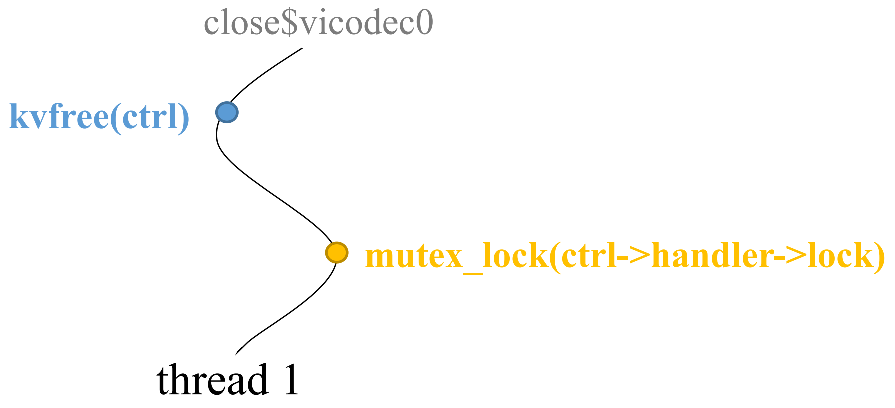
DDRace: Finding Concurrency UAF Vulnerabilities in Linux Drivers with Directed Fuzzing

Ming Yuan[1] , Bodong Zhao[1] , Penghui Li[2], Jiashuo Liang[3], Xinhui Han[3], Xiapu Luo[4], Chao Zhang[1]

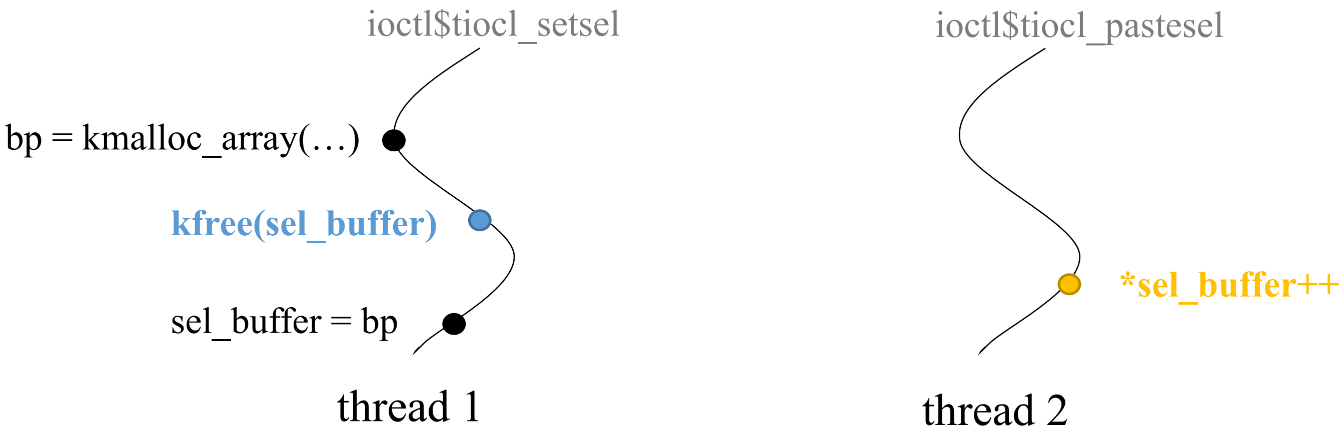
1 Tsinghua University
2 The Chinese University of Hong Kong
3 Peking University
4 The Hong Kong Polytechnic University

Background

- Sequential Use-After-free
  - use-after-free vulnerabilities in single thread

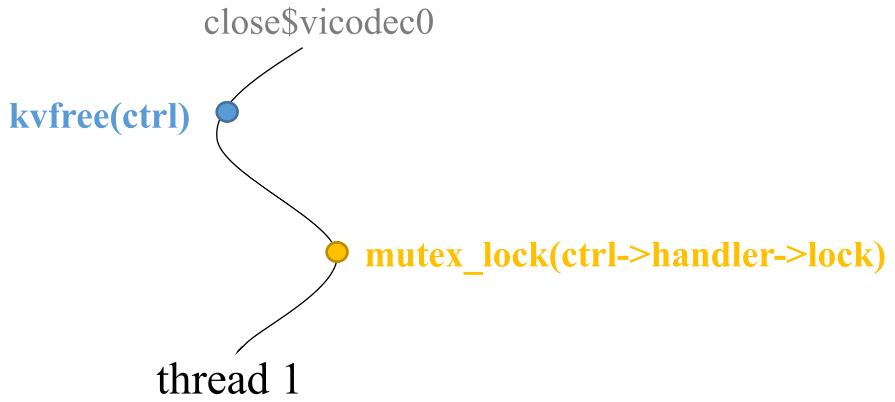


- Concurrency Use-After-free
  - use-after-free vulnerabilities caused by concurrency bugs

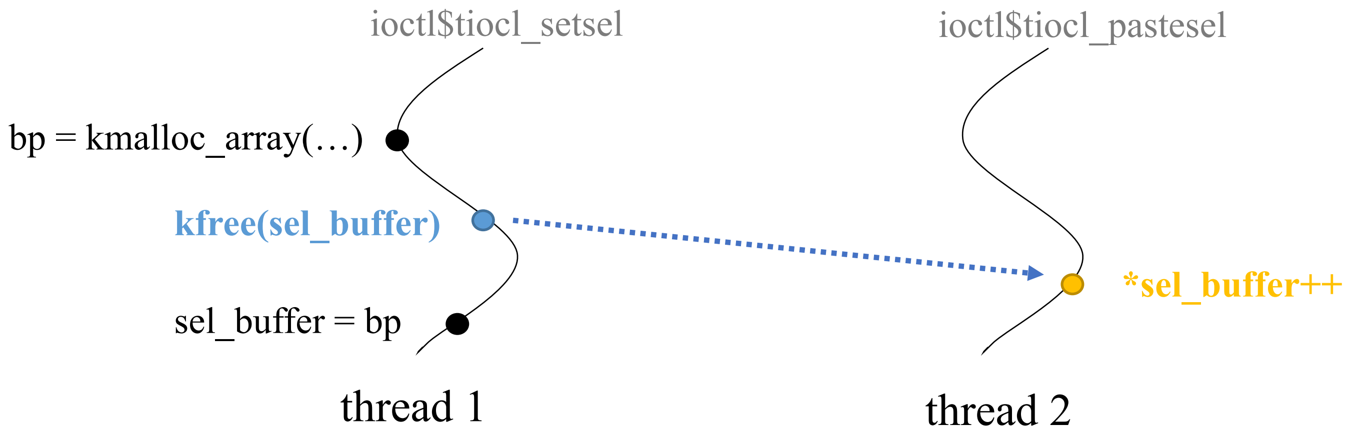


Background

- Sequential Use-After-free
  - use-after-free vulnerabilities in single thread



- Concurrency Use-After-free
  - use-after-free vulnerabilities caused by concurrency bugs



Motivation Example

```
spin_lock_irqsave(…);
q = func_table[i];

delta = (q ? -strlen(q) : 1) +
        strlen(kbs->kb_string);

if (delta <= funcbufleft)  {
        ...

} else {

  fnw = kmalloc(sz, GFP_KERNEL);

// funcbufptr holds the old func_table[i]

  func_table[i] = fnw + ...
    ...
  kfree(funcbufptr);
}

strcpy(func_table[i],kbs->kb_string);
spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

```
p = func_table[i];
    ...

for (;*p && sz; p++, sz--)
    ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
           strlen(kbs->kb_string);

2060. if (delta <= funcbufleft)  {
           ...

2071. } else {

2078.   fnw = kmalloc(sz, GFP_KERNEL);

  // funcbufptr holds the old func_table[i]

2094.   func_table[i] = fnw + ...
          ...
2104.   kfree(funcbufptr);
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

```
2023. p = func_table[i];
          ...

2025. for (;*p && sz; p++, sz--)
            ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
              strlen(kbs->kb_string);

2060. if (delta <= funcbufleft)  {
              ...

2071. } else {

2078.    fnw = kmalloc(sz, GFP_KERNEL);

  // funcbufptr holds the old func_table[i]

2094.    func_table[i] = fnw + ...        I1
            ...
2104.    kfree(funcbufptr);               I2
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

```
2023. p = func_table[i];                  I3
          ...

2025. for (;*p && sz; p++, sz--)          I4
            ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
             strlen(kbs->kb_string);

2060. if (delta <= funcbufleft)  {
             ...

2071. } else {

2078.   fnw = kmalloc(sz, GFP_KERNEL);

  // funcbufptr holds the old func_table[i]

2094.   func_table[i] = fnw + ...        I1
          ...
2104.   kfree(funcbufptr);               I2
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

- Observation:
  - the exploration space is infinite
  - code coverage or input distance metric have limitations in fuzzing concurrency targets

```
2023. p = func_table[i];                  I3
        ...
2025. for (;*p && sz; p++, sz--)          I4
          ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
              strlen(kbs->kb_string);

2060. if (delta <= funcbufleft)  {
          ...

2071. } else {

2078.   fnw = kmalloc(sz, GFP_KERNEL);

  // funcbufptr holds the old func_table[i]

2094.   func_table[i] = fnw + ...
          ...
2104.   kfree(funcbufptr);
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

- Observation:
  - the exploration space is infinite
  - code coverage or input distance has limitations in fuzzing concurrency targets

```
2023. p = func_table[i];
          ...

2025. for (;*p && sz; p++, sz--)
          ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
              strlen(kbs->kb_string);

2060. if (delta <= funcbufleft)  {
              ...

2071. } else {

2078.   fnw = kmalloc(sz, GFP_KERNEL);

  // funcbufptr holds the old func_table[i]

2094.   func_table[i] = fnw + ...
           ...
2104.   kfree(funcbufptr);
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

- Observation:
  - the exploration space is infinite
  - code coverage or input distance has limitations in fuzzing concurrency targets

```
2023. p = func_table[i];
          ...

2025. for (;*p && sz; p++, sz--)
          ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
              strlen(kbs->kb_string);

2060. if (delta <= funcbufleft)  {
             ...

2071. } else {

2078.    fnw = kmalloc(sz, GFP_KERNEL);

  // funcbufptr holds the old func_table[i]

2094.    func_table[i] = fnw + ...
           ...
2104.    kfree(funcbufptr);
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

- Observation:
  - the exploration space is infinite
  - code coverage or input distance has limitations in fuzzing concurrency targets
  - state aging causes poor reproducibility of seeds

```
2023. p = func_table[i];
         ...

2025. for (;*p && sz; p++, sz--)
          ...
```

ioctl$KDGKBSENT

Motivation Example

```
2046. spin_lock_irqsave(…);
2047. q = func_table[i];

2058. delta = (q ? -strlen(q) : 1) +
              strlen(kbs->kb_string);

2060. if (delt
          ...

2071. } else {

2078.    fnw =

  // funcbufptr holds the old func_table[i]

2094.    func_table[i] = fnw + ...
           ...
2104.    kfree(funcbufptr);
      }

2110. strcpy(func_table[i],kbs->kb_string);
2111. spin_unlock_irqrestore(…);
```

ioctl$KDSKBSENT

- Observation:
  - the exploration space is infinite
  - code coverage or input distance has limitations in fuzzing concurrency targets
  - oducibility of seeds

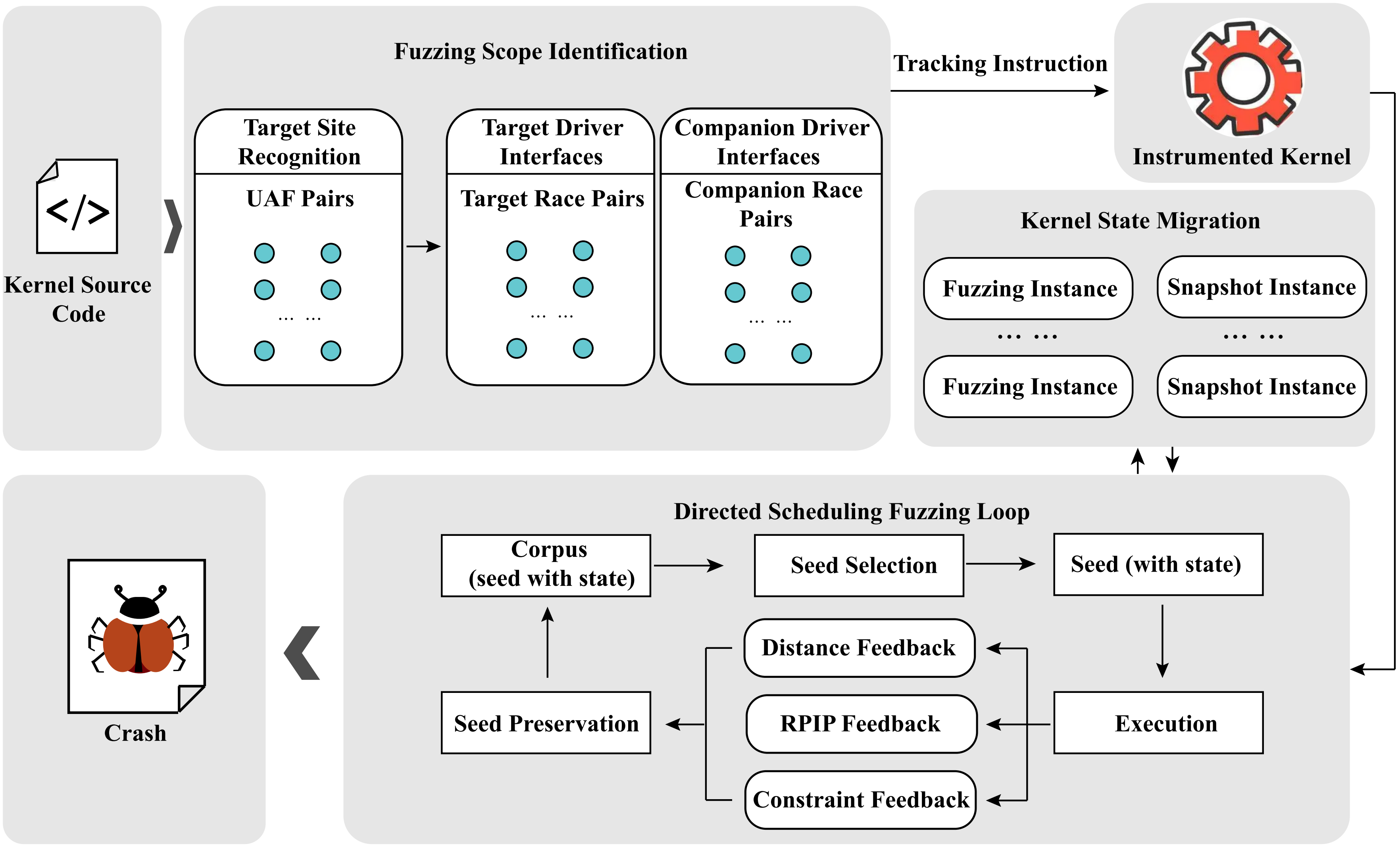
```
2023. p = func_table[i];
        ...

2025. for (;*p && sz; p++, sz--)
        ...
```

ioctl$KDGKBSENT

In order to efficiently discover concurrency UAF vulnerabilities with limited computing resources, we need a concurrency directed fuzzing solution.

Our Approach: DDRace

- Concurrency directed fuzzer for finding concurrency UAF in Linux driver



Fuzzing Scope Identification

- Target Sites Recognition
  - lightweight dynamic trace analysis
  - instrument and monitor USE and FREE operations

ioctl$VT_DISALLOCATE

```
if (i != fg_console)
{
    vc = vc_cons[i].d;
    kfree(vc);
}
```

ioctl$PIO_FONTX

```
vc = vc_cons[fg_console].d;
        ...
if (vc->mode != KD_TEXT)
```

Fuzzing Scope Identification

- Target Race Pairs Extraction
  - inter-procedural static analysis
  - identify target driver interfaces

ioctl$VT_DISALLOCATE

```
if (i != fg_console)
{
   vc = vc_cons[i].d;
   kfree(vc);
}
```

ioctl$PIO_FONTX

```
vc = vc_cons[fg_console].d;
        ...
if (vc->mode != KD_TEXT)
```

Fuzzing Scope Identification

- Target Race Pairs Extraction
  - inter-procedural static analysis
  - identify target driver interfaces
  - identify target race pairs
    - located in the control flow paths from target driver interfaces to target sites

ioctl$VT_DISALLOCATE

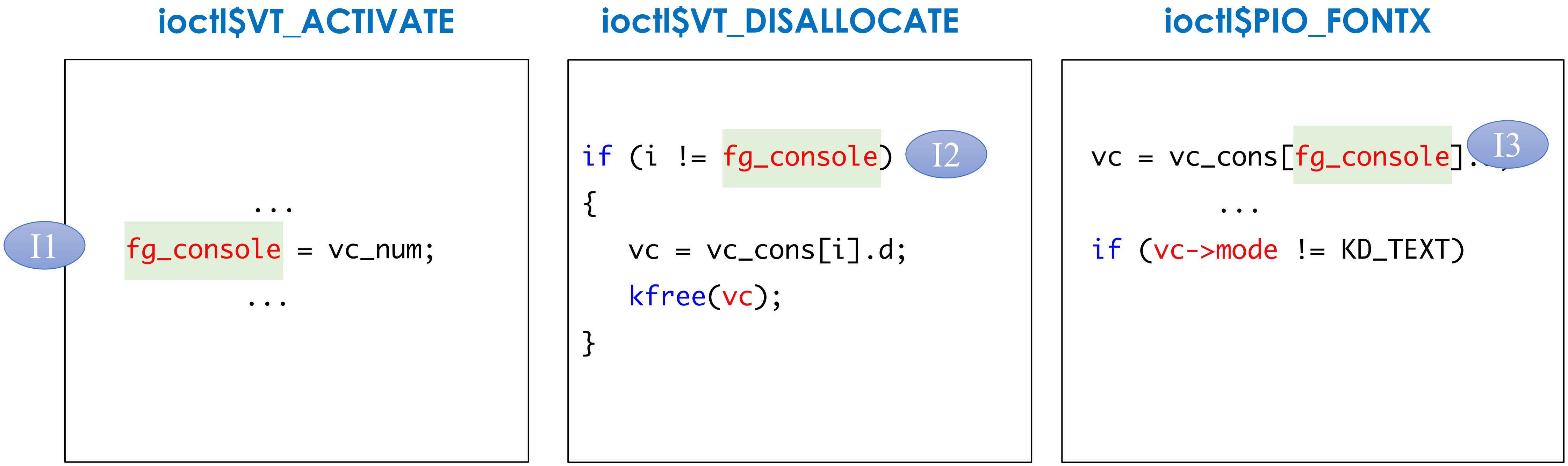
```
if (i != fg_console)
{
   vc = vc_cons[i].d;
   kfree(vc);
}
```

ioctl$PIO_FONTX

```
vc = vc_cons[fg_console].d;
       ...
if (vc->mode != KD_TEXT)
```

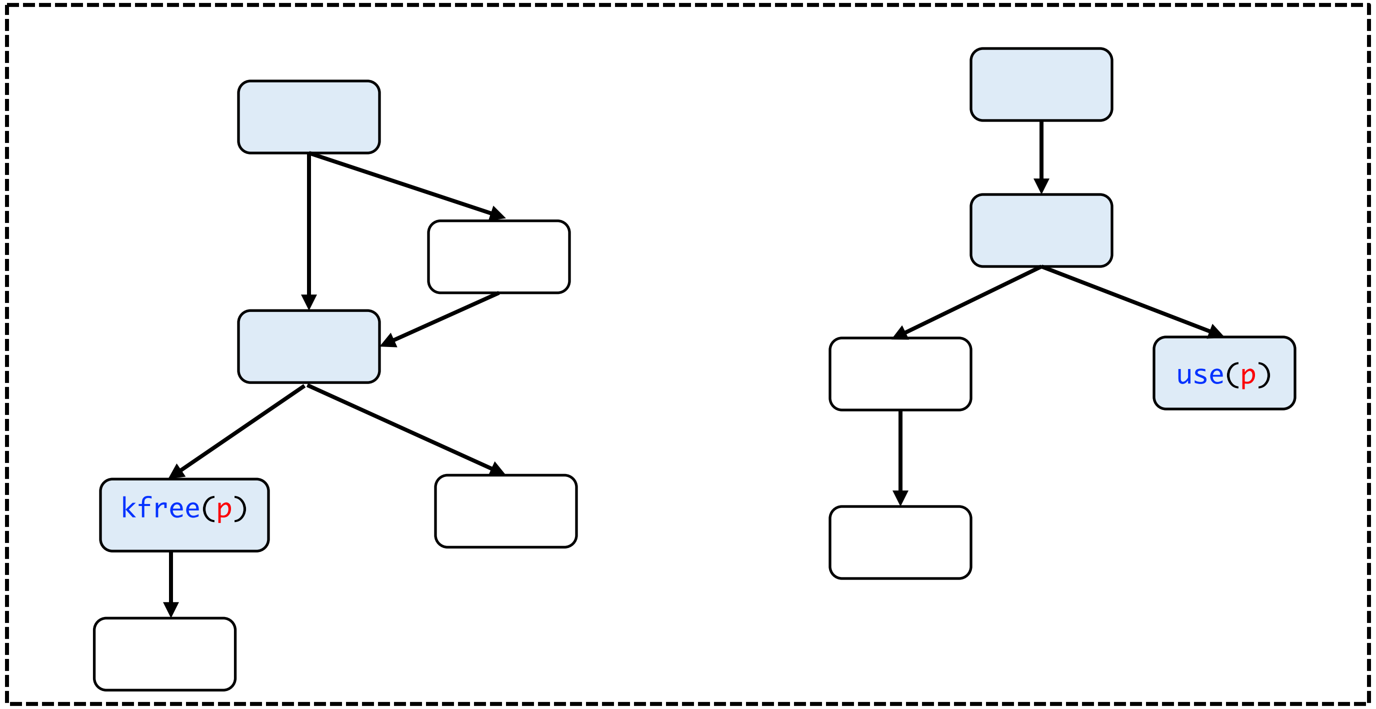
Fuzzing Scope Identification

- Companion Race Pairs Extraction
  - identify write operations located in other driver interfaces
  - companion race pairs: <I1, I2>, <I1, I3>
  - concurrency companion driver interfaces: ioctl$VT_ACTIVATE



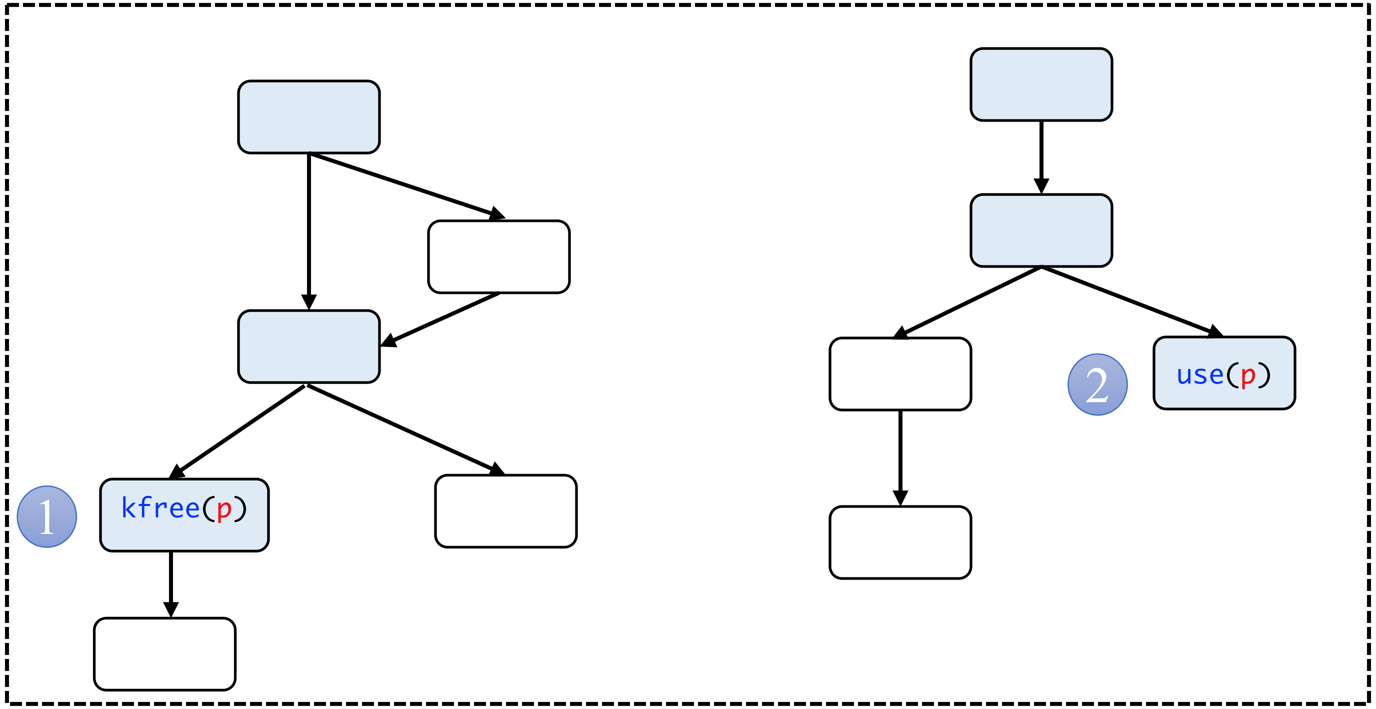
Directed Scheduling Fuzzing Loop

- Distance Metrics and Feedback Mechanisms
  - Dominator Depth Distance



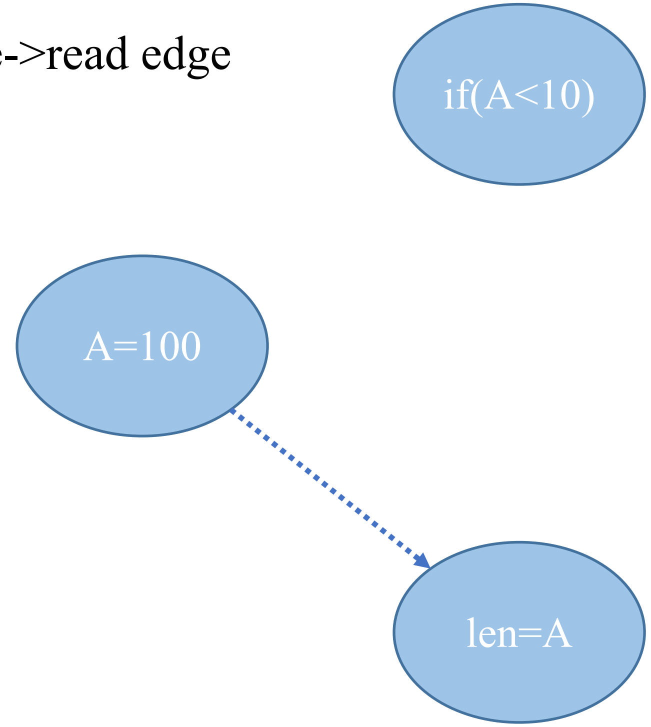
Directed Scheduling Fuzzing Loop

- Distance Metrics and Feedback Mechanisms
  - Dominator Depth Distance
  - Vulnerability Model Constraint Distance
    - Value: FREE and USE operates the same memory
    - Order: USE is executed after FREE



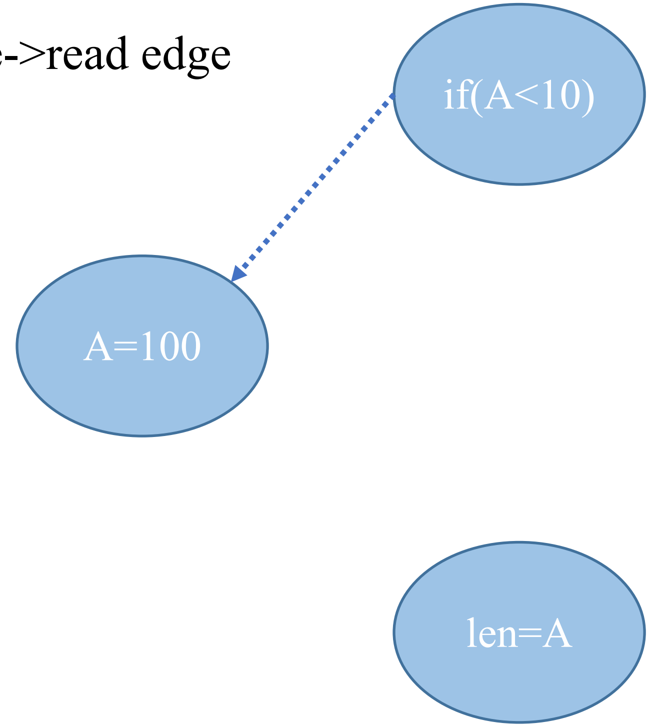
Directed Scheduling Fuzzing Loop

- Distance Metrics and Feedback Mechanisms
  - Race Pair Interleaving Path (RPIP) Feedback
    - thread-interleaving edge
      - analogous to code branch edge
      - focus on read->write (only value changed matters) and write->read edge



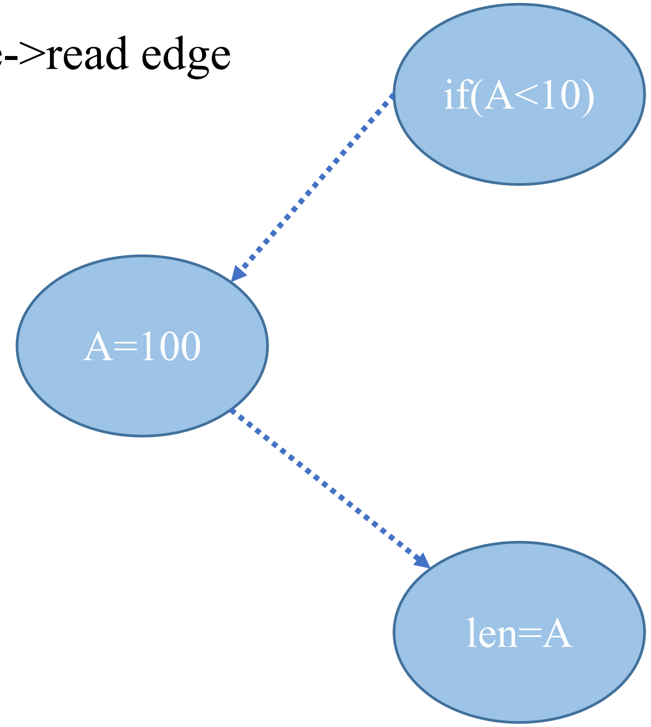
Directed Scheduling Fuzzing Loop

- Distance Metrics and Feedback Mechanisms
  - Race Pair Interleaving Path (RPIP) Feedback
    - thread-interleaving edge
      - analogous to code branch edge
      - focus on read->write (only value changed matters) and write->read edge



Directed Scheduling Fuzzing Loop

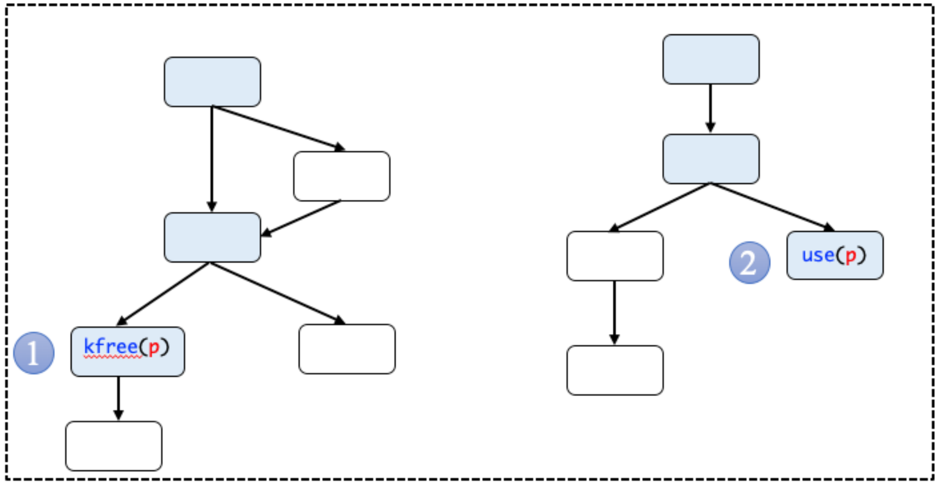
- Distance Metrics and Feedback Mechanisms
  - Race Pair Interleaving Path (RPIP) Feedback
    - thread-interleaving edge
      - analogous to code branch edge
      - focus on read->write (only value changed matters) and write->read edge
    - race pair interleaving path



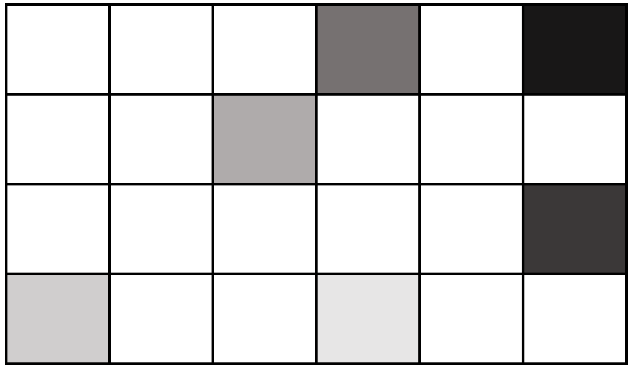
Directed Scheduling Fuzzing Loop

- Seed Selection
  - lower distance value
    - closer to targets
  - infrequent RPIP
    - exploring rare thread interleaving

distance metric

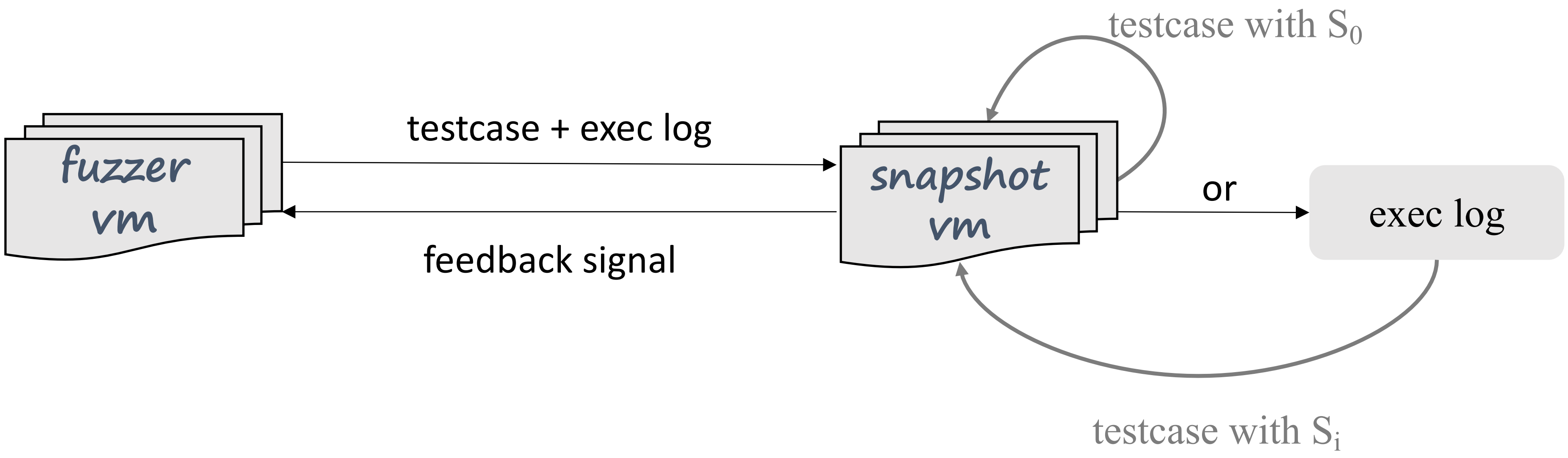


RPIP metric



Adaptive State Migration

- Utilize qemu's snapshot feature
- Trade-off between accuracy and overhead
  - only concerned with highly valued testcase
  - prefer using the initial state (S_0)

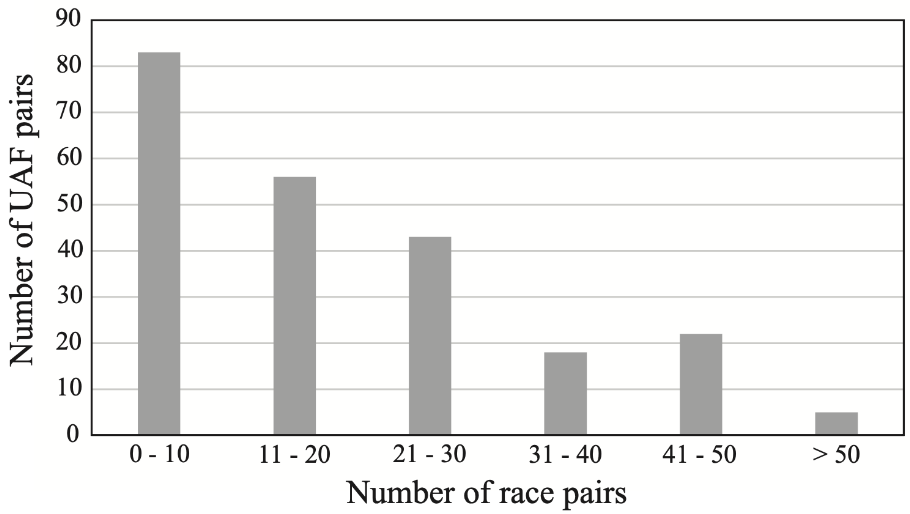


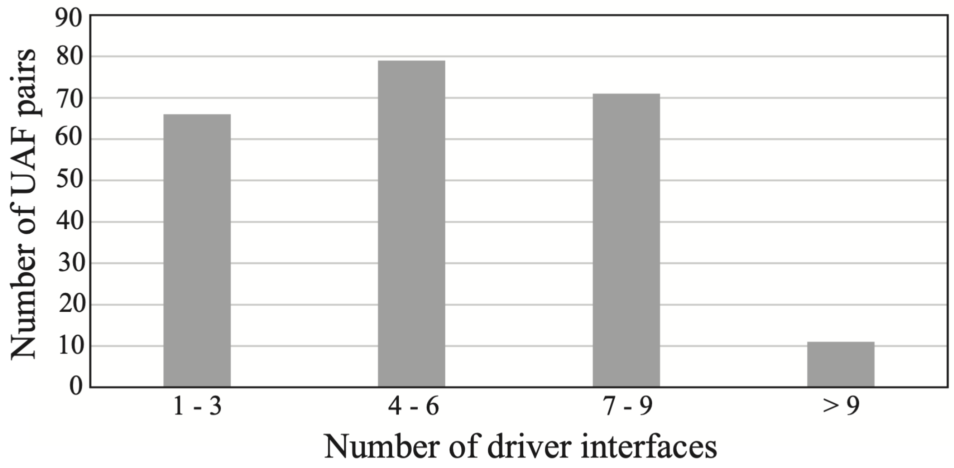
Evaluation

- RQ1: How effective is DDRace at extracting target-related race pairs ?
- RQ2: What is the capability of DDRace in exposing concurrency UAF vulnerabilities?
- RQ3: How is DDRace comparable to existing approaches?
- Experiments for 6 drivers of Linux upstream kernel v4.19 on qemu-system-x86_64
  - tty, drm, sequencer, midi , vivid and floppy

Race Pair Extraction (RQ1)

- Run original Syzkaller for 24h, obtain 227 UAF pairs
- Race Pair statics
  - target race pairs + companion race pairs
  - target driver interfaces + concurrency companion driver interfaces





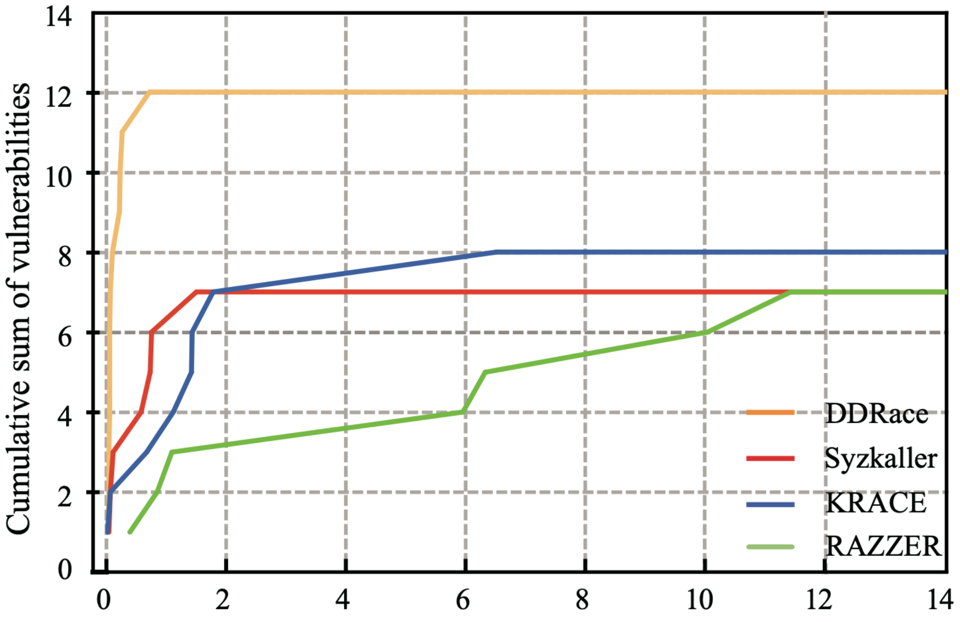
Vulnerability Discovery (RQ2)

- DDRace found 12 concurrency UAF in 6 drivers
  - 3 CVEs

Vul. ID	Driver	File Names of Target Pairs	Status
1	drm	drivers/gpu/drm ↔ drivers/gpu/drm	Confirmed†
2	drm	drivers/gpu/drm/drm_gem.c ↔ drivers/gpu/drm/vgem/vgem_drv.c	Confirmed & Fixed†
3	drm	drivers/gpu/drm/drm_gem.c ↔ drivers/gpu/drm/vkms/vkms_gem.c	known
4	drm	drivers/gpu/drm/drm_auth.c ↔ drivers/gpu/drm/drm_ioctl.c	known
5	floppy	drivers/block/floppy.c ↔ drivers/block/floppy.c	Confirmed & Fixed†
6	floppy	drivers/block/floppy.c ↔ drivers/block/floppy.c	Confirmed & Fixed†
7	tty	drivers/tty/vt/selection.c ↔ drivers/tty/n_tty.c	known
8	tty	drivers/tty/vt/keyboard.c ↔ drivers/tty/vt/keyboard.c	known
9	tty	drivers/tty/vt/vt_ioctl.c ↔ drivers/tty/vt/vt.c	known
10	tty	drivers/tty/vt/vt_ioctl.c ↔ drivers/tty/vt/vt.c	known
11	sequencer	sound/core/seq/seq_ports.c ↔ sound/core/seq/seq_ports.c	known
12	midi	sound/core/rawmidi.c ↔ sound/core/rawmidi.c	known

Comparison (RQ3)

- Comparision with Syzkaller, RAZZER, KRACE
- Vulnerability Findings
  - DDRace outperforms in vulnerability detection by 66.7%, 66.7% and 50%, respectively
- Vulnerability Triggering Time



Conclusion

- A new concurrency directed fuzzing solution DDRace for finding concurrency UAF in Linux drivers
- DDRace identifies the fuzzing scope to reduce code space & thread interleaving space
- A new vulnerability-related distance metric and a novel concurrency feedback mechanism to assist directed fuzzing
- A new adaptive kernel state migration scheme to ensure the reproducibility of seeds
- Outperform existing fuzzers

Thanks for listening!

Q & A