Effective Static Analysis of Concurrency Use-After-Free Bugs in Linux Device Drivers

Jia-Ju Bai[1], Julia Lawall[2], Qiu-Liang Chen[1], Shi-Min Hu[1]

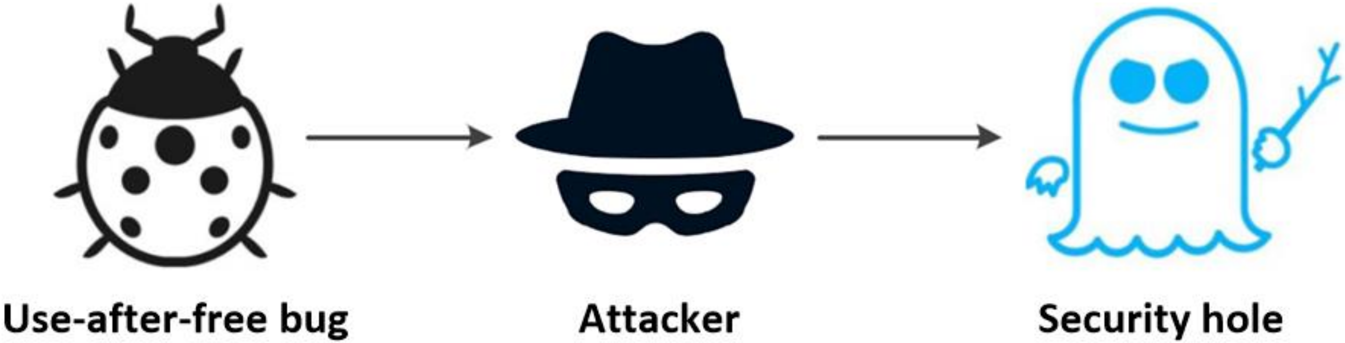
[1] Tsinghua University, [2] Sorbonne University/Inria/LIP6

清華大學
Tsinghua University

inventors for the digital world

Background

- Use-after-free bugs in device drivers
  - Reliability: may cause system crashes
  - Security: can be exploited to attack the operating system



Background

- Sequential use-after-free bug

```
void DriverExit(struct device *pdev) {
    kfree(pdev->buf);
    pdev->num = 0;
    pdev->buf->last = NULL;
}
```

Thread 1

- Concurrency use-after-free bug

```
void DriverFunc1(struct device *pdev) {
    kfree(pdev->buf);
    pdev->buf = kmalloc(...)
    pdev->buf->last = NULL;
}
```

Thread 1

```
void DriverFunc2(struct device *pdev) {
    spin_lock(...);
    pdev->buf->first = NULL;
    spin_unlock(...);
}
```

Thread 2

Example

- Linux cw1200 driver

```
FILE: linux-4.19/drivers/net/wireless/st/cw1200/main.c
208. static const struct ieee80211_ops cw1200_ops = {
         ......
215.     .hw_scan = cw1200_hw_scan,
         ......
223.     .bss_info_changed = cw1200_bss_info_changed,
         ......
238. };
```

```
FILE: linux-4.19/drivers/net/wireless/st/cw1200/scan.c
 54. int cw1200_hw_scan(...) {
         ......
 91.     mutex_lock(&priv->conf_mutex);
         ......
123.     mutex_unlock(&priv->conf_mutex);
125.     if (frame.skb)
126.         dev_kfree_skb(frame.skb); // FREE
         ......
129. }
```

```
FILE: linux-4.19/drivers/net/wireless/st/cw1200/sta.c
1799. void cw1200_bss_info_changed(...) {
          ......
1807.     mutex_lock(&priv->conf_mutex);
          ......
1849.     cw1200_upload_beacon(...);
          ......
2075.     mutex_unlock(&priv->conf_mutex);
          ......
2081. }
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
2189. static int cw1200_upload_beacon(...) {
          ......
2221.     mgmt = (void *)frame.skb->data; // READ
          ......
2238. }
```

Lifetime: Sep. 2013 ~ Dec.2018

Fix Commit: 4f68ef64cd7f

Study of Linux kernel commits

- Use-after-free commits
  - Jan.2016 ~ Dec.2018 (3 years)

Time	**Commits**	**Drivers**	**Concurrency**	**Tool use**
2016 (Jan - Dec)	186	111	42 (38%)	26
2017 (Jan - Dec)	478	205	87 (42%)	49
2018 (Jan - Dec)	285	145	66 (46%)	52
Total	949	461	195 (42%)	127

42% of driver commits fixing use-after-free bugs involve concurrency

Study of Linux kernel commits

- Tool use
  - Tools mentioned in driver commits

Tool use	KASAN	Syzkaller	Coverity	Coccinelle	LDV
Type	Runtime	Runtime	Static	Static	Static
Commit	92	28	4	2	1
Concurrency	38	18	0	0	0

It is important to explore static analysis to detect concurrency use-after-free bugs in device drivers!

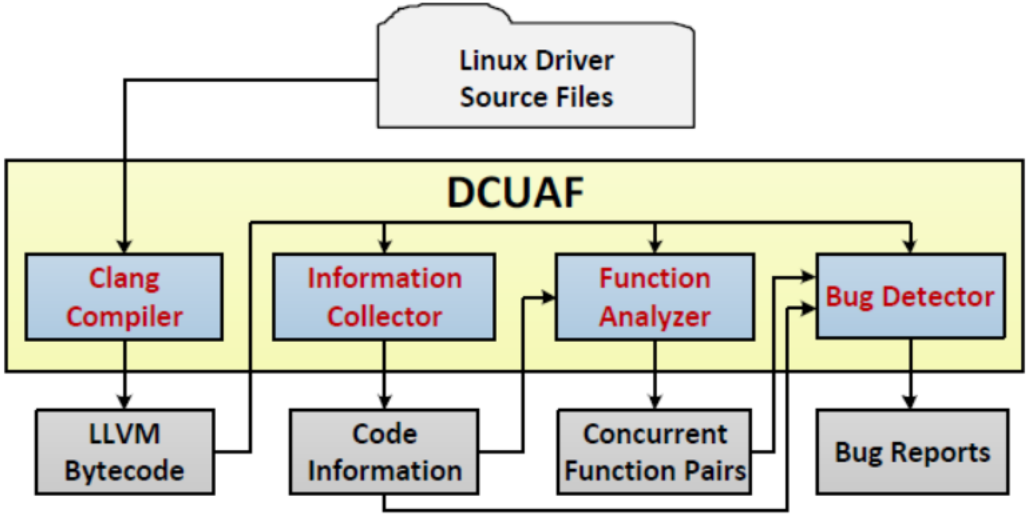
Challenges

- Identify driver functions that can be concurrently executed
  - Poor documentation about concurrency
  - Many functions defined in the driver code

- Accuracy and efficiency of code analysis
  - Large size of the Linux driver code base
  - Many function calls across different source files

Approach

- DCUAF
  - Automated and effective approach of detecting concurrency use-after-free bugs in device drivers
  - LLVM-based static analysis



Approach

- Basic idea
  - Step1: Use a ***local-global strategy*** to identify concurrent function pairs from driver source code
  - Step2: Use a ***summary-based lockset analysis*** to detect concurrency use-after-free bugs.

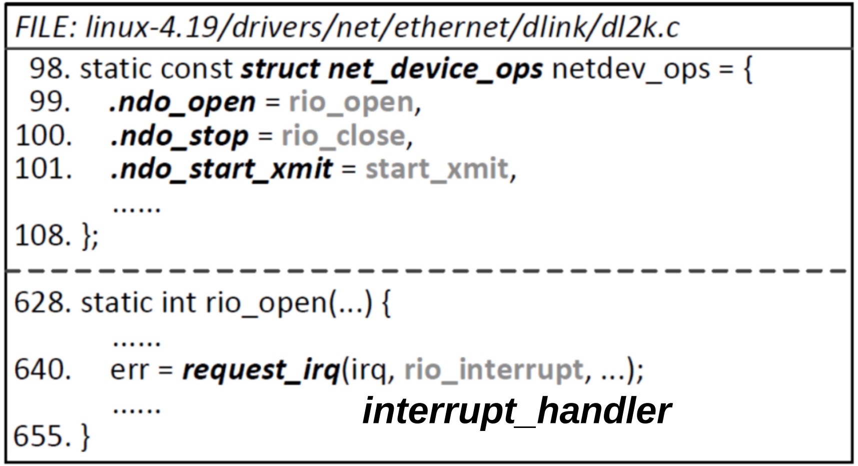
Local-global strategy

- Driver interfaces are the entries of a device driver
  - Kernel-driver interfaces
  - Interrupt handler interfaces

- Driver concurrency is often determined by the concurrent execution of driver interfaces

Local-global strategy

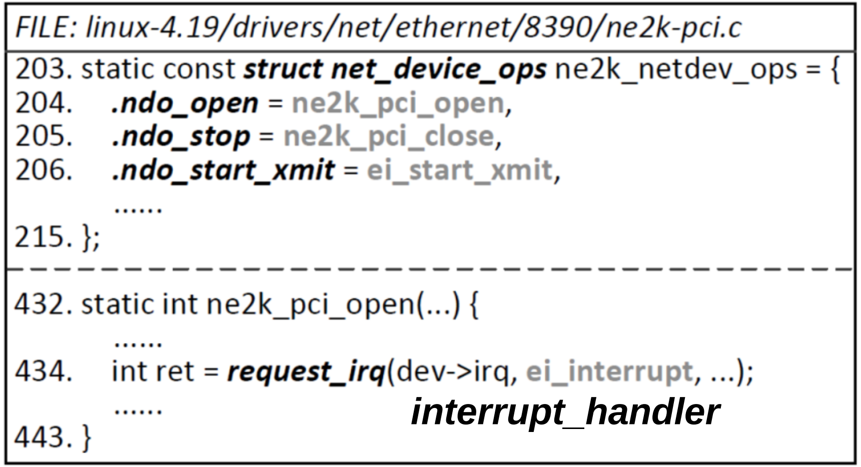
- Examples
  - Linux dl2k and ne2k-pci drivers



FILE: linux-4.19/drivers/net/ethernet/dlink/dl2k.c

```
static const struct net_device_ops netdev_ops = {
    .ndo_open = rio_open,
    .ndo_stop = rio_close,
    .ndo_start_xmit = start_xmit,
    ......
};
---------------------------------------------
static int rio_open(...) {
    ......
    err = request_irq(irq, rio_interrupt, ...);
    ......
}
```

interrupt_handler



FILE: linux-4.19/drivers/net/ethernet/8390/ne2k-pci.c

```
static const struct net_device_ops ne2k_netdev_ops = {
    .ndo_open = ne2k_pci_open,
    .ndo_stop = ne2k_pci_close,
    .ndo_start_xmit = ei_start_xmit,
    ......
};
---------------------------------------------
static int ne2k_pci_open(...) {
    ......
    int ret = request_irq(dev->irq, ei_interrupt, ...);
    ......
}
```

interrupt_handler

> “.ndo_start_xmit” can be concurrently executed with “interrupt handler”

> “.ndo_open” is never concurrently executed with “.ndo_close”

Local-global strategy

- How to extract concurrent function pairs?
  - **Local stage:** analyze the source code of each driver
  - **Global stage:** statistically analyze the local results of all drivers

Local stage

- S1: identify possible concurrent function pairs
  - Compare lock-acquiring function calls

- S2: drop possibly false concurrent function pairs
  - Collect “ancestors” of the two functions in call graph
  - Drop pairs of functions that have a common “ancestor”

- S3: extract *local concurrent interface pairs*
  - Identify and record driver interface assignments related to concurrent function pairs

Global stage

- S1: gather local concurrent interface pairs of all drivers
- S2: statistically extract *global concurrent interface pairs*
  - Ratio: concurrent pairs / all pairs

Driver Interface 1	Driver Interface 2	Pair	Concurrent	
spi_driver.probe	spi_driver.remove	227	3	✗
file_operations.open	file_operations.close	462	3	✗
hc_driver.urb_enqueue	hc_driver.endpoint_disable	16	9	✓
Interrupt handler	snd_pcm_ops.trigger	49	25	✓

- S3: identify concurrent function pairs in each driver

Summary-based lockset analysis

- Context-sensitive and flow-sensitive lockset analysis
  - Maintain locksets
- Field-based alias analysis
  - Identify the same locks
- Summary-based analysis
  - Reuse the results of already analyzed functions
- Procedure
  - S1: collect the lockset of each variable access
  - S2: check the held locksets of the variable accesses to find bugs

Evaluation

- Driver code in Linux 3.14 and 4.19
  - Use a common PC with four CPUs
  - Run on four threads
  - Make *allyesconfig* of x86

Evaluation

- Local-global strategy

Description		Linux 3.14	Linux 4.19
Code handling	Source files (.c)	7957	13100
	Source code lines	5.1M	7.9M
Local stage	Dropped function pairs	61.4K	99.8K
	Remaining function pairs	40.7K	67.8K
Global stage	Global concurrent interface pairs	694	1497
	Concurrent function pairs	15.6K	69.5K
Time usage		15m	18m

Evaluation

- Bug detection

Description	**Linux 3.14**	**Linux 4.19**
Detected (real / all)	526 / 559	640 / 679
Confirmed / reported	-	95 / 130
Time usage	9m	10m

Some confirmed bugs:

- https://github.com/torvalds/linux/commit/7418e6520f22
- https://github.com/torvalds/linux/commit/2ff33d663739
- https://github.com/torvalds/linux/commit/c85400f886e3

Evaluation

- False positives
  - Alias analysis may incorrectly identify the same locks
  - Flow-sensitive analysis does not validate path conditions
  - ……

- False negatives
  - Function-pointer analysis is not performed
  - Other kinds of synchronization are neglected
  - ……

Conclusion

- Concurrency use-after-free bugs are often hard to detect

- DCUAF: automated and effective
  - Local-global strategy of extracting concurrent function pairs
  - Summary-based lockset analysis

- Find hundreds of new real bugs in Linux device drivers