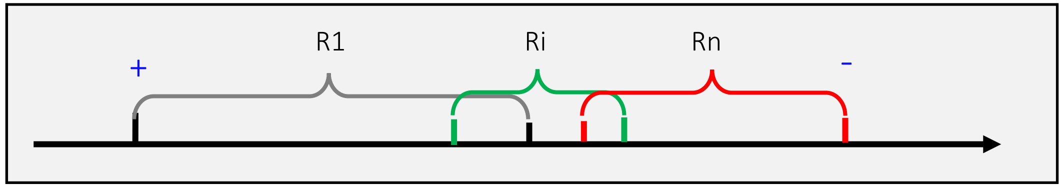
- Programmers ***intentionally omit*** refcounting operations
  - complicated rules guiding the omission
- Which missed refcounting is at fault?

Legal refcounting-omission rules



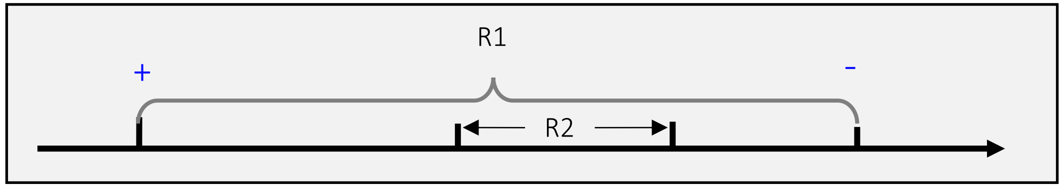
overlapping rule

- Safe to omit R1--, R2++



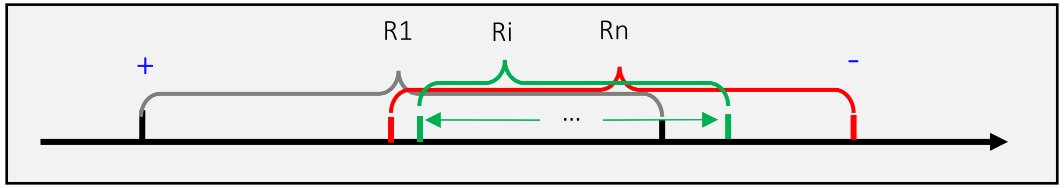
transmitting-overlapping rule

- Safe to omit R1--, Ri++, Ri--, Rn++



containing rule

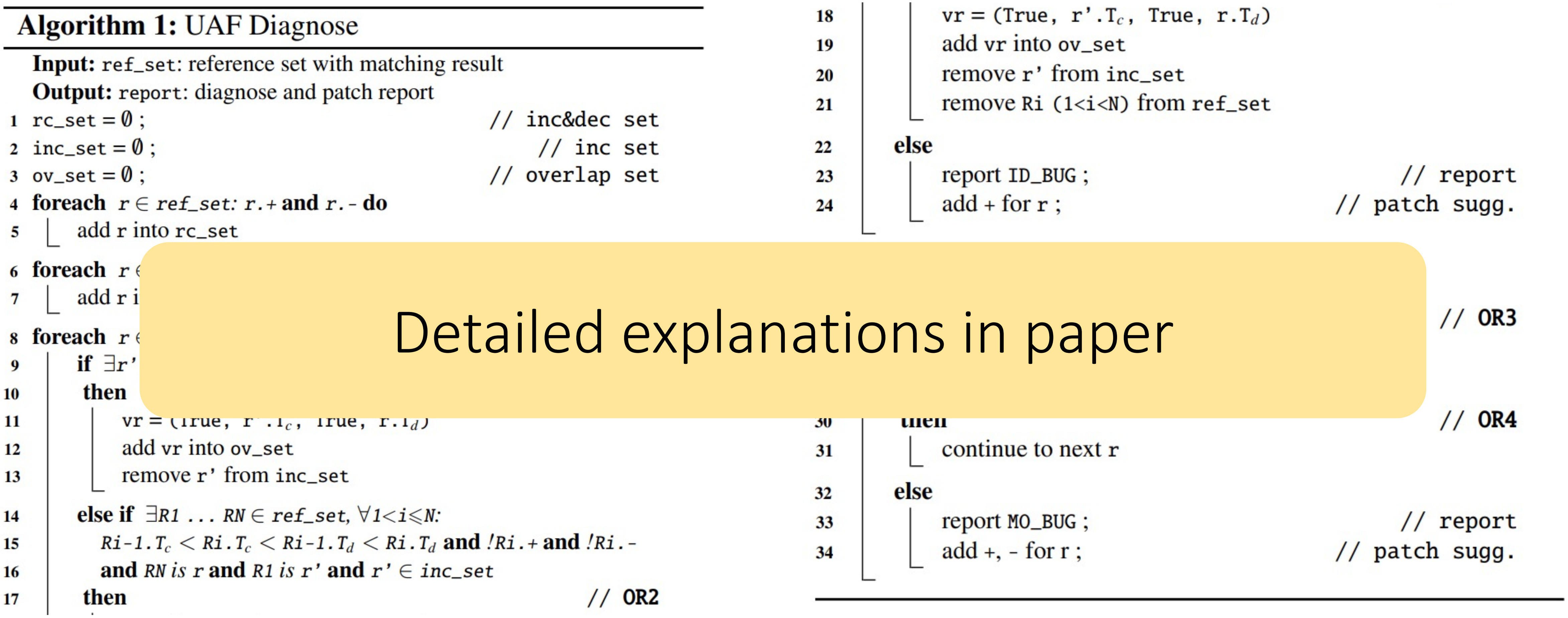
- Safe to omit R2++ & R2--



overlapping-containing rule

- Safe to omit R1--, Ri++ & Ri--, Rn++

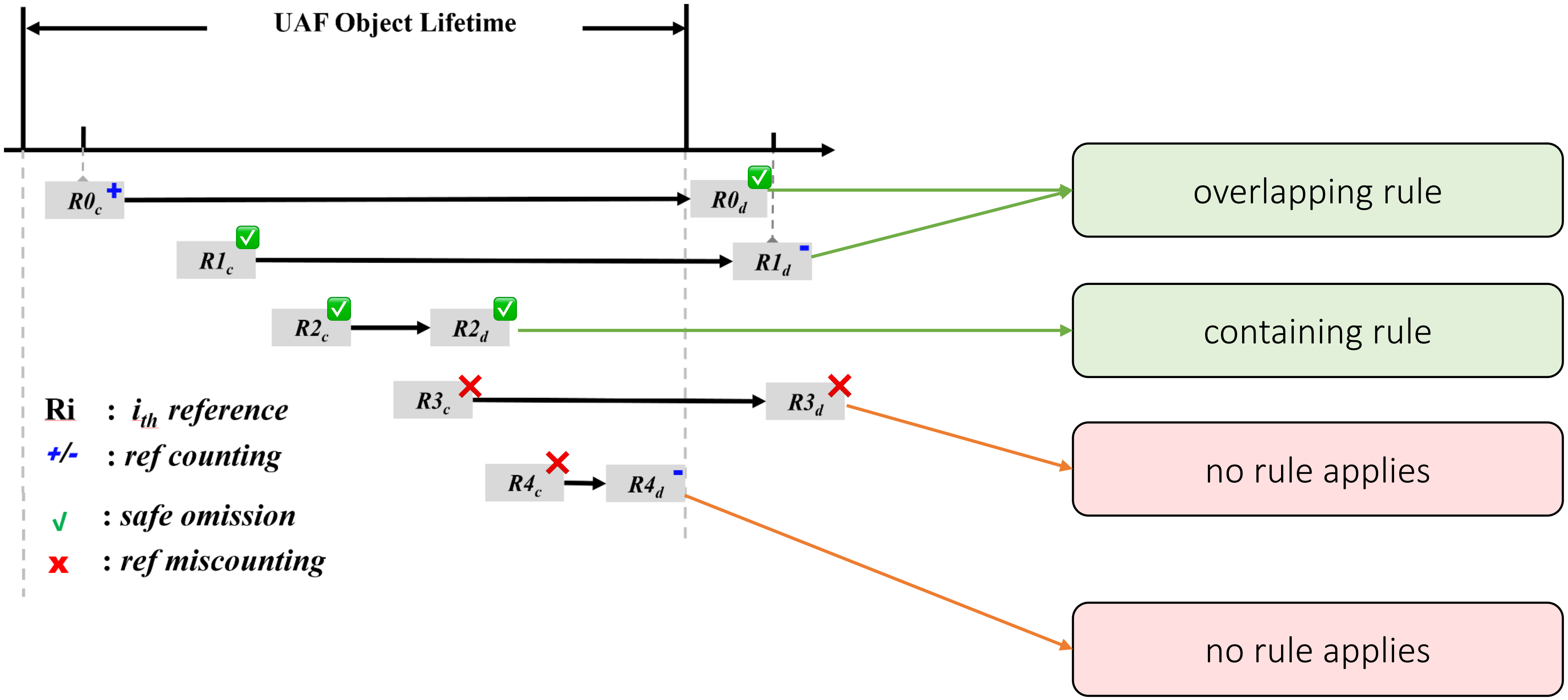
Bug diagnosis algorithm



```
Algorithm 1: UAF Diagnose
Input: ref_set: reference set with matching result
Output: report: diagnose and patch report
rc_set = ∅ ;                                   // inc&dec set
inc_set = ∅ ;                                  // inc set
ov_set = ∅ ;                                   // overlap set
foreach r ∈ ref_set: r.+ and r.- do
    add r into rc_set
foreach r ∈
    add r i
foreach r ∈
    if ∃r'
    then
        vr = (True, r'.$T_c$, True, r.$T_d$)
        add vr into ov_set
        remove r' from inc_set
    else if ∃R1 ... RN ∈ ref_set, ∀1<i⩽N:
      Ri-1.$T_c$ < Ri.$T_c$ < Ri-1.$T_d$ < Ri.$T_d$ and !Ri.+ and !Ri.-
      and RN is r and R1 is r' and r' ∈ inc_set
    then                                       // OR2
        vr = (True, r'.$T_c$, True, r.$T_d$)
        add vr into ov_set
        remove r' from inc_set
        remove Ri (1<i<N) from ref_set
    else
        report ID_BUG ;                        // report
        add + for r ;                          // patch sugg.
                                               // OR3
    then                                       // OR4
        continue to next r
    else
        report MO_BUG ;                        // report
        add +, - for r ;                       // patch sugg.
```

Detailed explanations in paper

Bug diagnosis

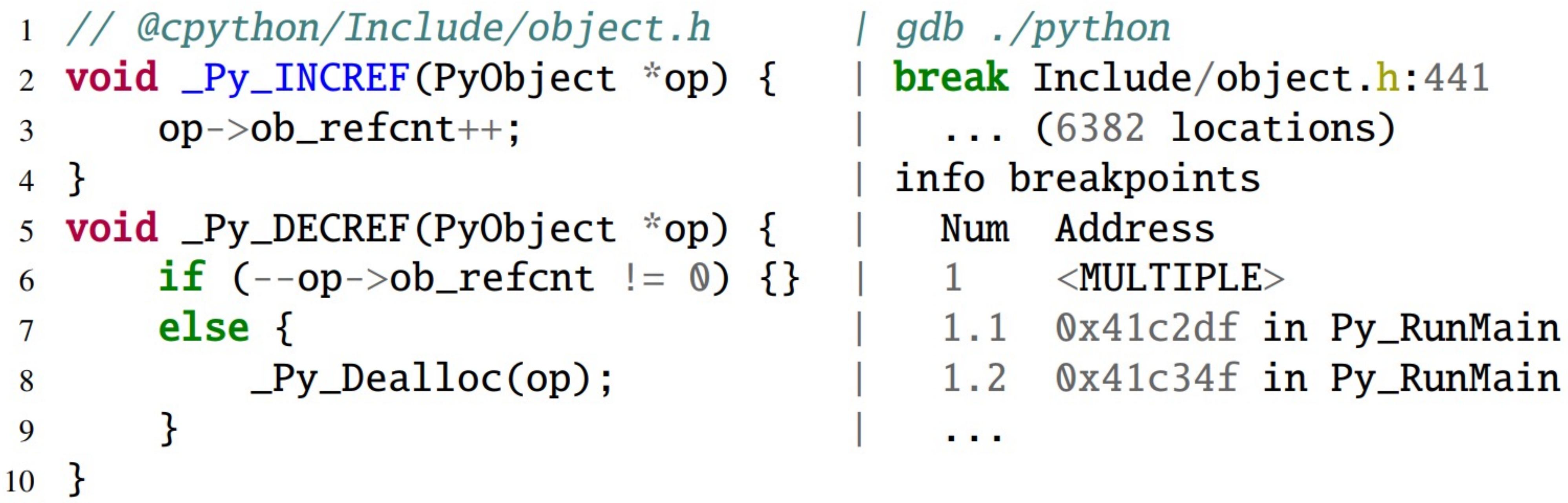


Challenges of **binary-level** diagnosis

- C1: identify reference/refcounting operations
- C2: correlate refcounting to reference creation/destruction

Reference/refcounting detection (w/ source)

- Detecting with debug info and annotation

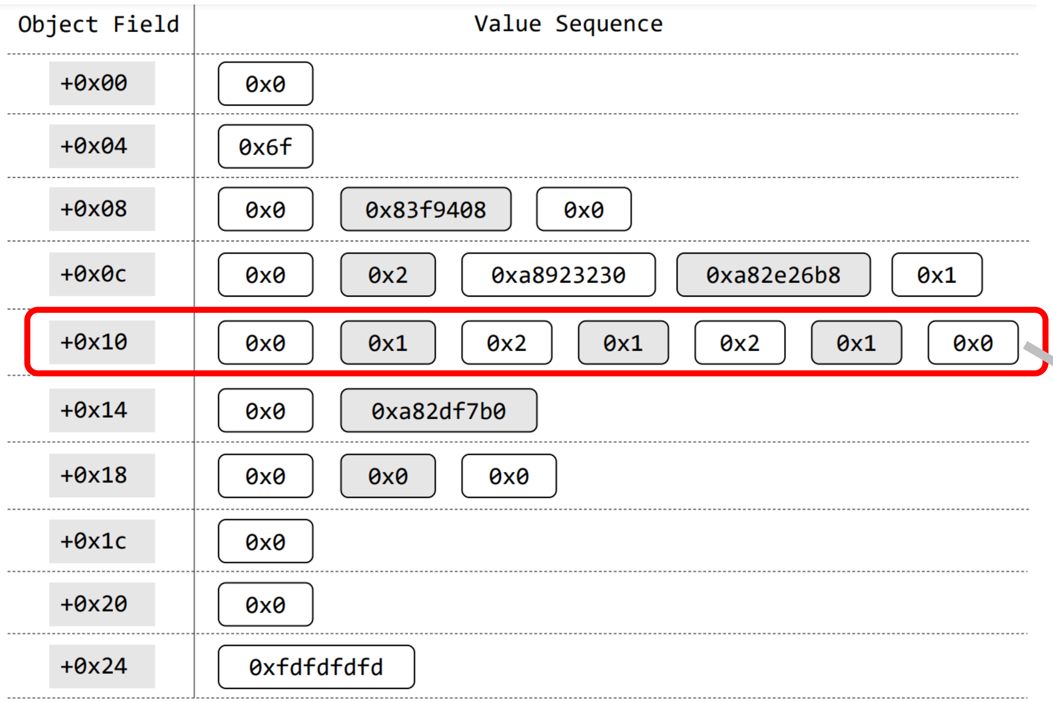


```
// @cpython/Include/object.h        | gdb ./python
void _Py_INCREF(PyObject *op) {     | break Include/object.h:441
    op->ob_refcnt++;                |   ... (6382 locations)
}                                   | info breakpoints
void _Py_DECREF(PyObject *op) {     |   Num  Address
    if (--op->ob_refcnt != 0) {}    |   1    <MULTIPLE>
    else {                          |   1.1  0x41c2df in Py_RunMain
        _Py_Dealloc(op);            |   1.2  0x41c34f in Py_RunMain
    }                               |   ...
}
```

Reference Counting in Python 3.9

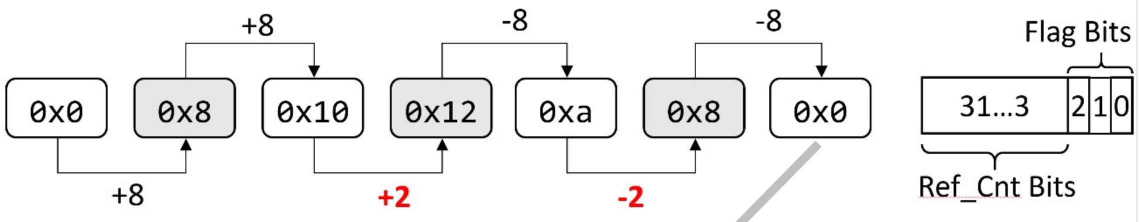
Reference/refcounting detection (w/o source)

- (H1) fix-step changing
  - data-flow analysis
- (H2) control-dependent free
  - control-flow analysis



Object Field	Value Sequence
+0x00	0x0
+0x04	0x6f
+0x08	0x0 0x83f9408 0x0
+0x0c	0x0 0x2 0xa8923230 0xa82e26b8 0x1
+0x10	0x0 0x1 0x2 0x1 0x2 0x1 0x0
+0x14	0x0 0xa82df7b0
+0x18	0x0 0x0 0x0
+0x1c	0x0
+0x20	0x0
+0x24	0xfdfdfdfd

A Python Object Value Sequence

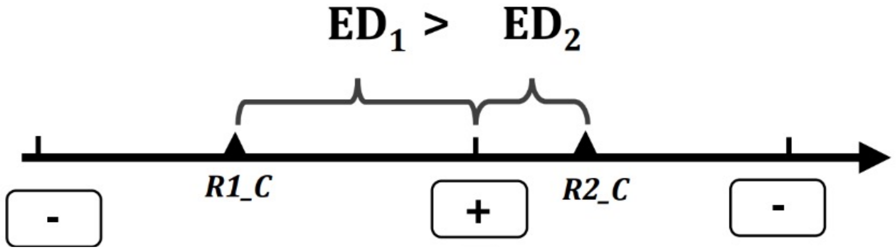


False Negatives (IE-CTreeNode Object)

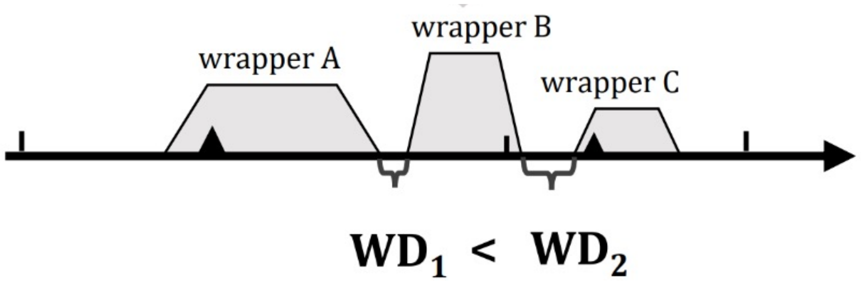
```
if (value == 0)
free(…);
```

Refcounting & reference correlation

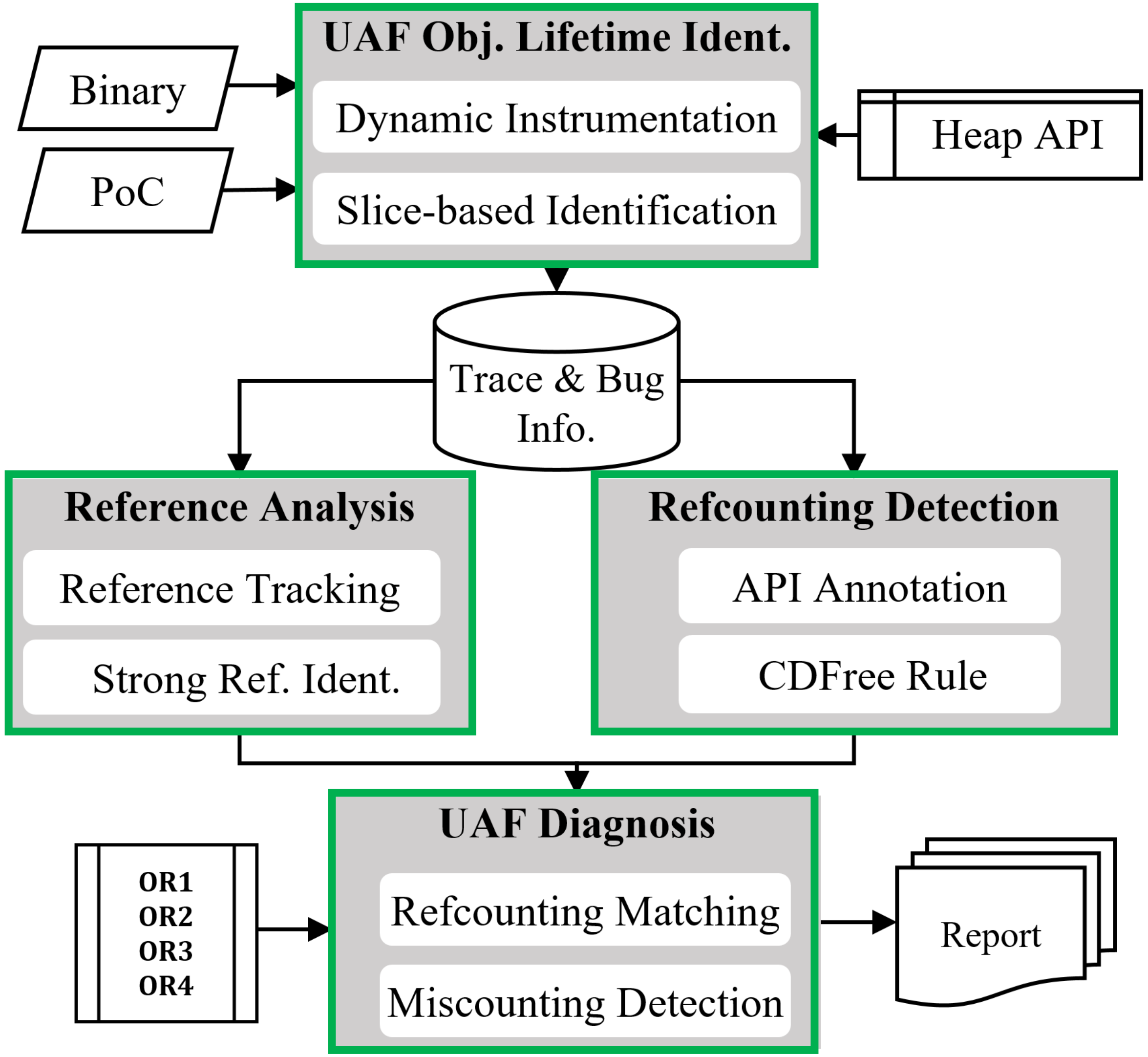
- Based on execution distance (ED)



- Based on wrapper distance (WD)



FreeWill architecture



Evaluation

- Q1: accuracy of root cause diagnosis
- Q2: efficacy of patch suggestion
- Q3: accuracy of reference & refcounting detection

- 76 UAF bugs
  - 32 from Chrome, Firefox and IE
  - 21 from Linux and MacOS
  - 23 from Python and PHP

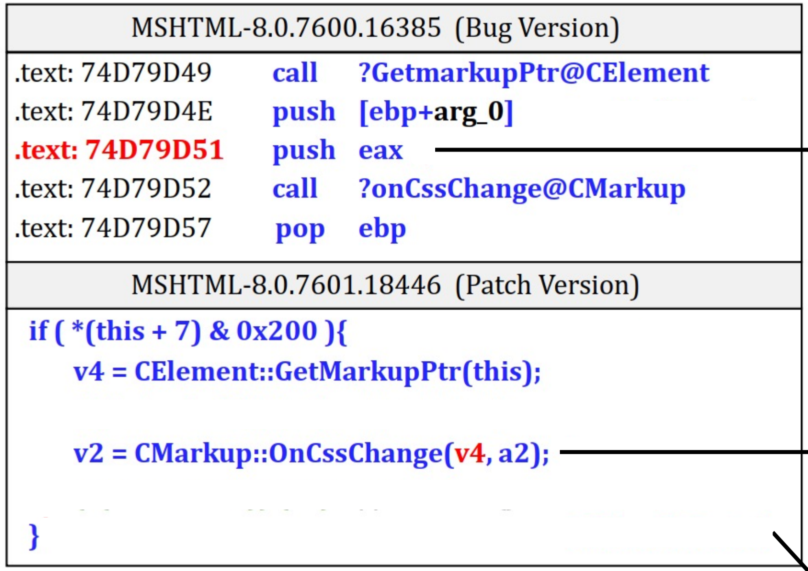
Q1: diagnosis (76 bugs)

- 48 bugs caused by reference miscounting
  - 36 bugs - programmers fail to count the reference (no INC, no DEC)
  - 12 bugs - only decrease but no increase (no INC, has DEC)
- 18 bugs caused by dangling use

Dataset	**Web Browser (32)**			**Kernel (21)**		**Script Engine (23)**	
	IE	**Chrome**	**Firefox**	**Linux**	**MacOS**	**Python**	**PHP**
no INC, no DEC	14	0	0	6	2	10	4
no INC, has DEC	0	0	0	6	3	2	1
dangling use	4	6	4	2	1	0	2
null-deref	0	0	1	1	0	3	0

Q2: patch suggestions

- 56 out of 71 patch suggestions matched with official ones



```
MSHTML-8.0.7600.16385  (Bug Version)
.text: 74D79D49      call   ?GetmarkupPtr@CElement
.text: 74D79D4E      push   [ebp+arg_0]
.text: 74D79D51      push   eax
.text: 74D79D52      call   ?onCssChange@CMarkup
.text: 74D79D57      pop    ebp

MSHTML-8.0.7601.18446  (Patch Version)
 if ( *(this + 7) & 0x200 ){
     v4 = CElement::GetMarkupPtr(this);

     v2 = CMarkup::OnCssChange(v4, a2);

 }
```

Patch Suggestion (CVE-2014-1776)

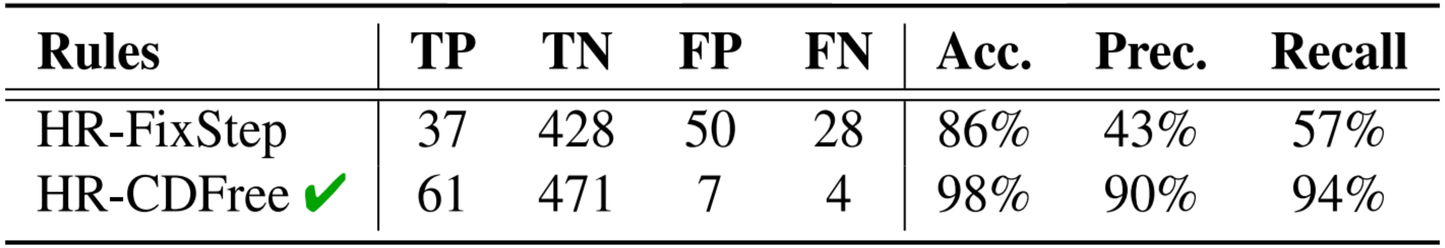
Diagnose: PUSH creates a new reference, but no refcounting.

Patch: add increment and decrement to argument

Official patch: call AddRef() and Release() to increase and decrease the refcount of argument

Q3: reference & refcounting detection

- On average, each UAF object has 2000 references
- Along one trace, 543 objects created, 65 refcounted
- Accuracy of counter detection



Rules	TP	TN	FP	FN	Acc.	Prec.	Recall
HR-FixStep	37	428	50	28	86%	43%	57%
HR-CDFree ✓	61	471	7	4	98%	90%	94%

Conclusion: FreeWill

- Diagnosing UAF bugs due to premature free
  - identify *reference miscounting* as a common reason
  - automatically detect reference miscounting
  - suggest possible patches
- Evaluation on large programs/systems
  - complete analysis within 15 minutes
  - 56 patch suggestions

Thanks & Questions

heliang@iscas.ac.cn honghu@psu.edu