How Double-Fetch Situations turn into Double-Fetch Vulnerabilities: A Study of Double Fetches in the Linux Kernel

Pengfei Wang, National University of Defense Technology;

Jens Krinke, University College London;

Kai Lu and Gen Li, National University of Defense Technology;

Steve Dodier-Lazaro, University College London

26th USENIX Security Symposium,

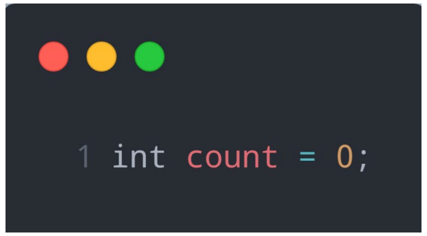
August 16–18, 2017 • Vancouver, BC, Canada

J.T. ZHANG 2019.06, NUAA

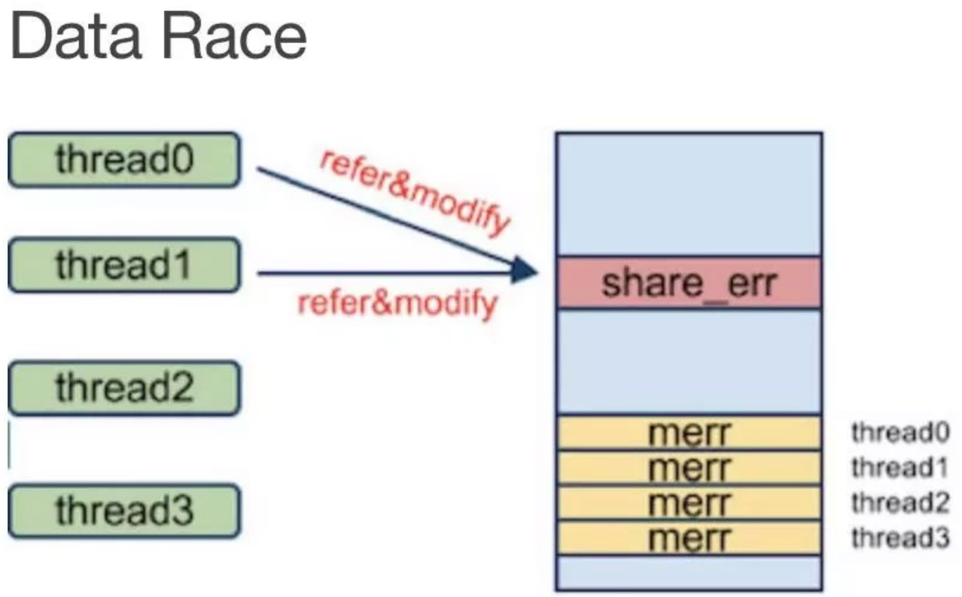
Multicore – Concurrent Programming

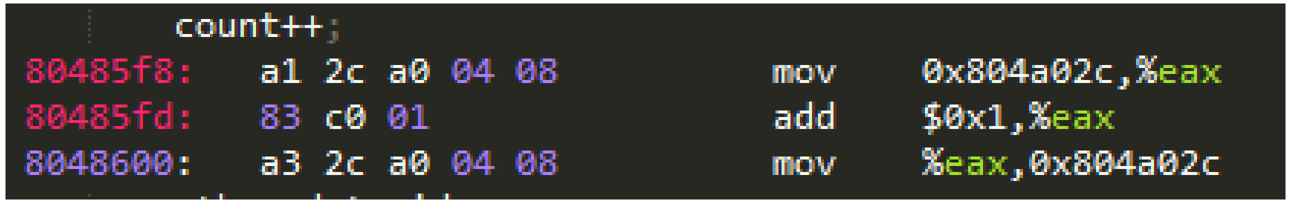
Data Race

- Shared memory
- At least one of the accesses is a write
- Atomicity-violation

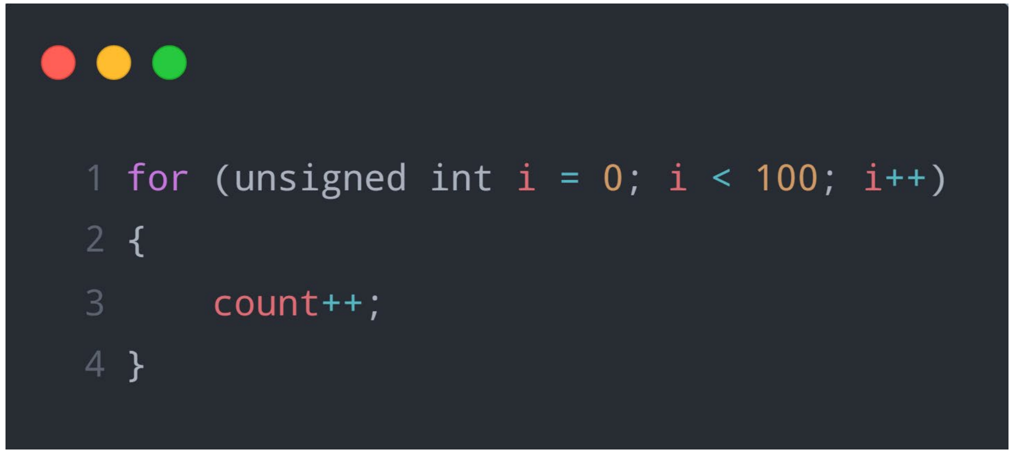


```
int count = 0;
```





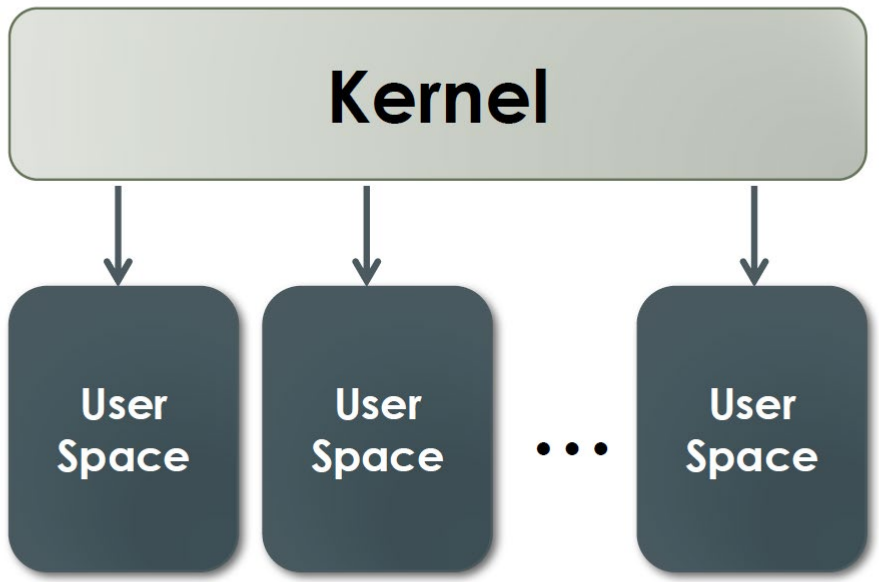
```
        count++;
80485f8:    a1 2c a0 04 08              mov     0x804a02c,%eax
80485fd:    83 c0 01                    add     $0x1,%eax
8048600:    a3 2c a0 04 08              mov     %eax,0x804a02c
```



```
for (unsigned int i = 0; i < 100; i++)
{
    count++;
}
```

Kernel/User Space Protection

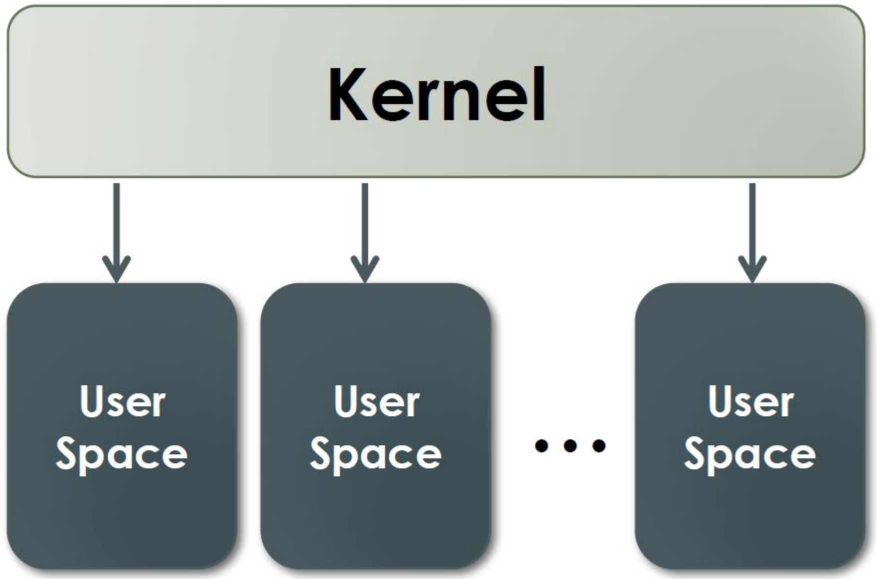
- Kernel space
- User space

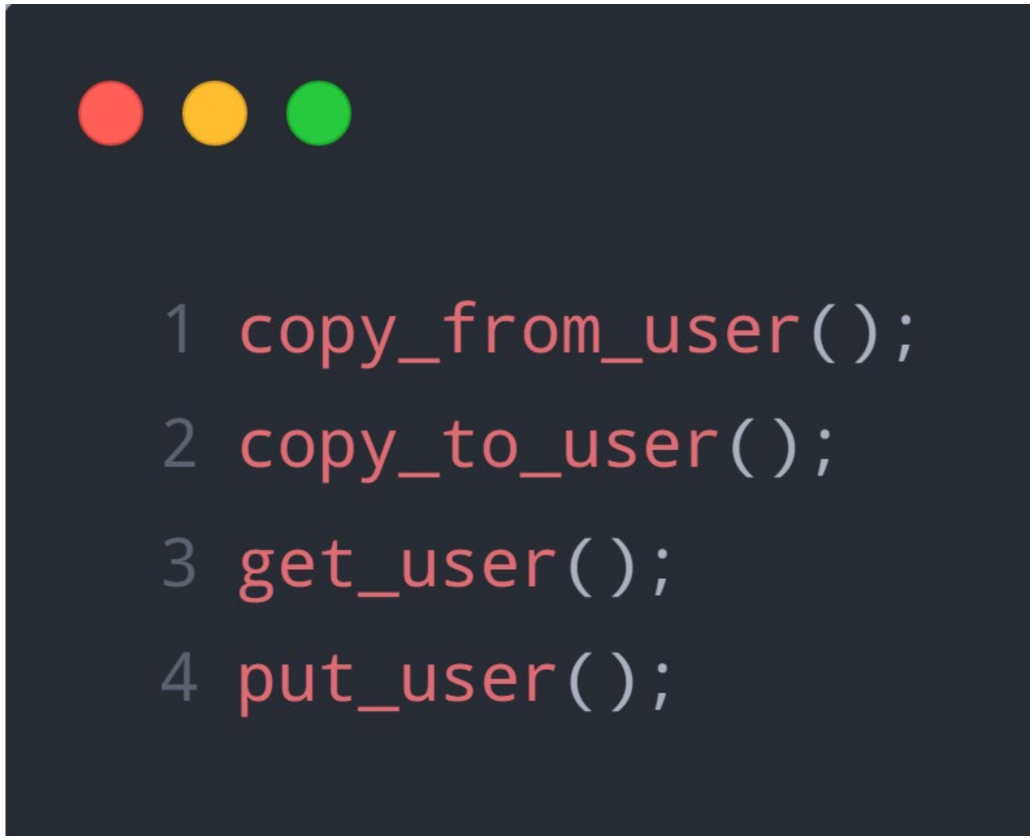


- Each user process has its own virtual memory space
- User space is isolated - Virtual address → Physical address
- Only the kernel can access all user spaces

Kernel/User Space Data Exchange

- Transfer Functions





```
copy_from_user();
copy_to_user();
get_user();
put_user();
```

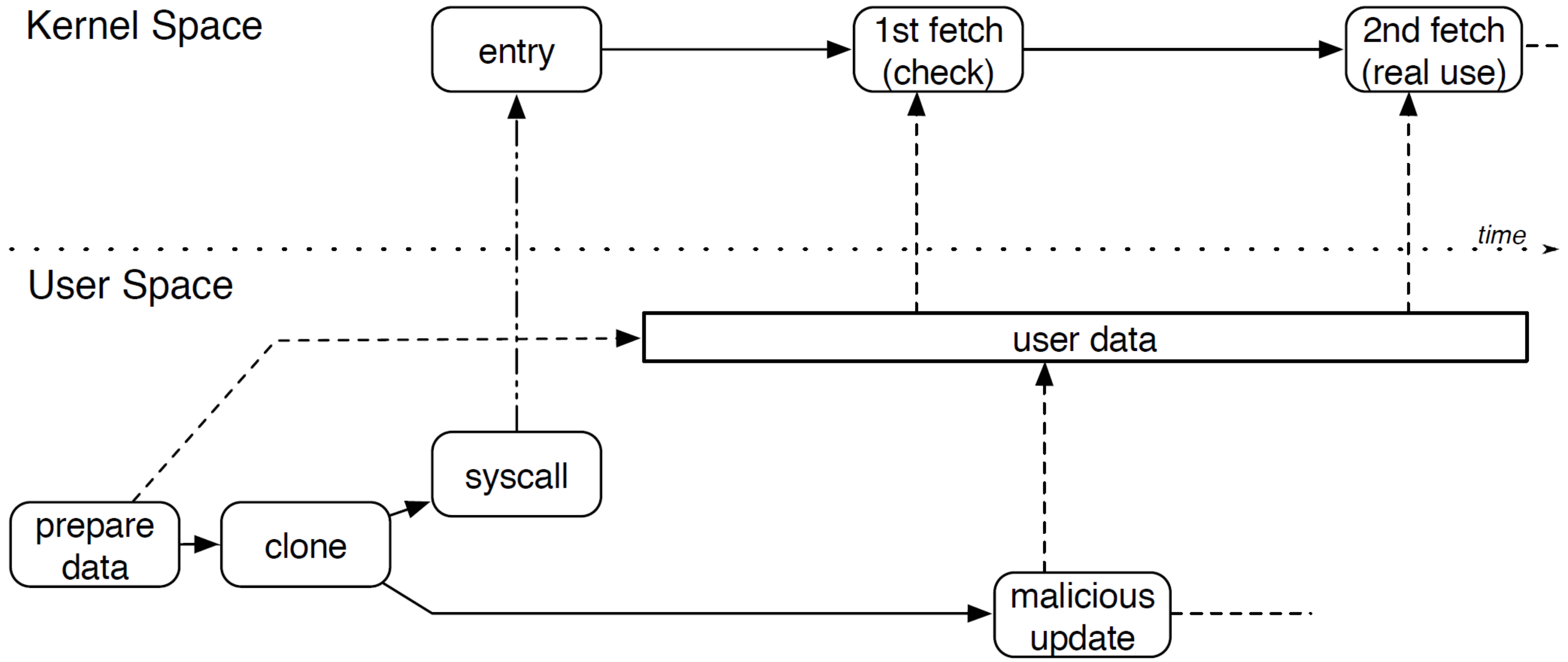
Memory Access in Drivers

- Kernel components
- Bridge between software & hardware

In Linux : Device → File

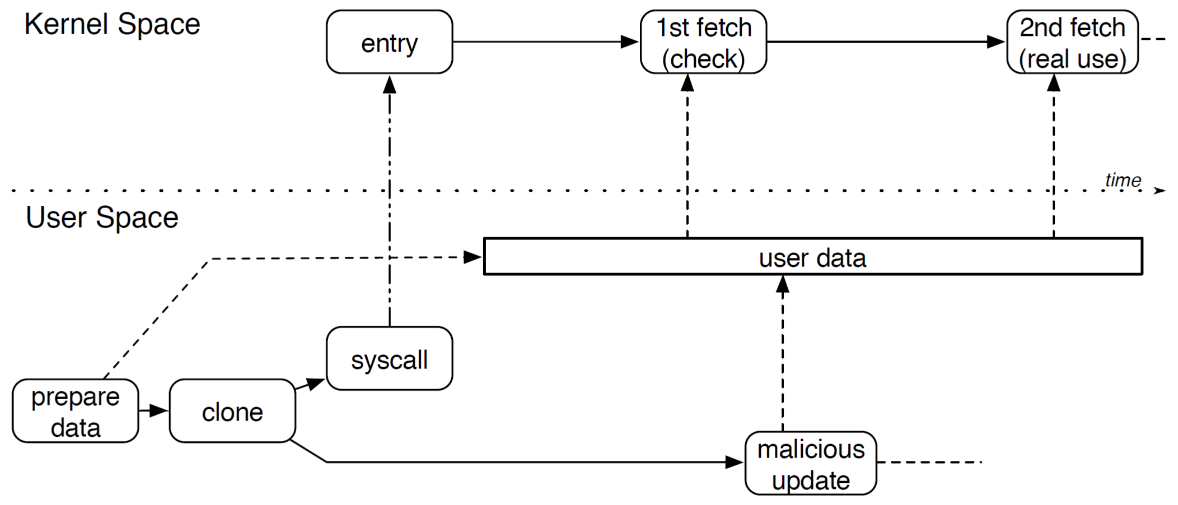
- File I/O system call
- Copy data from kernel to user space – `read();`
- Copy data from user space to kernel – `write();`

What is Double-Fetch?



- Kernel fetch data from the same memory location twice
- During two fetch, the memory is modified

Double-Fetch Grade



Benign double fetch

- Protection schemes
- Double-fetched value is not used

Harmful double fetch

- Could cause failures under specific conditions

Double-fetch vulnerability

- Privilege escalation
- Information leakage
- Kernel crash

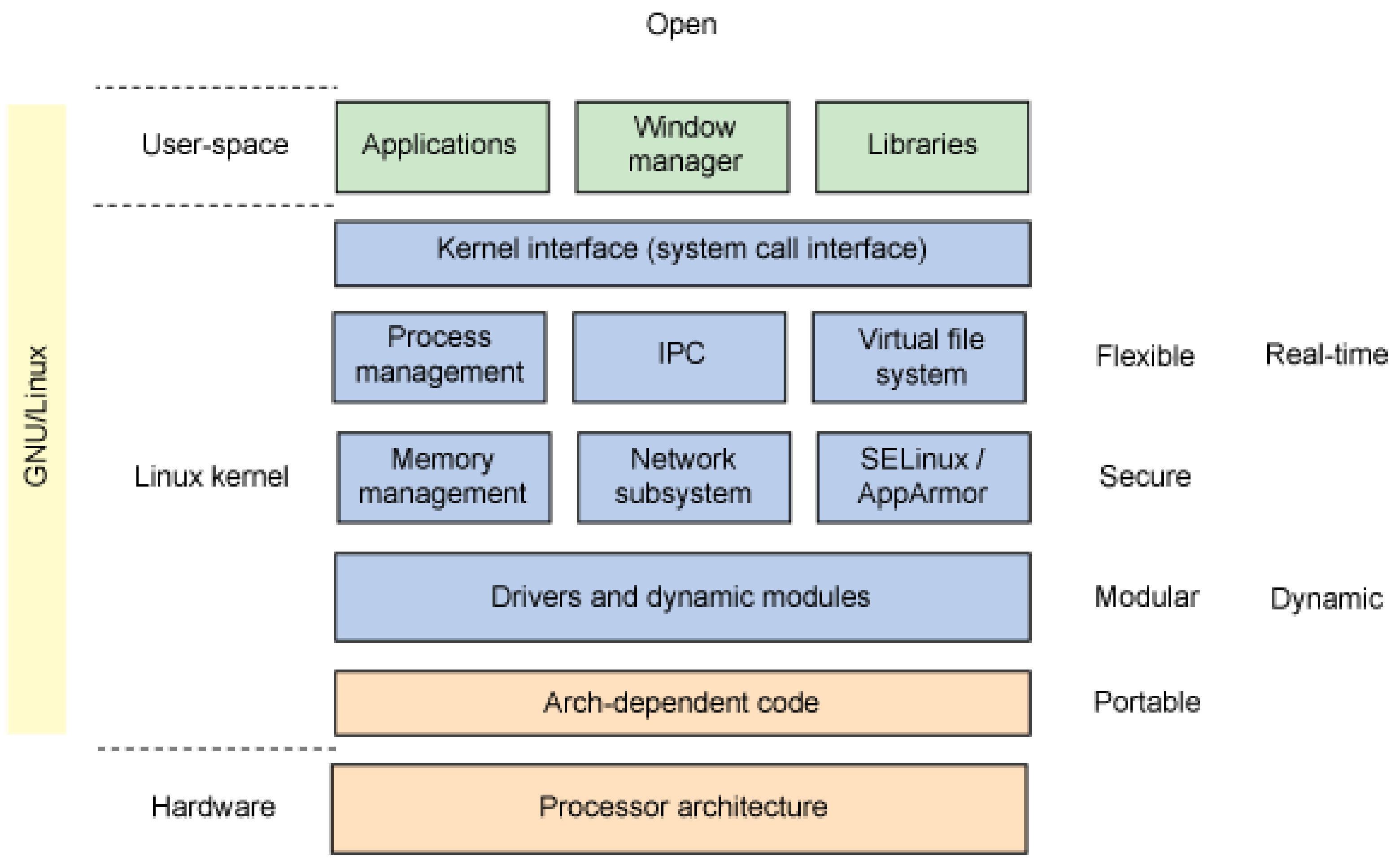
Related Works

Dynamic Analysis

- Devices?
- Architectures?
- Overhead?

Static Analysis

- Source code
- Missing context – false reports



Pattern-based static analysis

Basic Pattern Matching Analysis

- Identify multiple invocations of *transfer function*
- Analyze manually & categorize

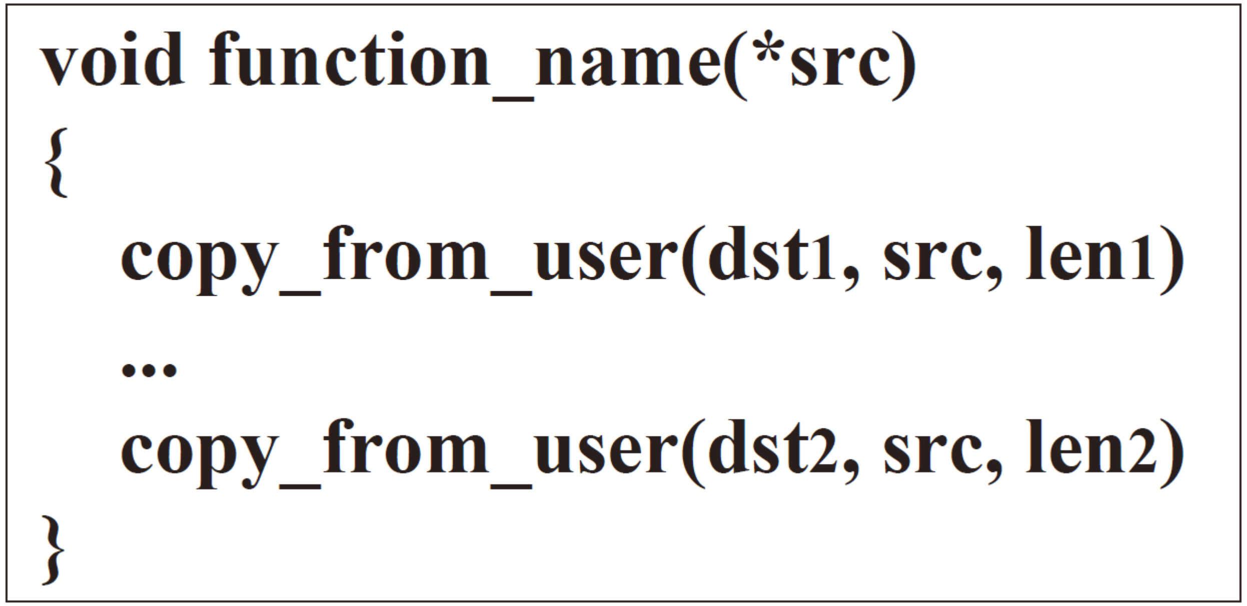
Refined Double-Fetch Bug Detection

- Find double-fetch vulnerabilities

Basic Pattern Matching Analysis

Basic Pattern: In one function, fetching data from user space more than twice

- Matching all **get_user()** or **copy_from_user()** variants
- Target address and copy size can be different
- Source address must be the same



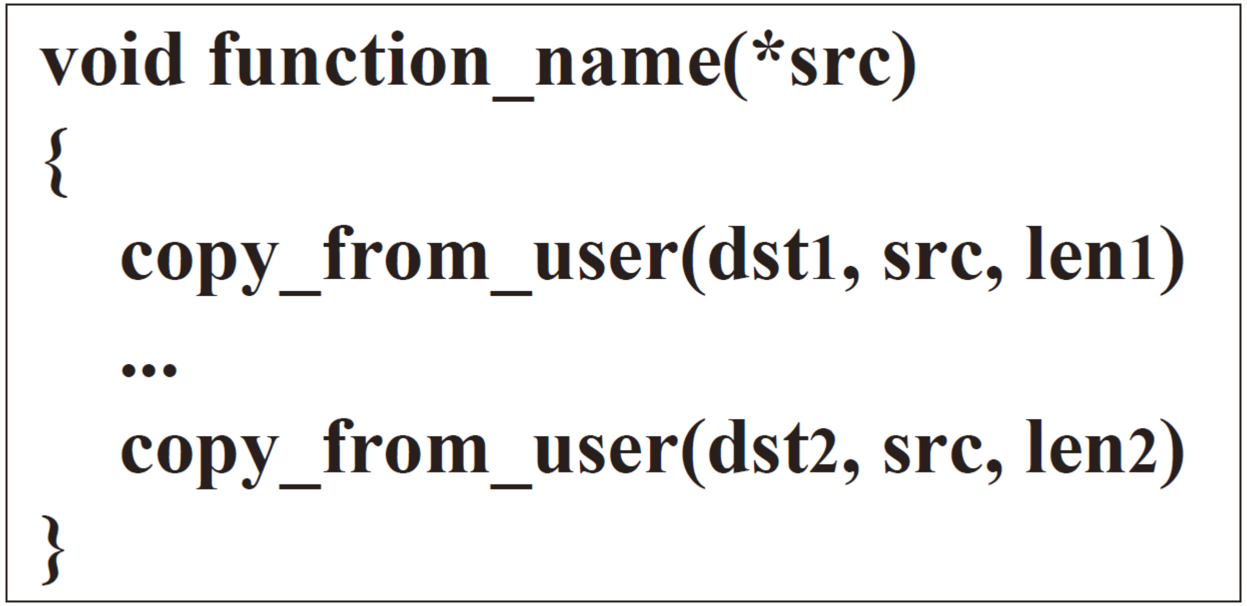
```
void function_name(*src)
{
    copy_from_user(dst1, src, len1)
    ...
    copy_from_user(dst2, src, len2)
}
```

Basic Pattern Matching Analysis

False Positives

- Fetching different element of the same structure
- Adding offset

Automatically removed



```
void function_name(*src)
{
    copy_from_user(dst1, src, len1)
    ...
    copy_from_user(dst2, src, len2)
}
```

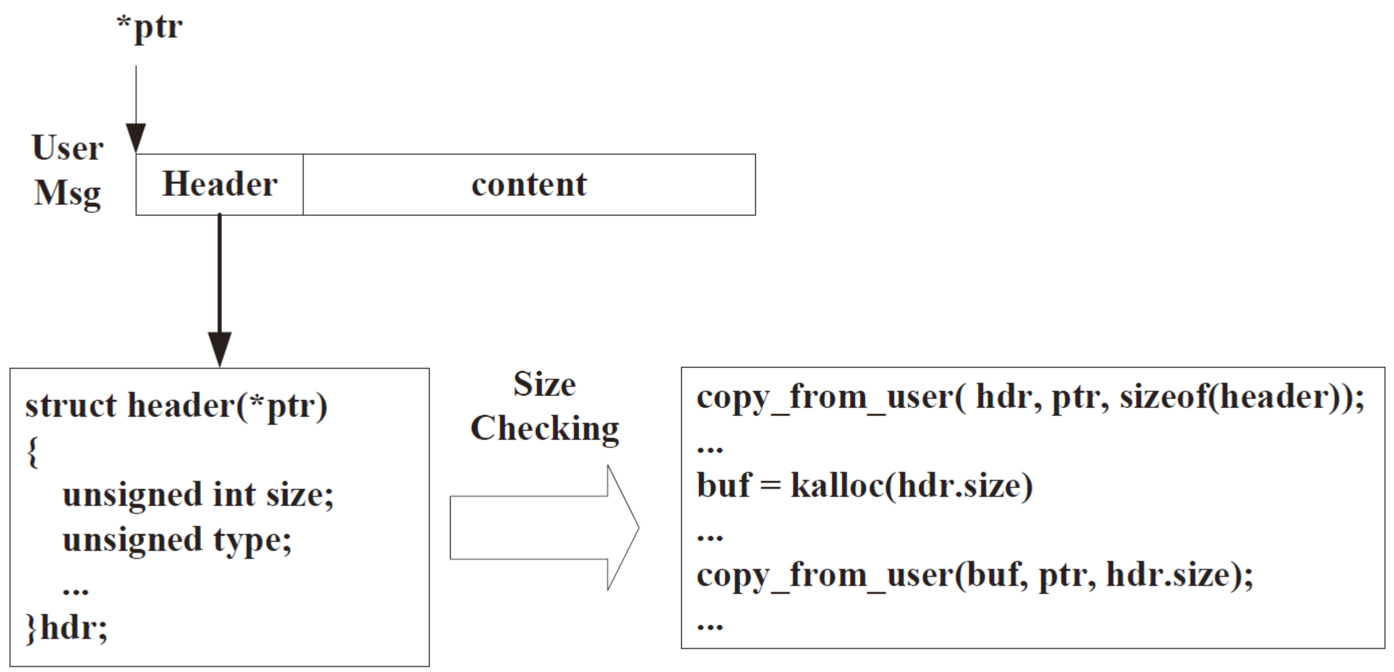
Double Fetch Categorization

- Size checking
- Type selection
- Shallow copy

Double Fetch Categorization – Size Checking

Message Data

- Header
- Body (vary in length)



```
struct header(*ptr)
{
   unsigned int size;
   unsigned type;
   ...
}hdr;
```

```
copy_from_user( hdr, ptr, sizeof(header));
...
buf = kalloc(hdr.size)
...
copy_from_user(buf, ptr, hdr.size);
...
```

First fetch

- Header only
- Get message size, check validity, allocate buffer

Second fetch

- Complete message copied to local buffer

Double Fetch Categorization – Size Checking

Adaptec RAID controller driver - commctrl.c (CVE-2016-6480)

```
static int ioctl_send_fib(struct aac_dev* dev,
                          void __user *arg)
{
  struct hw_fib * kfib;
...
  if (copy_from_user((void *)kfib, arg, sizeof(...))) {
    aac_fib_free(fibptr);
    return -EFAULT;
  }
...
  size = le16_to_cpu(kfib->header.Size) + sizeof(...);
  if (size < le16_to_cpu(kfib->header.SenderSize))
    size = le16_to_cpu(kfib->header.SenderSize);
  if (size > dev->max_fib_size) {
...
    kfib = pci_alloc_consistent(dev->pdev, size, &daddr);
...
  }

  if (copy_from_user(kfib, arg, size)) {
    retval = -EFAULT;
    goto cleanup;
  }

  if (kfib->header.Command == cpu_to_le16(...)) {
...
  } else {
    retval =
      aac_fib_send(le16_to_cpu(kfib->header.Command),...
             le16_to_cpu(kfib->header.Size) , FsaNormal,
             1, 1, NULL, NULL);
...
  }
...
}
```

Double Fetch Categorization – Size Checking

Results

- 30 occurrences
- 22 in drivers
- 4 of 22 in drivers occurrences cause double-fetch bugs

Double Fetch Categorization – Type Selection

Message Data

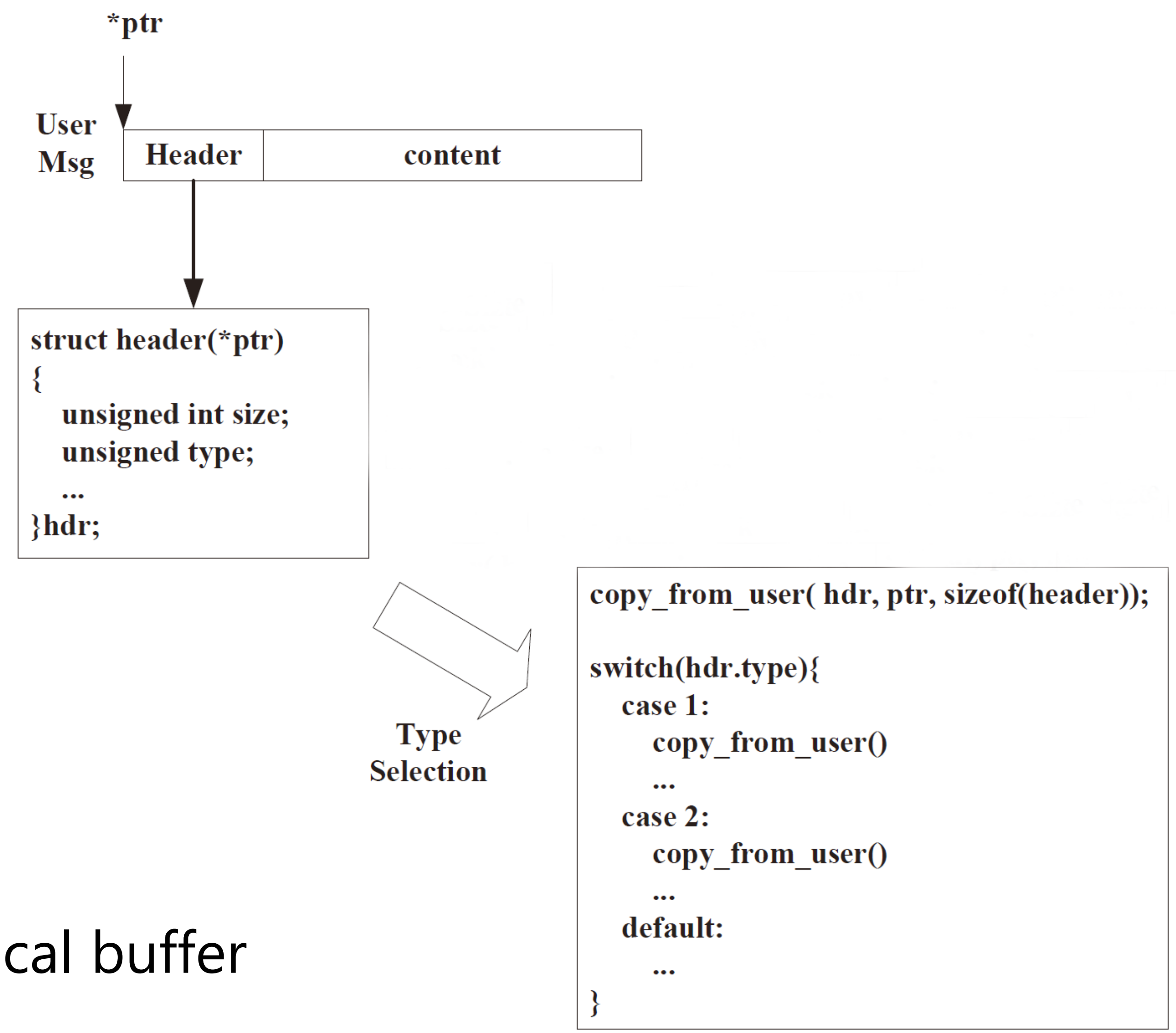
- Header
- Body

First fetch

- Header only
- Recognize message type

Second fetch

- Complete message copied to local buffer



```
struct header(*ptr)
{
   unsigned int size;
   unsigned type;
   ...
}hdr;
```

```
copy_from_user( hdr, ptr, sizeof(header));

switch(hdr.type){
   case 1:
      copy_from_user()
      ...
   case 2:
      copy_from_user()
      ...
   default:
      ...
}
```

Double Fetch Categorization – Type Selection

Network driver - xgb3_main.c

No vulnerability

```
static int cxgb_extension_ioctl(struct net_device *dev,
                                void __user *useraddr)
{
...
  u32 cmd;
...
  if (copy_from_user(&cmd, useraddr, sizeof(cmd)))
    return -EFAULT;

  switch (cmd) {
```

```
  case CHELSIO_SET_QSET_PARAMS:{
...
    struct ch_qset_params t;
...
    if (copy_from_user(&t, useraddr, sizeof(t))
      return -EFAULT;
    if (t.qset_idx >= SGE_QSETS)
      return -EINVAL;
...
    break;
  }
```

```
  case CHELSIO_SET_QSET_NUM:{
    struct ch_reg edata;
...
    if (copy_from_user(&edata, useraddr, sizeof(edata)))
      return -EFAULT;
    if (edata.val < 1 ||
        (edata.val > 1 && !(...)))
      return -EINVAL;
...
    break;
  }
```

Double Fetch Categorization – Type Selection

Results

- 11 occurrences
- 9 in drivers
- None of 11 occurrences cause double-fetch bugs

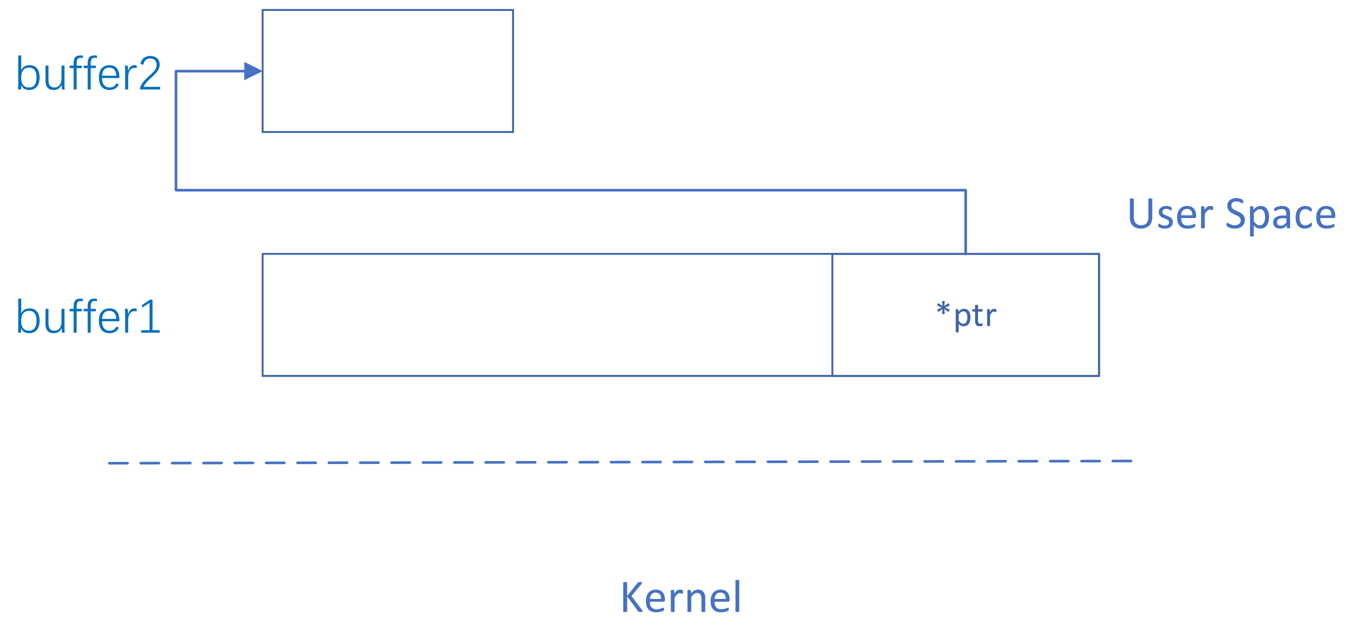
Double Fetch Categorization – Shallow Copy

Copy buffer1 → Shallow copy

Copy buffer2 → Deep copy

Results

- 31 occurrences
- 19 in drivers



Double Fetch Categorization – Shallow Copy

IBM S/390 SCLP console driver - sclp_ctl.c (CVE-2016-6130)

```c
static int sclp_ctl_ioctl_sccb(void __user *user_area)
{
  struct sclp_ctl_sccb ctl_sccb;
  struct sccb_header *sccb;
  int rc;

  if (copy_from_user(&ctl_sccb, user_area,
                     sizeof(ctl_sccb)))
    return -EFAULT;
...
```

```c
  sccb = (void *) get_zeroed_page(GFP_KERNEL | GFP_DMA);
  if (!sccb)
    return -ENOMEM;
  if (copy_from_user(sccb, u64_to_uptr(ctl_sccb.sccb),
                     sizeof(*sccb))) {
    rc = -EFAULT;
    goto out_free;
  }
  if (sccb->length > PAGE_SIZE || sccb->length < 8)
    return -EINVAL;
  if (copy_from_user(sccb, u64_to_uptr(ctl_sccb.sccb),
                     sccb->length)) {
    rc = -EFAULT;
    goto out_free;
  }
...
  if (copy_to_user(u64_to_uptr(ctl_sccb.sccb), sccb,
                   sccb->length))
    rc = -EFAULT;
...
}
```

Refined Double-Fetch Bug Detection

- Improved analysis
- Specifically identify double-fetch bugs

Rule 0 (basic rule) – two reads fetch data from the same location

- More rules…
- Coccinelle applied these rules one by one

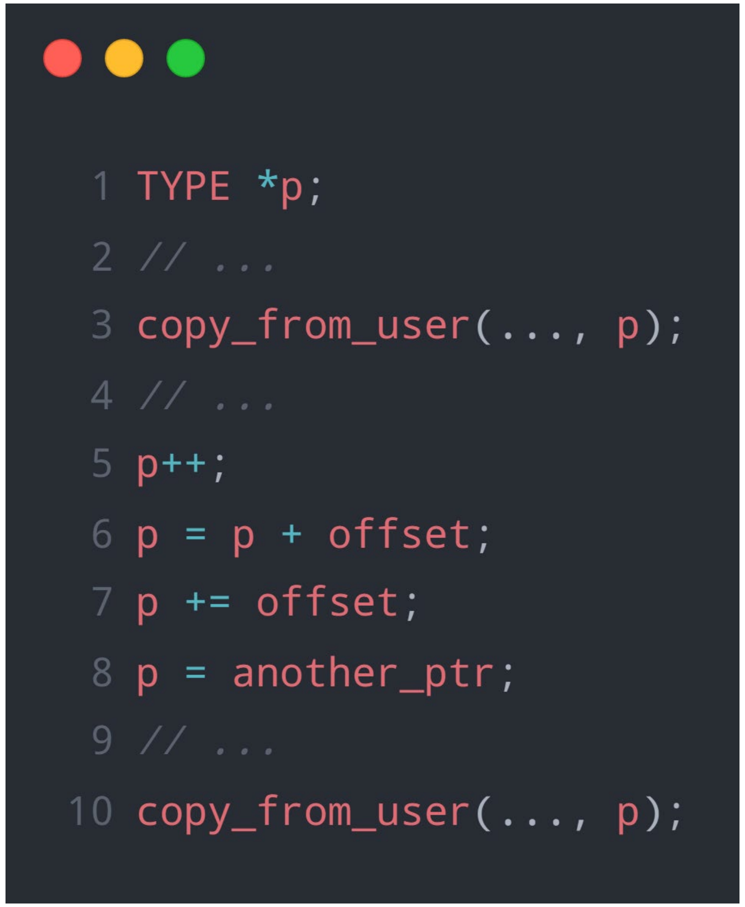
Refined Double-Fetch Bug Detection

Rule 1 – No pointer change

Keeping the user pointer unchanged between two fetches

- Self-increment (++/--)
- Adding an offset
- Assignment of another value

↓ False positives



```
TYPE *p;
// ...
copy_from_user(..., p);
// ...
p++;
p = p + offset;
p += offset;
p = another_ptr;
// ...
copy_from_user(..., p);
```

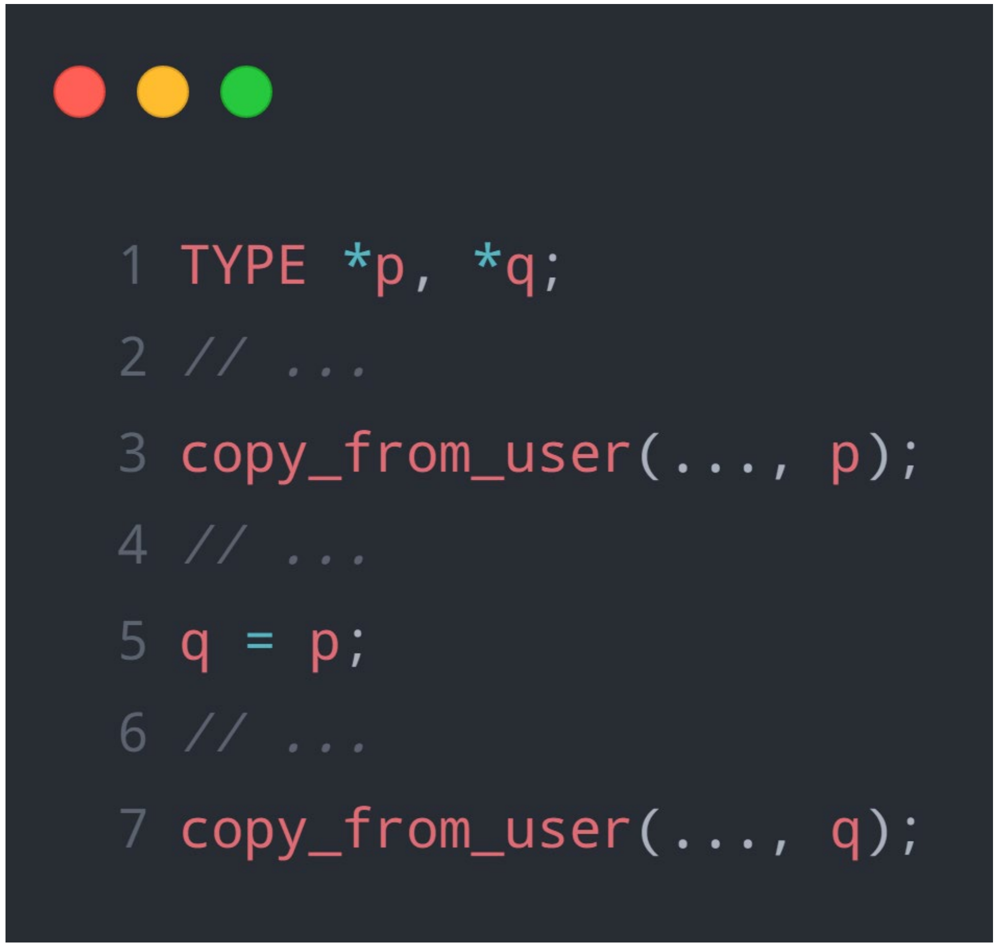
Refined Double-Fetch Bug Detection

Rule 2 – Pointer Aliasing

Source pointer is assigned to another pointer

- For convenience

↓ False negatives



```
TYPE *p, *q;
// ...
copy_from_user(..., p);
// ...
q = p;
// ...
copy_from_user(..., q);
```

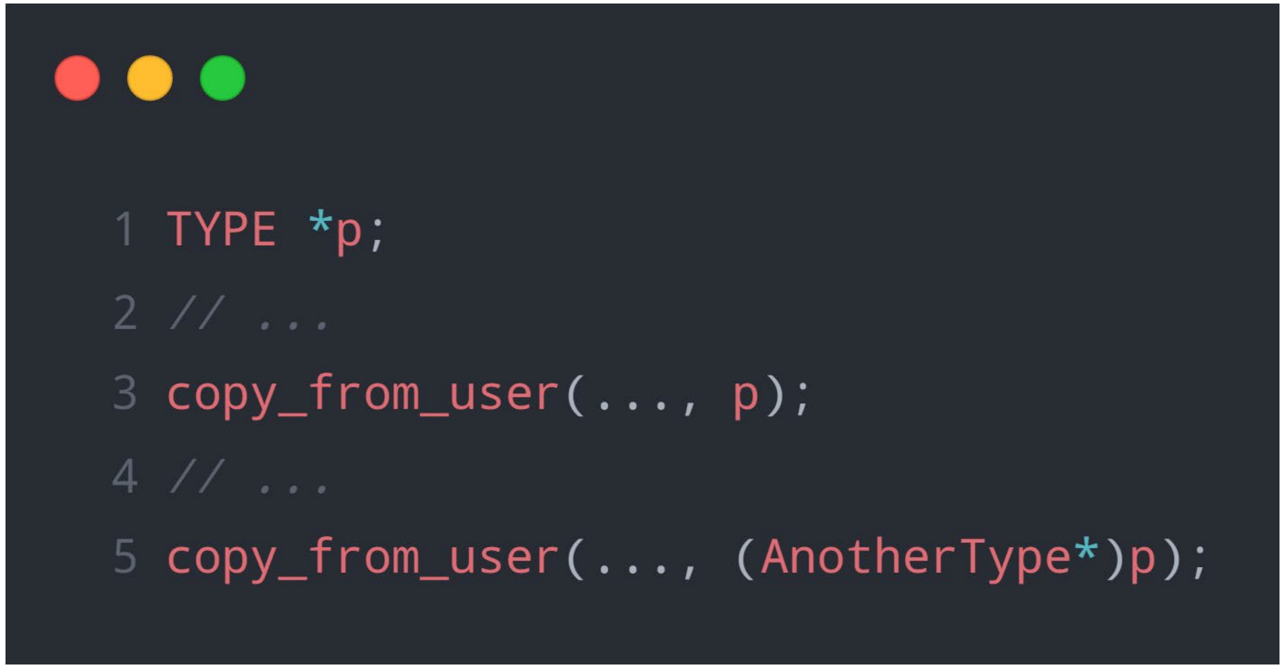
Refined Double-Fetch Bug Detection

Rule 3 – Explicit type conversion

Causing the same memory region to be manipulated by two types of pointers

- First fetch – (header pointer)
- Second fetch – message pointer

↓ False negatives



```
TYPE *p;
// ...
copy_from_user(..., p);
// ...
copy_from_user(..., (AnotherType*)p);
```

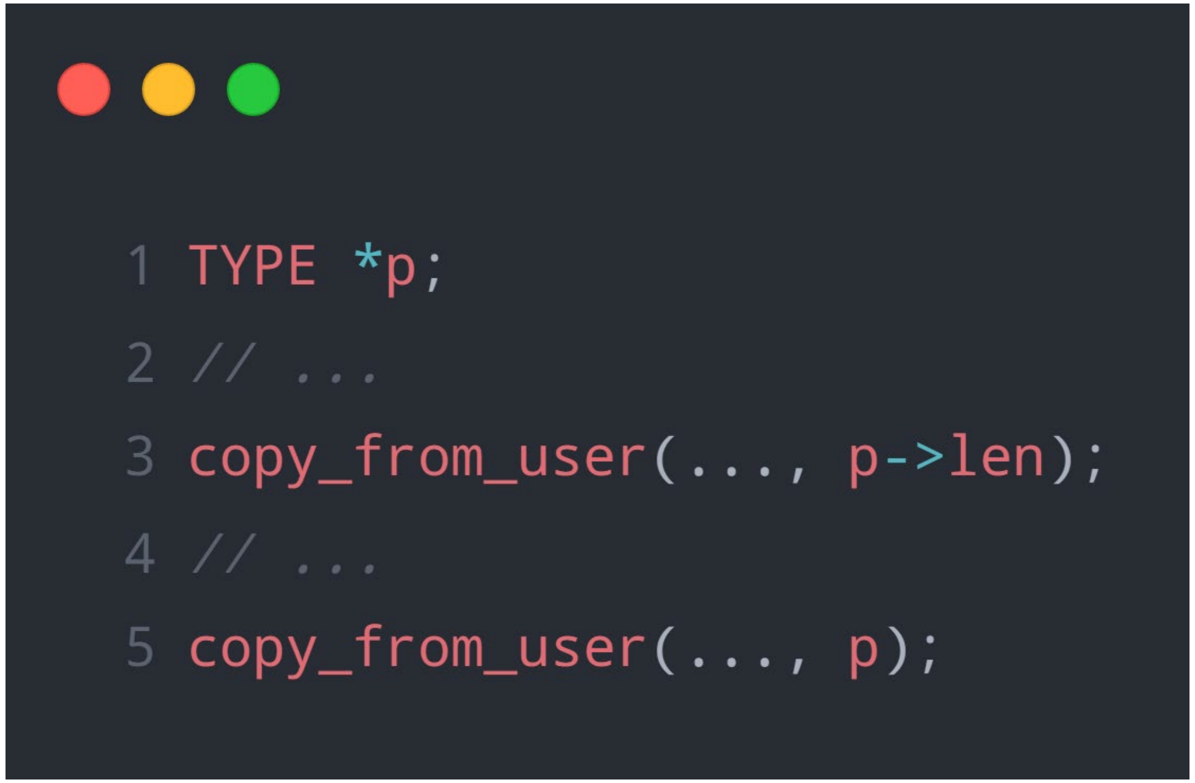
Refined Double-Fetch Bug Detection

Rule 4 – Combination of element fetch and pointer fetch

Pointer is both used to fetch the **whole data structure** as well as only **a part**

- Not using the same pointer
- But cover the same value

↓ False negatives



```
TYPE *p;
// ...
copy_from_user(..., p->len);
// ...
copy_from_user(..., p);
```

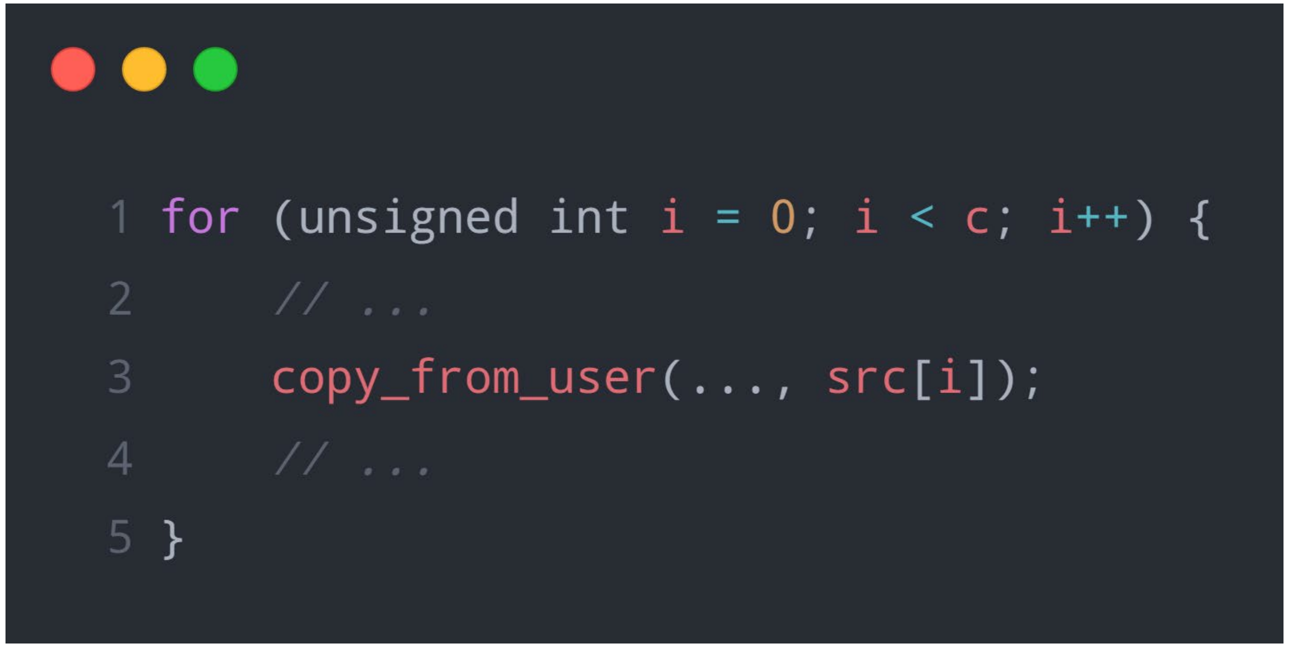
Refined Double-Fetch Bug Detection

Rule 5 – Loop involvement

One transfer function call in a loop will be reported as two calls

- Should be removed
- Cross-iteration transfer function call

↓ False positives

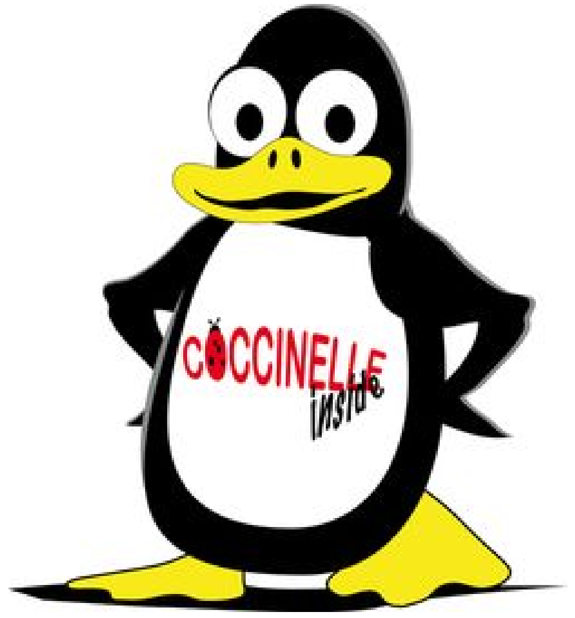


```
for (unsigned int i = 0; i < c; i++) {
    // ...
    copy_from_user(..., src[i]);
    // ...
}
```

Tool - Coccinelle

An open-source utility for matching and transforming the source code of programs written in the C programming language using a "semantic patch" syntax based on the patch syntax.

Semantic Patch Language (SmPL)



Evaluation

- Linux 4.5
- Android 6.0.1 (based on Linux 3.18)
- FreeBSD (master branch)

- Drivers are the hard-hit area
- Benign case could turn into a vulnerability with code update (CVE-2016-5728)

Double-Fetch BUG Prevention

- Don’t copy the header twice (only copy body in the second fetch)
- Use the same value
  - Only use the data from one of the fetches
- Overwrite data
  - Overwrite the header with the first fetch (widely adopted in FreeBSD)
- Compare data
  - Compare before using, abort the operation if the data is not the same
- Synchronize fetches
  - Guarantee the atomicity
  - Performance penalties

Limitations

- Preprocessed code
- Compiled code

Double-fetch can occur in:

- Macros
- Compiler optimization (in compiled binary, not in source code)
- Not labeled as *volatile*
    - Turn the memory access from single to multiple at the binary level

Thanks.

Q&A ?