Lecture 11
Exploit Demo 1

Computer and Network Security
December 19, 2022

Computer Science and Engineering Department

Outline

CVE List▾ CNAs▾ WGs▾ Board▾ About▾ News & Blog▾

NVD Go to for: CVSS Scores CPE Info

Search CVE List | Downloads | Data Feeds | Update a CVE Record | Request CVE IDs

TOTAL CVE Records: 189773

NOTICE: Transition to the all-new CVE website at WWW.CVE.ORG is underway and will last up to one year. (details)

NOTICE: Changes coming to CVE Record Format JSON and CVE List Content Downloads in 2022.

HOME > CVE > CVE-2022-0847

Printer-Friendly View

CVE-ID	
CVE-2022-0847	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

A flaw was found in the way the "flags" member of the new pipe buffer structure was lacking proper initialization in copy_page_to_iter_pipe and push_pipe functions in the Linux kernel and could thus contain stale values. An unprivileged local user could use this flaw to write to pages in the page cache backed by read only files and as such escalate their privileges on the system.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

CVE List▾ CNAs▾ WGs▾ Board▾ About▾ News & Blog▾

NVD Go to for: CVSS Scores CPE Info

Search CVE List | Downloads | Data Feeds | Update a CVE Record | Request CVE IDs

TOTAL CVE Records: 189773

NOTICE: Transition to the all-new CVE website at WWW.CVE.ORG is underway and will last up to one year. (details)

NOTICE: Changes coming to CVE Record Format JSON and CVE List Content Downloads in 2022.

HOME > CVE > CVE-2022-0847

2. What

1. Where

Printer-Friendly View

CVE-ID	
CVE-2022-0847	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
A flaw was found in the way the "flags" member of the new pipe buffer structure was lacking proper initialization in copy_page_to_iter_pipe and push_pipe functions in the Linux kernel and could thus contain stale values. An unprivileged local user could use this flaw to write to pages in the page cache backed by read only files and as such escalate their privileges on the system.	
References	
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.	

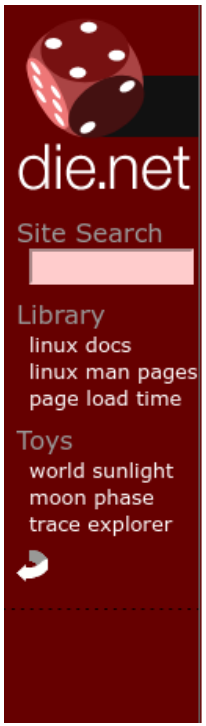
3. Consequence

Outline

Pipes - high level

```
1 test@test:~$ ls -l | grep file
```

- The stdout of `ls` is "connected" to the stdin of `grep`



die.net

Site Search

Library
linux docs
linux man pages
page load time

Toys
world sunlight
moon phase
trace explorer

pipe(2) - Linux man page

Name

pipe, pipe2 - create pipe

Synopsis

```
#include <unistd.h>
int pipe(int pipefd[2]);
#define _GNU_SOURCE             /* See feature_test_macros(7) */#include
<fcntl.h>              /* Obtain O_* constant definitions */#include <unistd.h>
int pipe2(int pipefd[2], int flags);
```

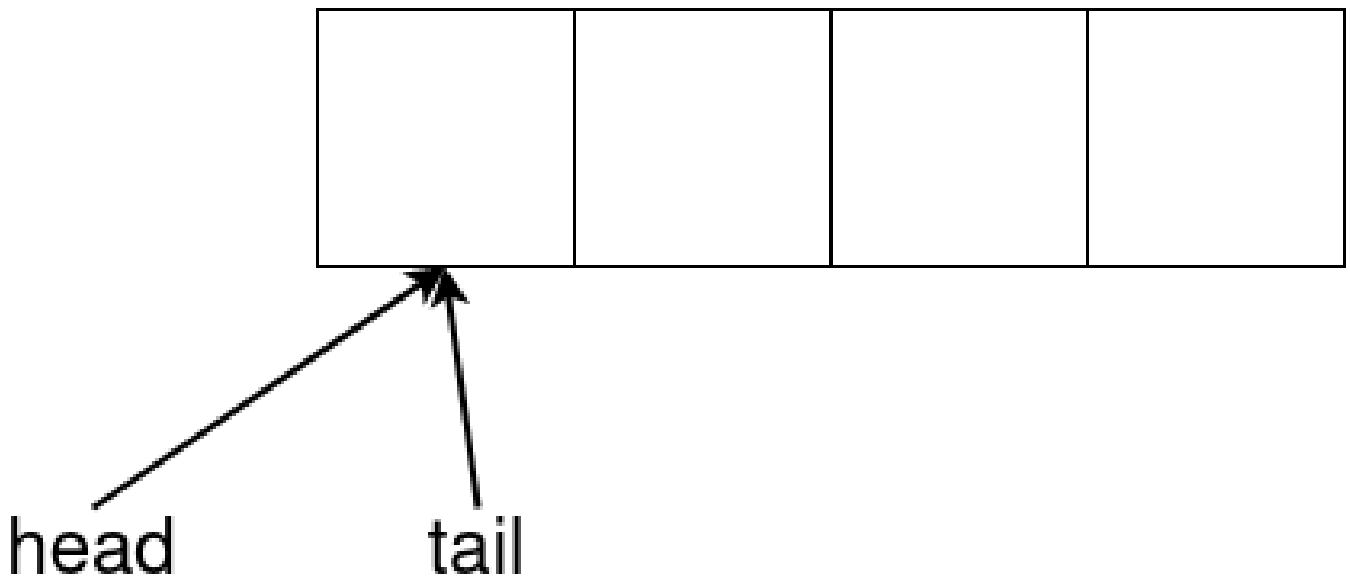
Description

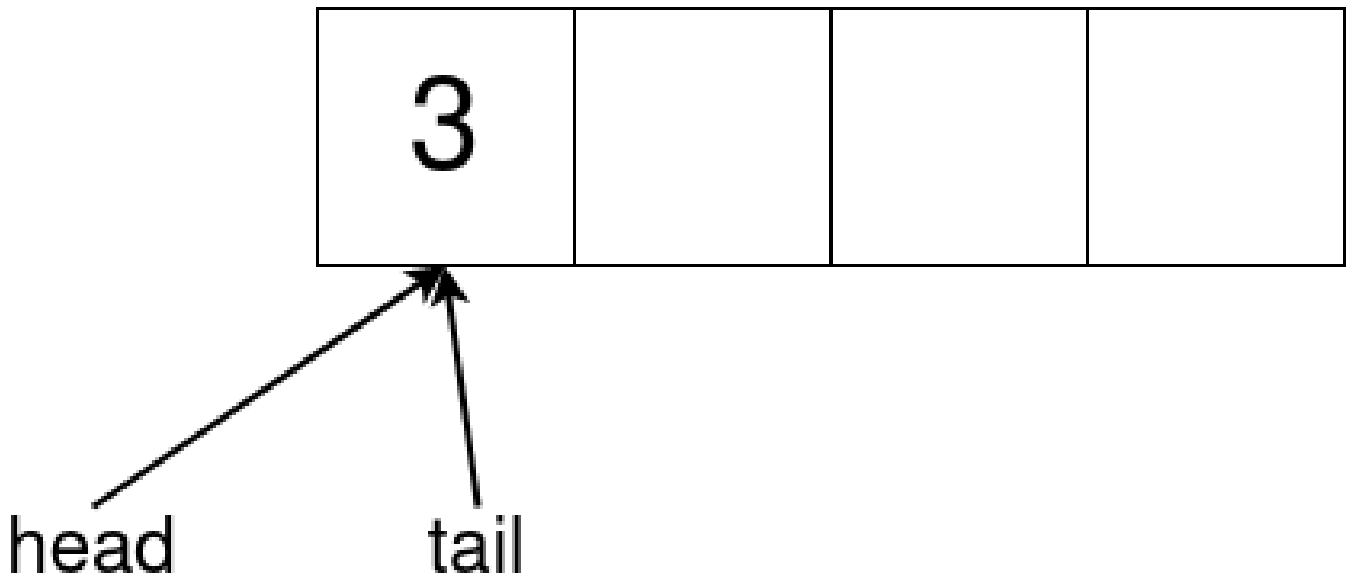
pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array *pipefd* is used to return two file descriptors referring to the ends of the pipe. *pipefd[0]* refers to the read end of the pipe. *pipefd[1]* refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe. For further details, see ***pipe****(7)*.

Pipes - kernel level

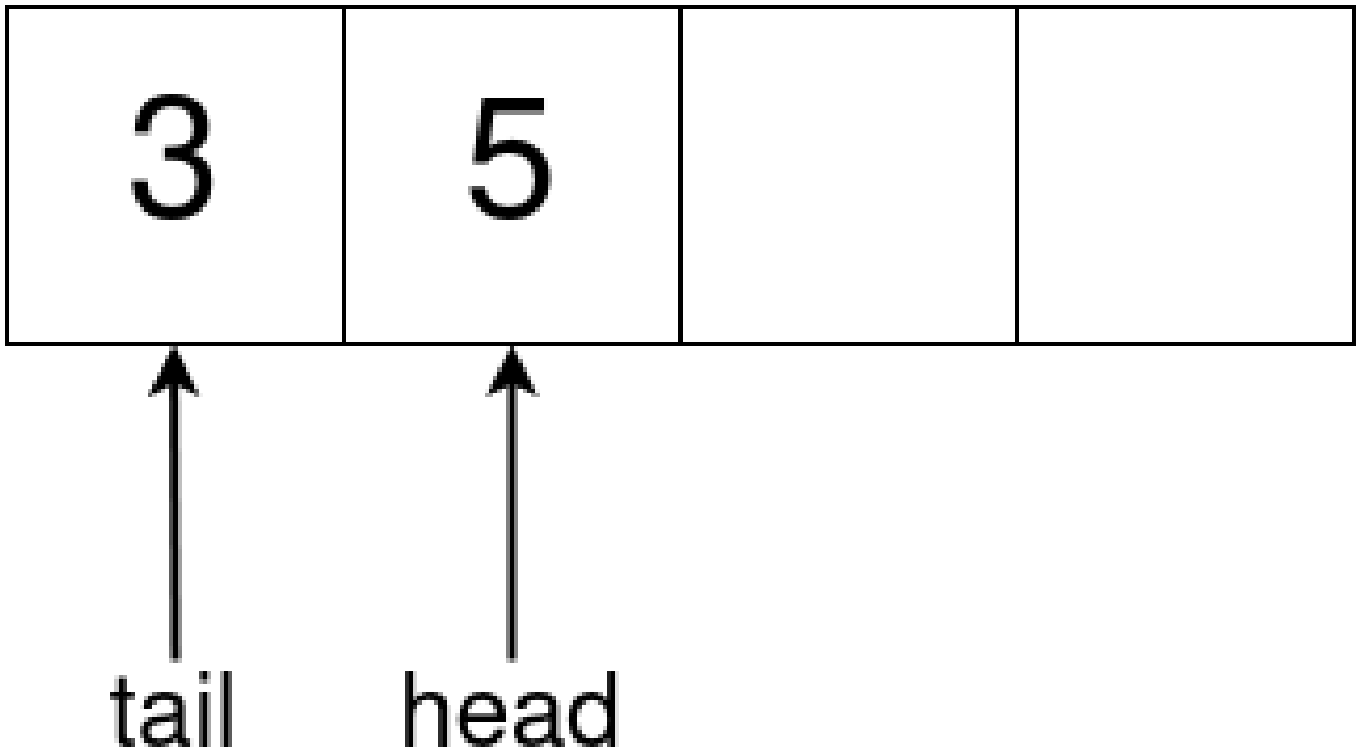
```
1  struct pipe_inode_info {
2          struct mutex mutex;
3          wait_queue_head_t rd_wait, wr_wait;
4          unsigned int head;
5          unsigned int tail;
6          unsigned int max_usage;
7          unsigned int ring_size;
8          ...
9          struct pipe_buffer *bufs;
10         struct user_struct *user;
11 #ifdef CONFIG_WATCH_QUEUE
12         struct watch_queue *watch_queue;
13 #endif
14 };
15
16 struct pipe_buffer {
17         struct page *page;
18         unsigned int offset, len;
19         const struct pipe_buf_operations *ops;
20         unsigned int flags;
21         unsigned long private;
22 };
```

Intermission: array queue

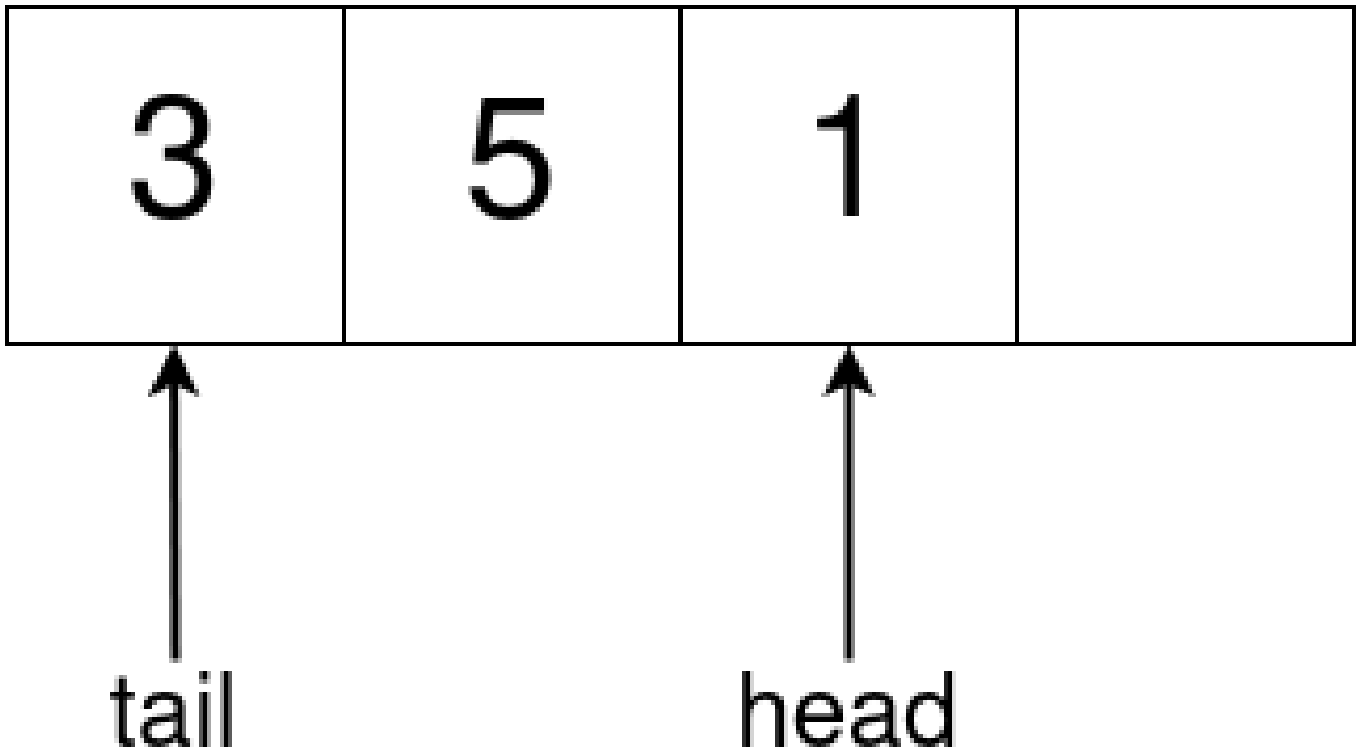




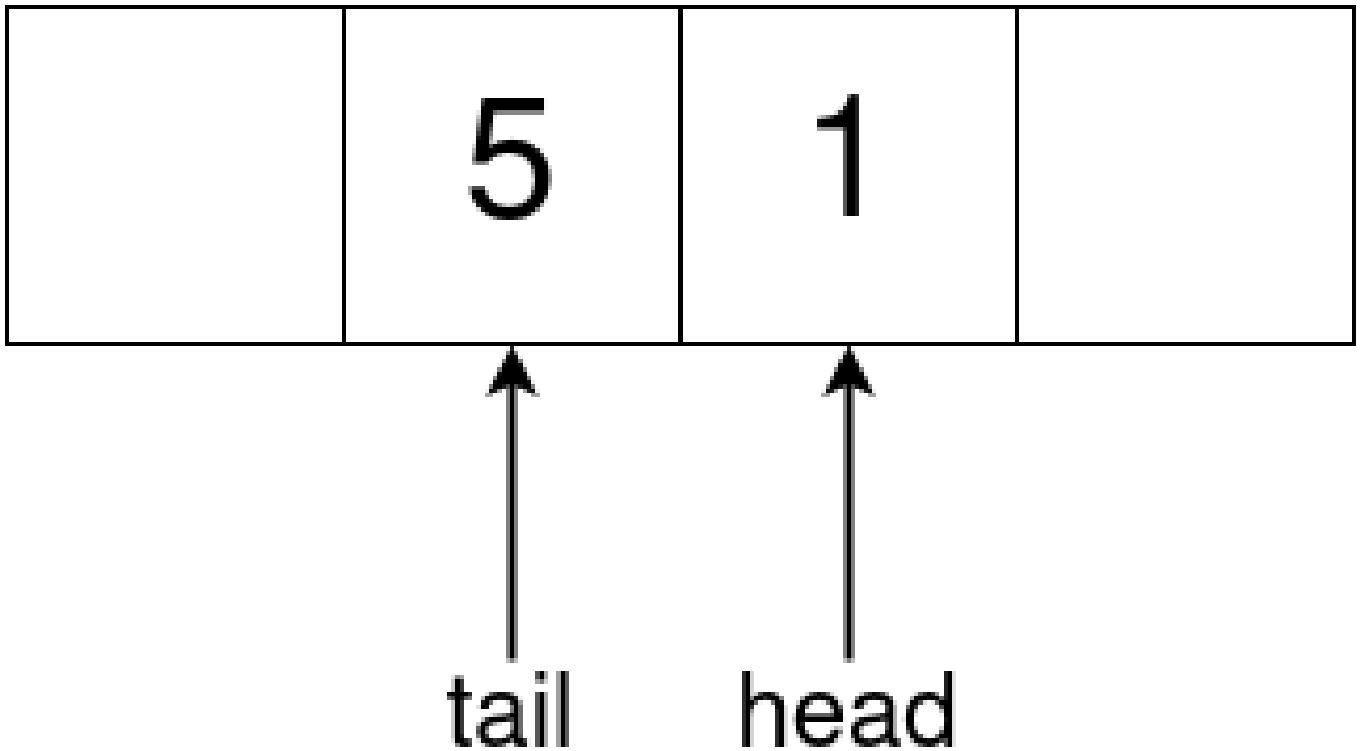
- enqueue(3)



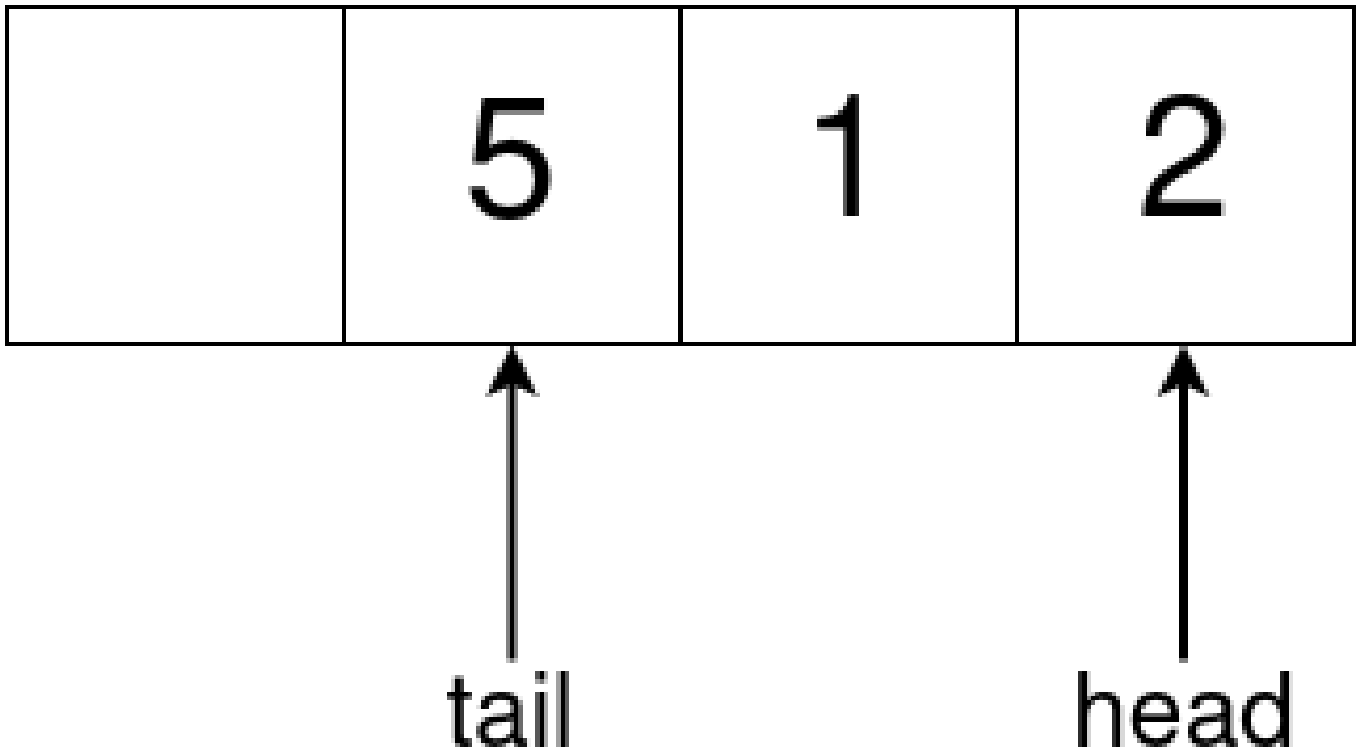
- enqueue(5)



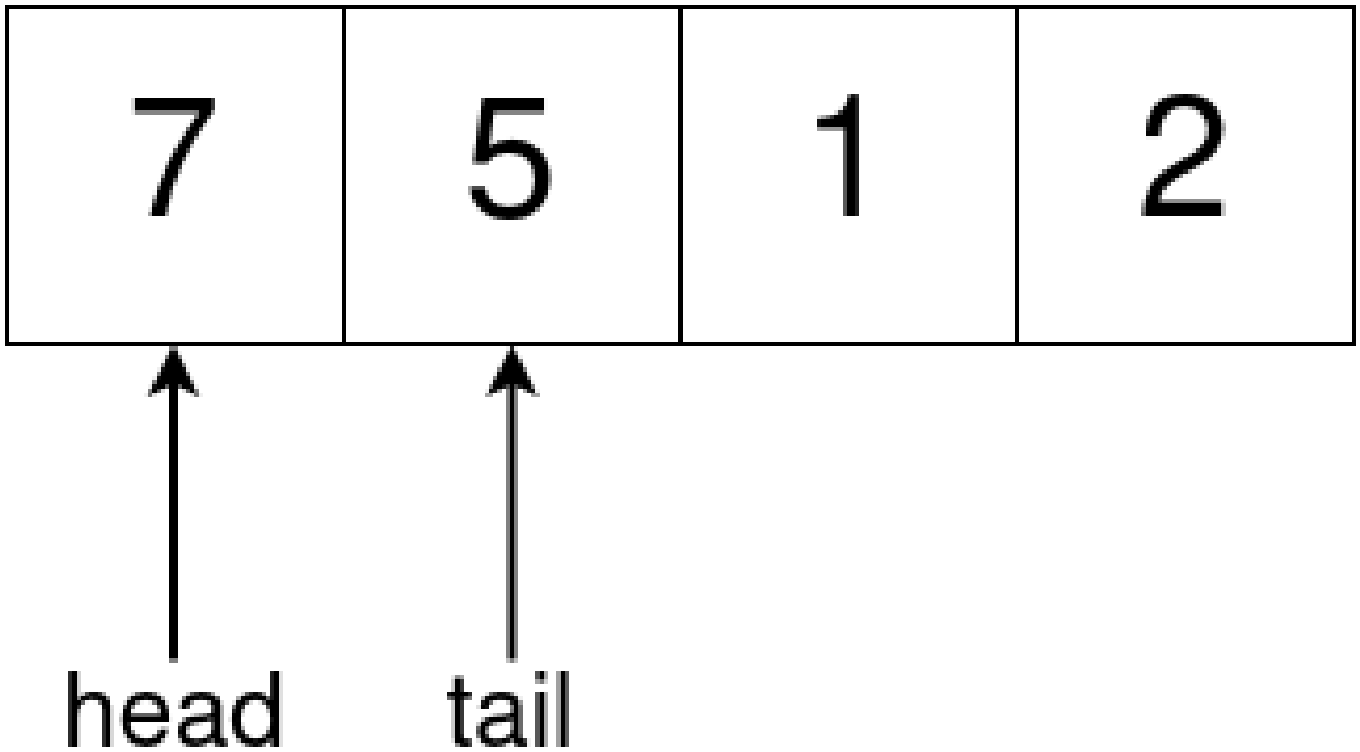
- enqueue(1)



- dequeue() = 3

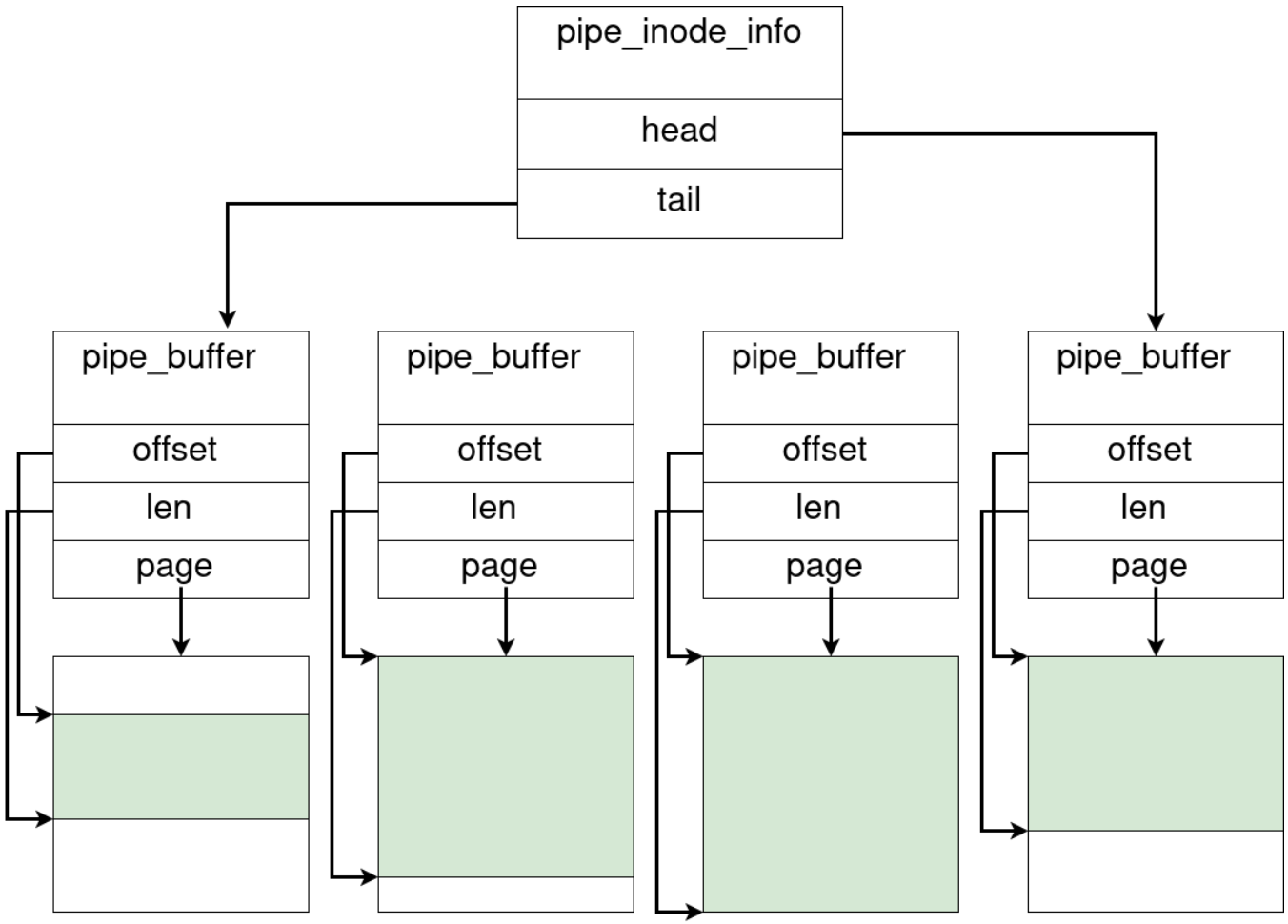


- enqueue(2)

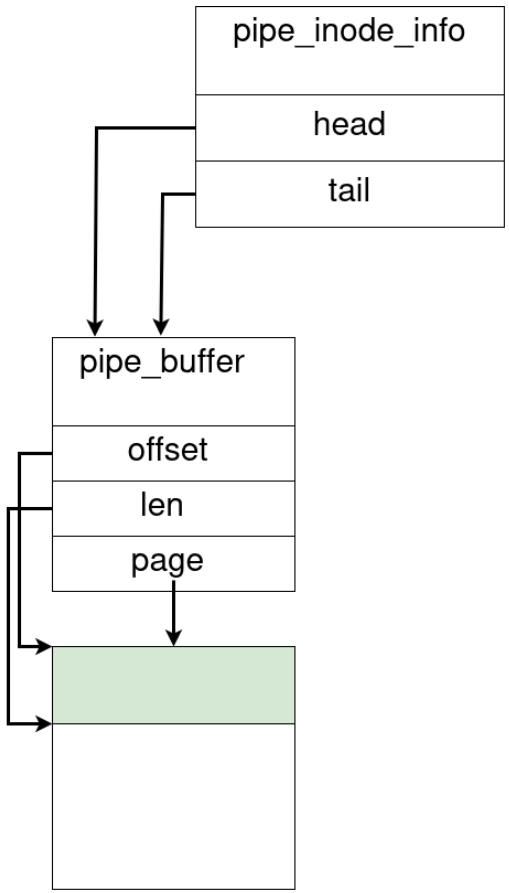


- enqueue(7)

Pipe buffer queue

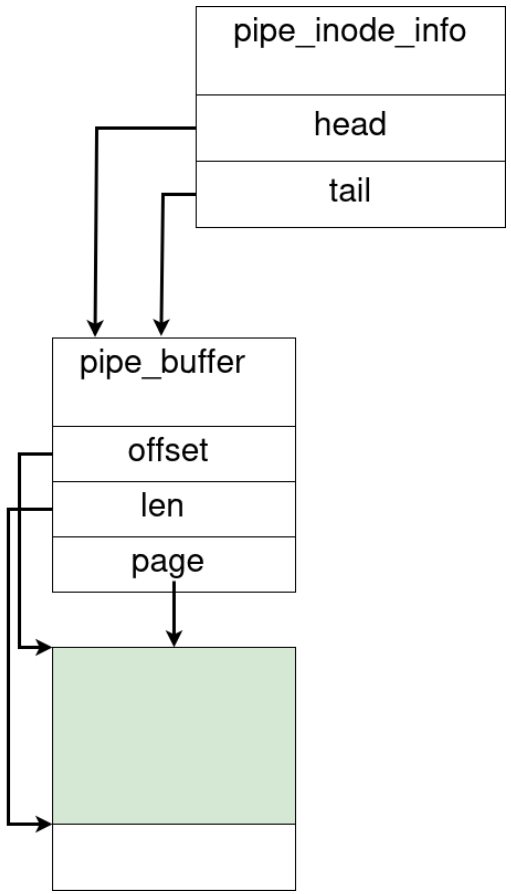


Pipe buffer queue



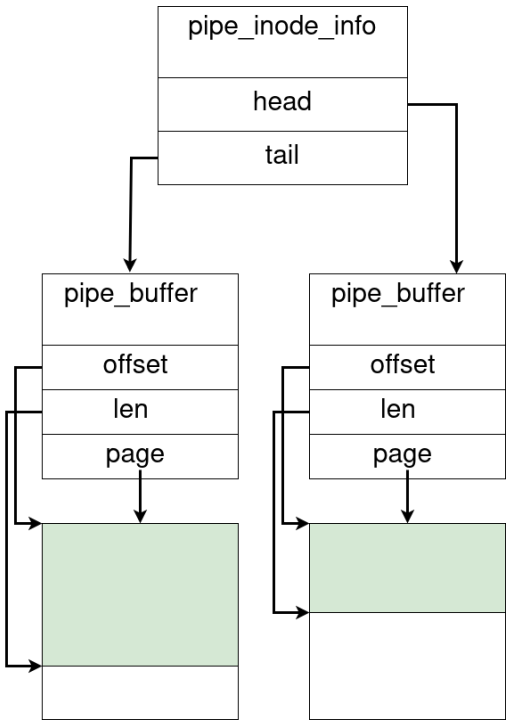
- write(pfd[1], buffer, 1000)

Pipe buffer queue



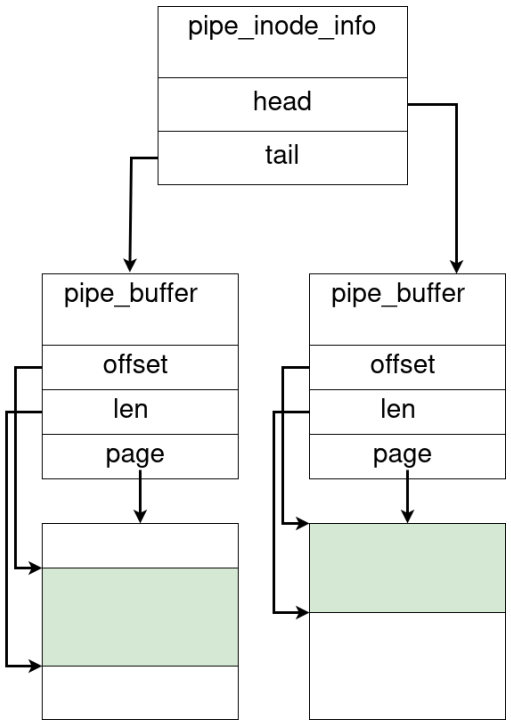
- write(pfd[1], buffer, 2000)

Pipe buffer queue



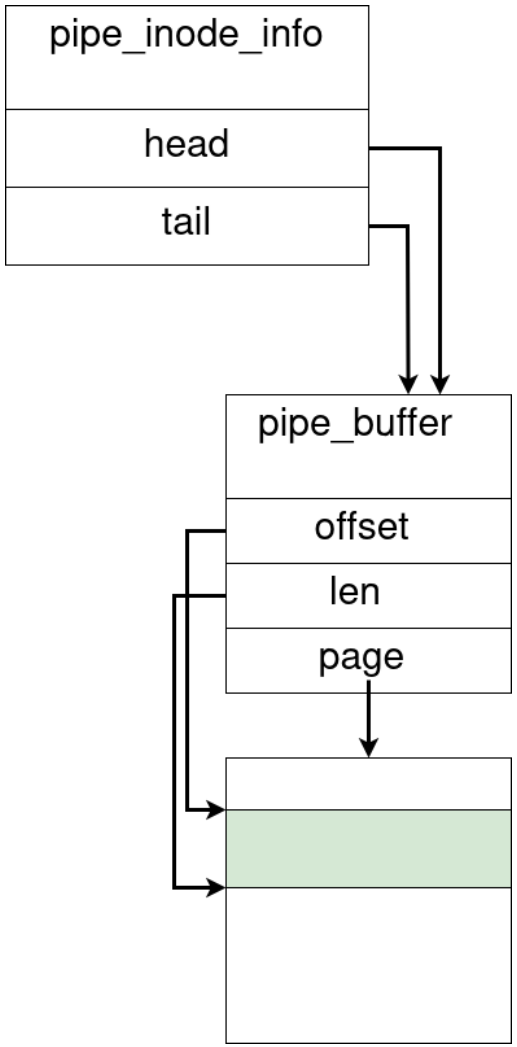
- write(pfd[1], buffer, 3000)

Pipe buffer queue



- read(pfd[0], buffer, 1000)

Pipe buffer queue



- read(pfd[0], buffer, 4000)

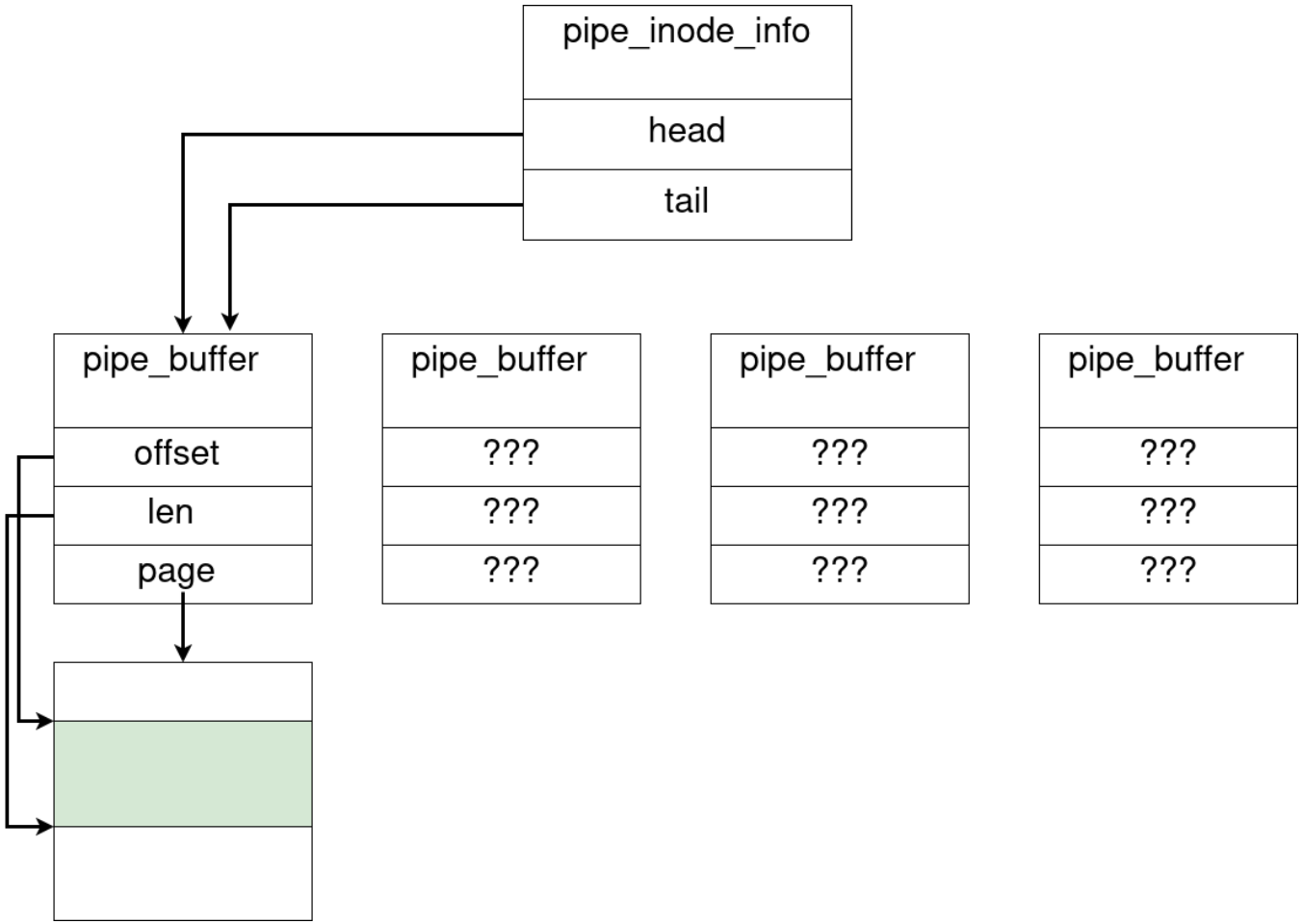
Pipe buffer queue

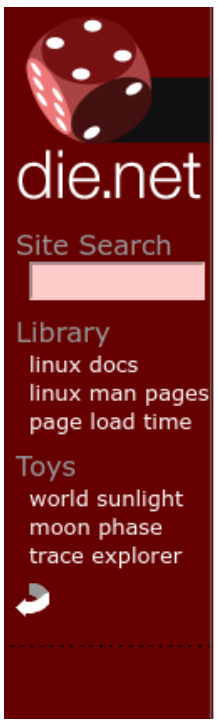
- All pipe_buffer structures are allocated when the pipe is created
- Aftewards, when the head/tail is moved, only a pointer assignment takes place

```
1 struct pipe_inode_info *alloc_pipe_info(void)
2 {
3         ...
4         pipe = kzalloc(sizeof(struct pipe_inode_info), GFP_KERNEL_ACCOUNT);
5
6         ...
7         pipe->bufs = kcalloc(pipe_bufs, sizeof(struct pipe_buffer),
8                              GFP_KERNEL_ACCOUNT);
```

```
1 pipe->head = head + 1;
2 spin_unlock_irq(&pipe->rd_wait.lock);
3
4 /* Insert it into the buffer array */
5 buf = &pipe->bufs[head & mask];
6 buf->page = page;
7 buf->ops = &anon_pipe_buf_ops;
8 buf->offset = 0;
9 buf->len = 0;
```

Pipe buffer queue





splice(2) - Linux man page

Name

splice - splice data to/from a pipe

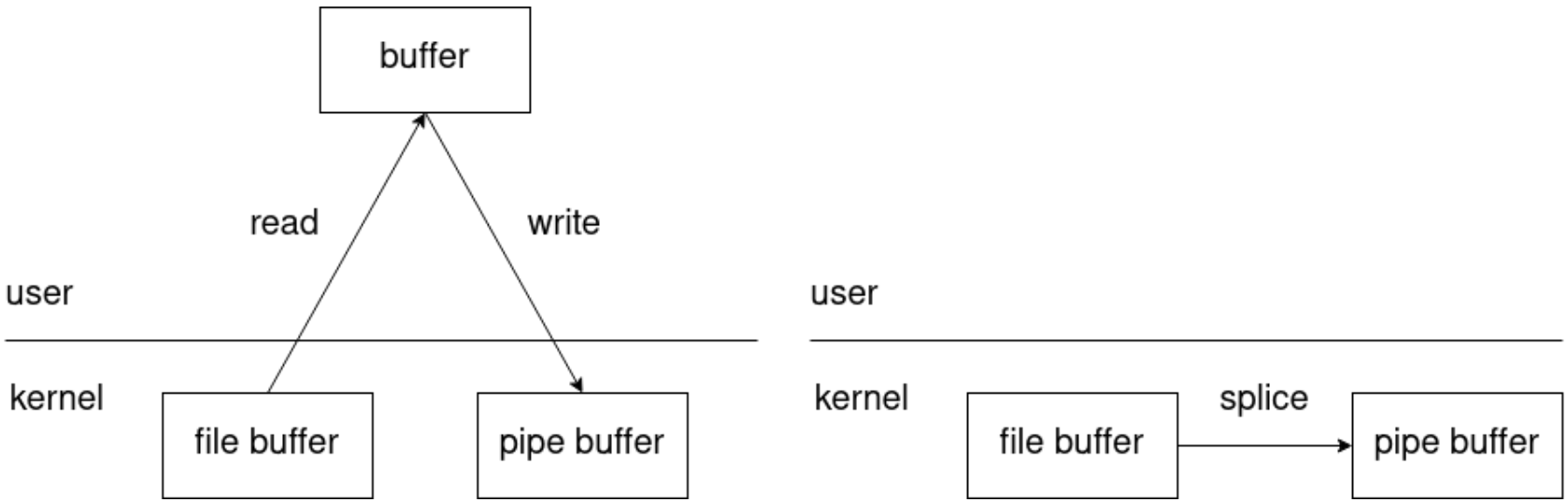
Synopsis

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <fcntl.h>

ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags );
```

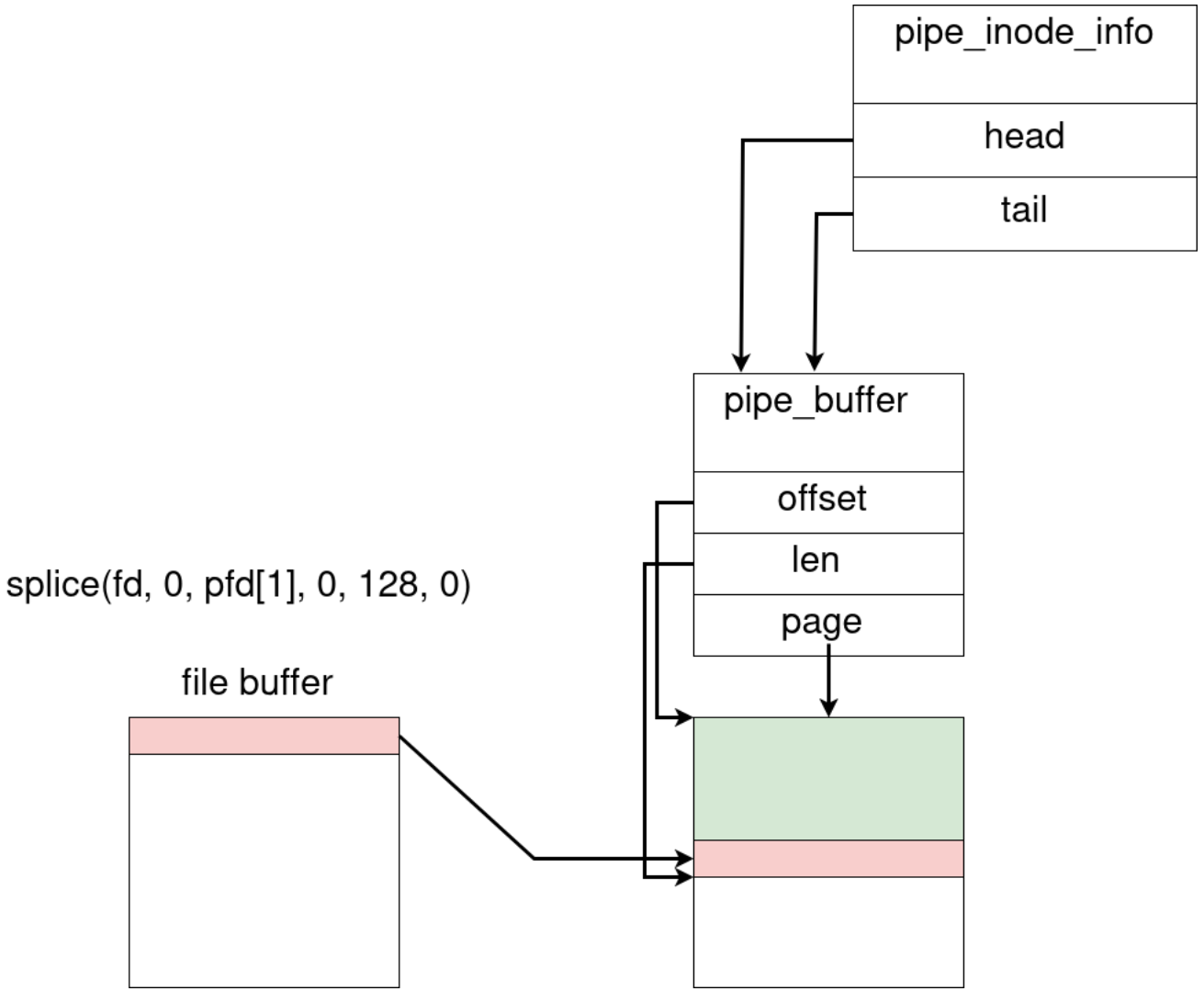
Description

splice() moves data between two file descriptors without copying between kernel address space and user address space. It transfers up to *len* bytes of data from the file descriptor *fd_in* to the file descriptor *fd_out*, where one of the descriptors must refer to a pipe.

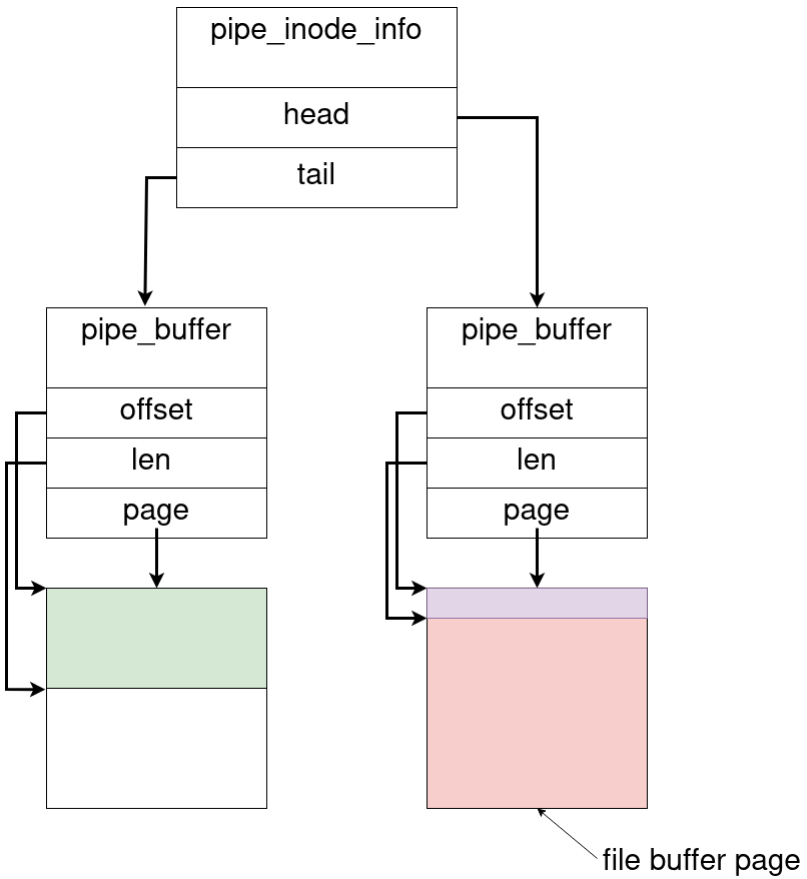
- zero-copy


buffer
read
write
user
kernel
file buffer
pipe buffer
user
kernel
splice
file buffer
pipe buffer

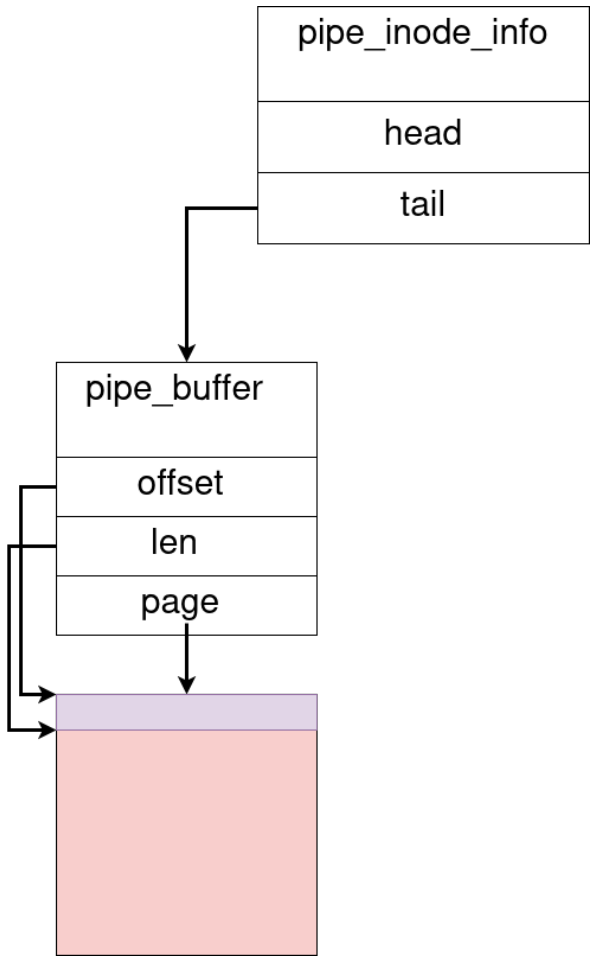
splice - kernel level (naive version)



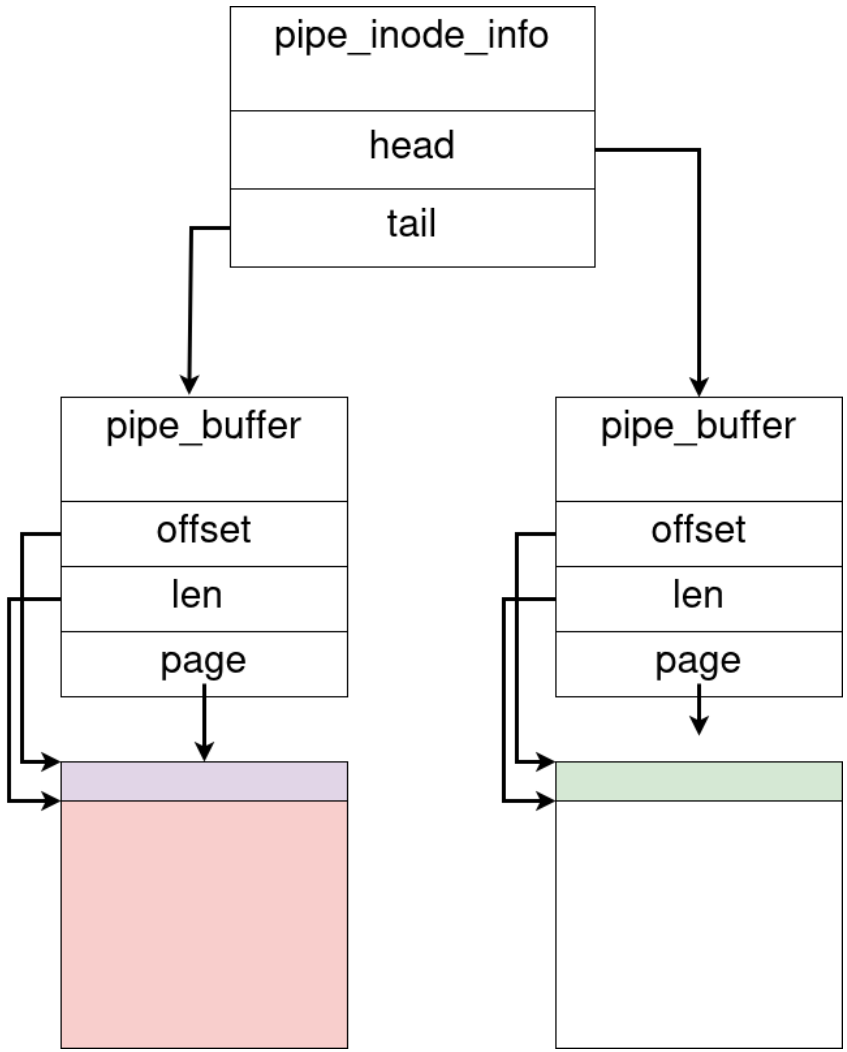
- The page from the page cache is "borrowed" by the pipe buffer
- Only a pointer assignment takes place



- splice(fd, 0, pfd[1], 0, 128, 0)
- write(pfd[1], buffer, 128)



- Cannot append data to existing page - it contains file data



splice - kernel level

- When writing to a pipe, the kernel must know whether it can append the data to the page from the pipe_buffer that sits at the front of the queue
- in kernel terminology, the pipe_buffer is “mergeable”

pipe write

```
1  pipe_write(struct kiocb *iocb, struct iov_iter *from)
2  {
3          ...
4          if (chars && !was_empty) {
5                  unsigned int mask = pipe->ring_size - 1;
6                  struct pipe_buffer *buf = &pipe->bufs[(head - 1) & mask];
7                  int offset = buf->offset + buf->len;
8
9                  if ((buf->flags & PIPE_BUF_FLAG_CAN_MERGE) &&
10                     offset + chars <= PAGE_SIZE) {
11                         ...
12                         ret = copy_page_from_iter(buf->page, offset, chars, from
       );
13                         ...
14                 }
15         }
16
17         for (;;) {
18                 if (!pipe_full(head, pipe->tail, pipe->max_usage)) {
19                         ...
20                         pipe->head = head + 1;
21                         /* Insert it into the buffer array */
22                         buf = &pipe->bufs[head & mask];
23                         buf->page = page;
24                         buf->ops = &anon_pipe_buf_ops;
25                         buf->offset = 0;
26                         buf->len = 0;
27
28                         ...
29                         copied = copy_page_from_iter(page, 0, PAGE_SIZE, from);
```

mergeable buffers

- "normal" pipe_buffers are mergeable
- pipe_buffers holding spliced pages are not mergeable

Outline

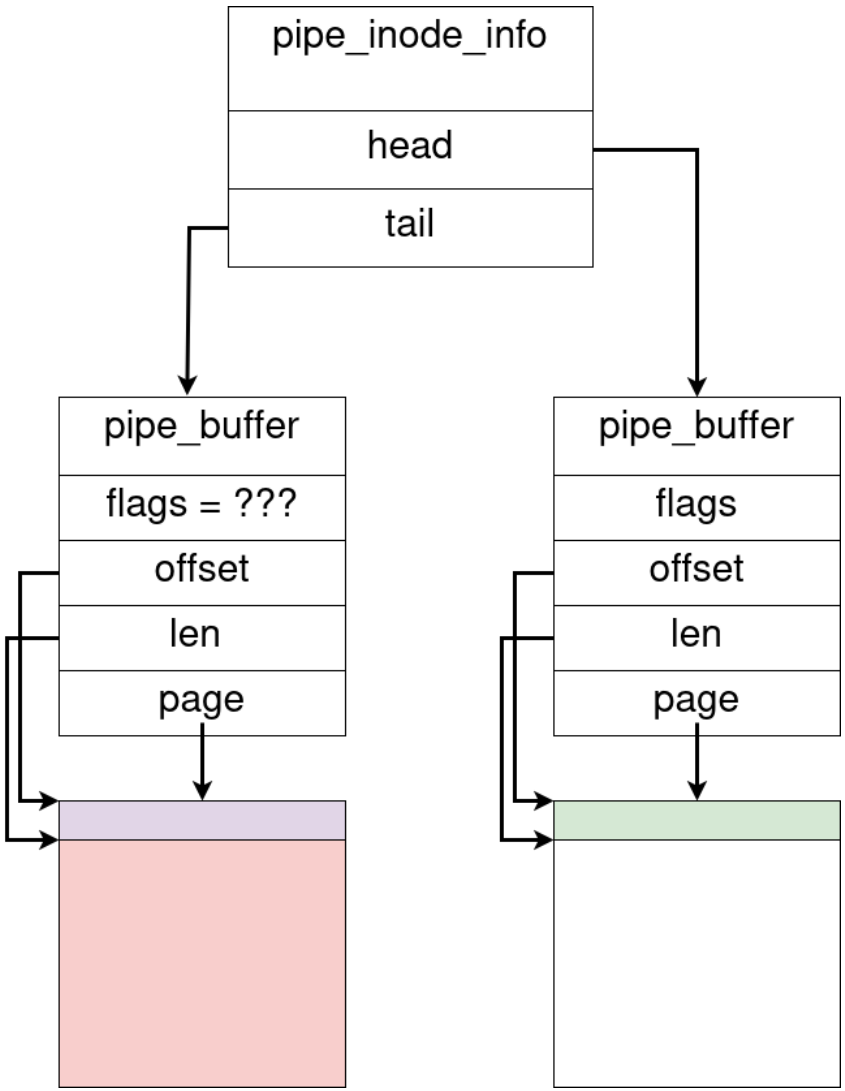
DirtyPipe vulnerability

```
From: Max Kellermann <max.kellermann@ionos.com>
To: linux-kernel@vger.kernel.org, viro@zeniv.linux.org.uk,
[...]

The functions copy_page_to_iter_pipe() and push_pipe() can both
allocate a new pipe_buffer, but the "flags" member initializer is
missing.
[...]

diff --git a/lib/iov_iter.c b/lib/iov_iter.c
index b0e0acdf96c1..6dd5330f7a99 100644
--- a/lib/iov_iter.c
+++ b/lib/iov_iter.c
@@ -414,6 +414,7 @@ static size_t copy_page_to_iter_pipe(struct page *page,
     size_t offset, size_t by
                return 0;

        buf->ops = &page_cache_pipe_buf_ops;
+       buf->flags = 0;
        get_page(page);
        buf->page = page;
        buf->offset = offset;
@@ -577,6 +578,7 @@ static size_t push_pipe(struct iov_iter *i, size_t size,
                        break;

                buf->ops = &default_pipe_buf_ops;
+               buf->flags = 0;
                buf->page = page;
                buf->offset = 0;
                buf->len = min_t(ssize_t, left, PAGE_SIZE);
```

DirtyPipe vulnerability

- pipe_buffers created by splice don't have the `flags` initialized

DirtyPipe vulnerability



DirtyPipe vulnerability

- if `flags` happen to have the `0x10` bit set (`PIPE_BUF_FLAG_CAN_MERGE`), subsequent data will be written to the page "borrowed" from the page cache
- normal users can modify files on which they only have read access

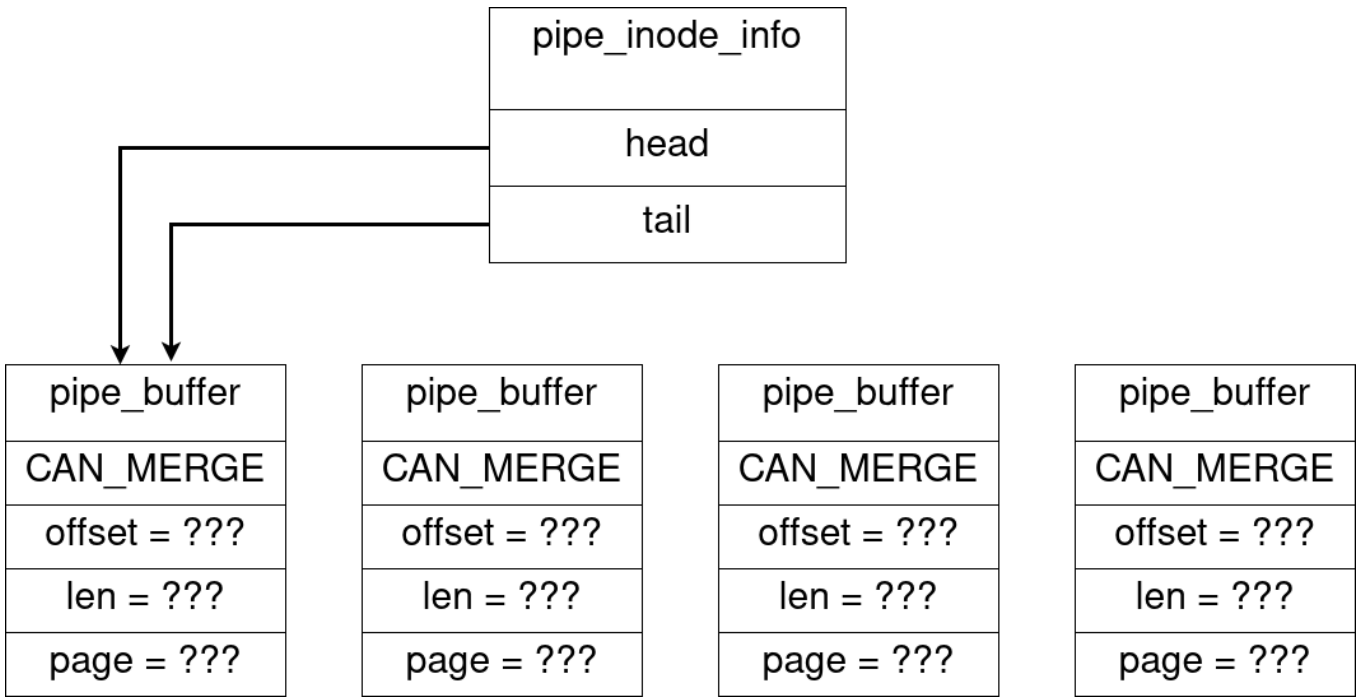
Exploit steps

- open a sensitive file
- splice one byte from the file into a pipe
- write more data to the pipe
- if lucky (the uninitialized `flags` have the `0x10` bit set), this data will be written to the file
- profit

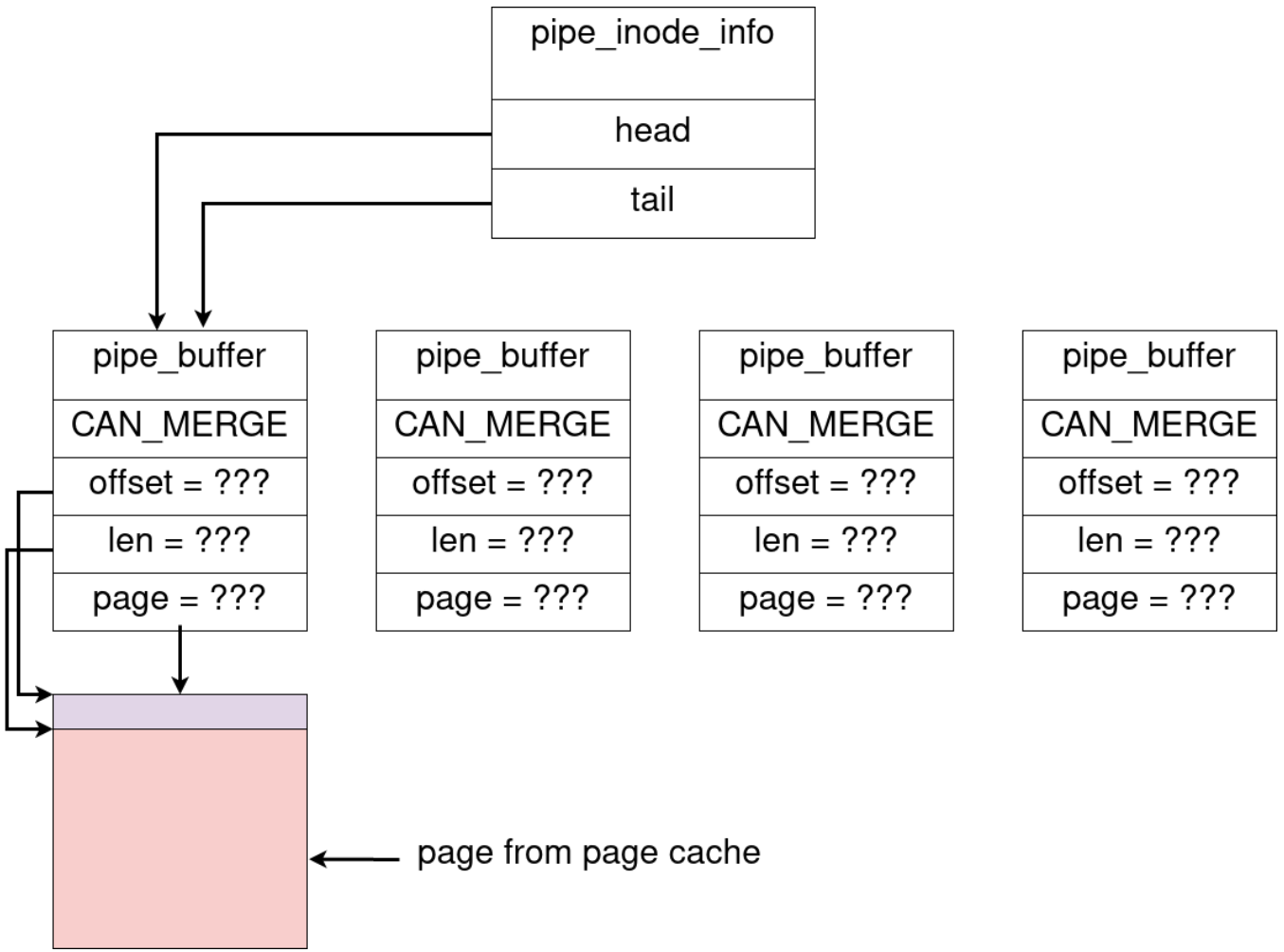
Exploit

- To improve our chances
  - fill the entire pipe with data, then read it back
  - this way all the pipe_buffers inside the pipe structure will have the `PIPE_BUF_FLAG_CAN_MERGE` set (because normal pipe_buffers have this flag set)
- Limitations
  - Cannot overwrite at offset 0 - because we need to splice at least 1 byte
  - Cannot cross page boundary

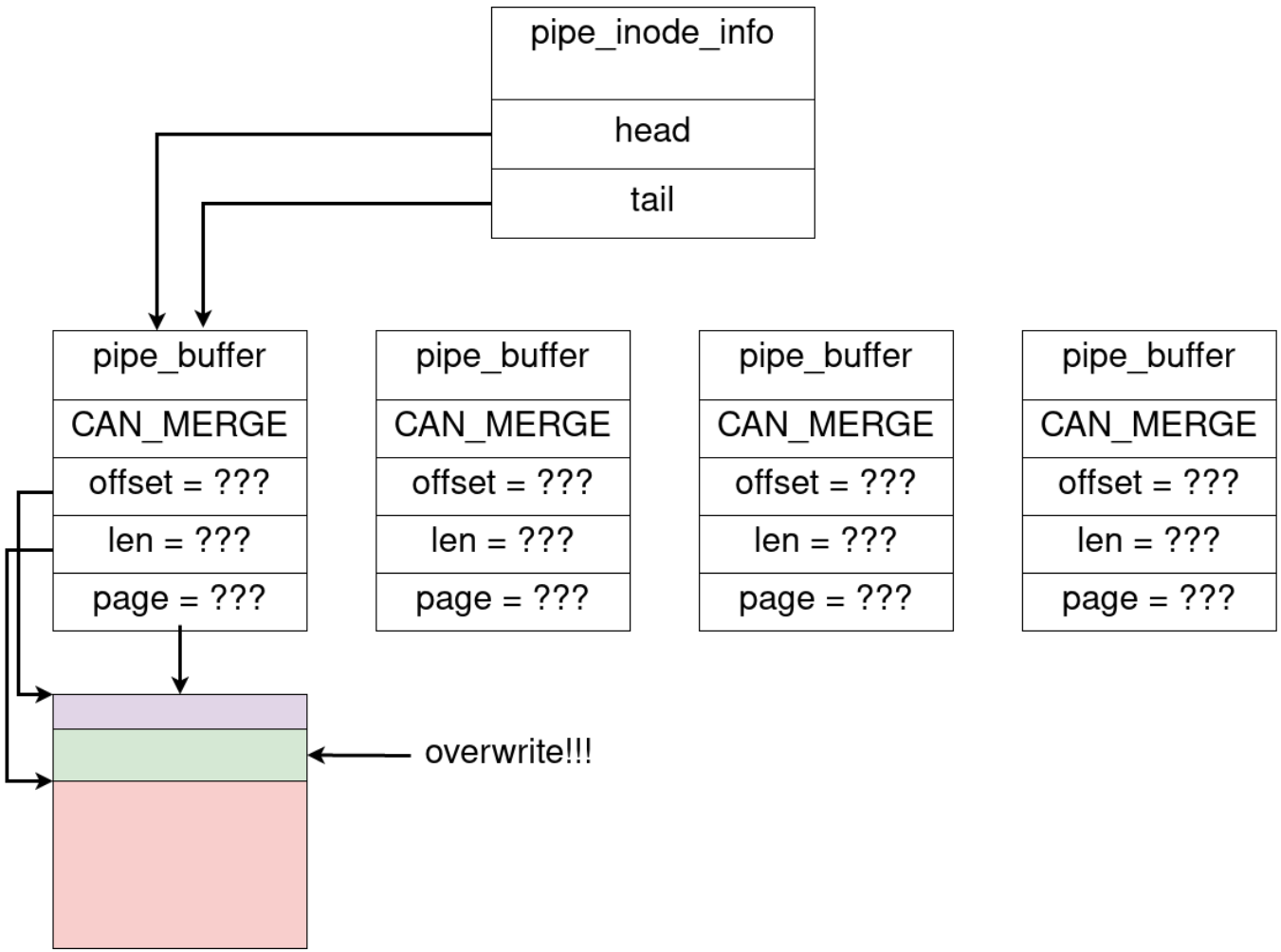
- write(); write(); write(); read(); read(); read()



- splice(fd, 0, pfd[1], 0, 1, 0)



- write(pfd[1], buffer, 128)

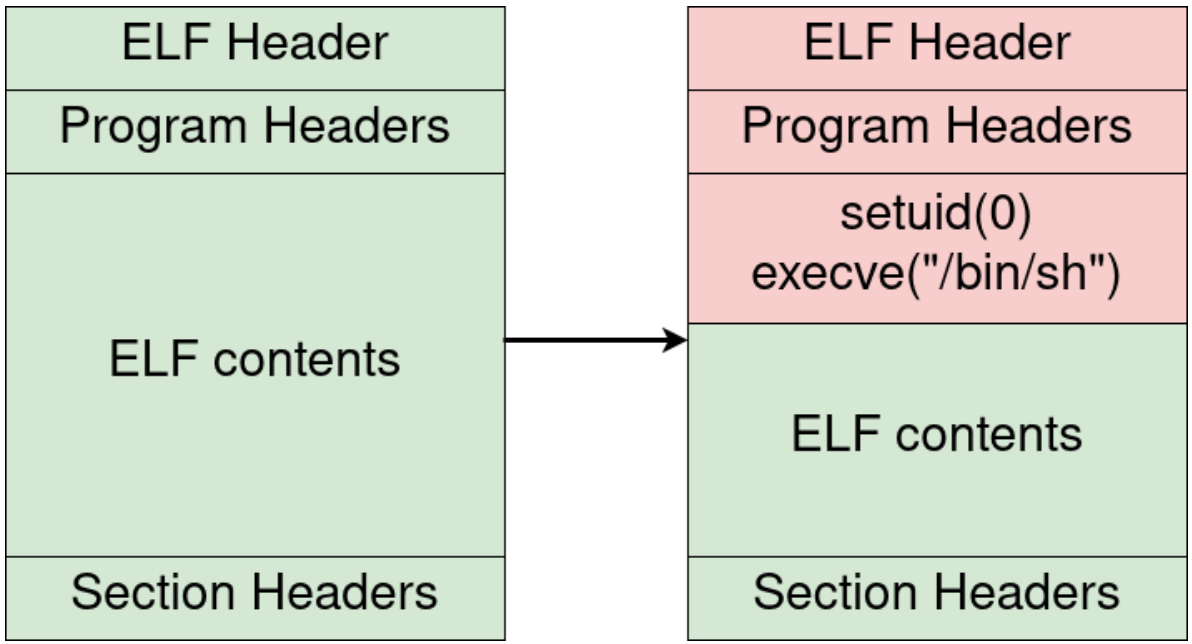


Outline

Exploit 1 - overwrite setuid ELF

- pick a setuid file (like `/usr/bin/sudo`)
- overwrite it completely from the start of the file
- replace contents witha small ELF that calls `setuid(0);` `execve(``/bin/sh'');`
- exploit in `exploit_elf`

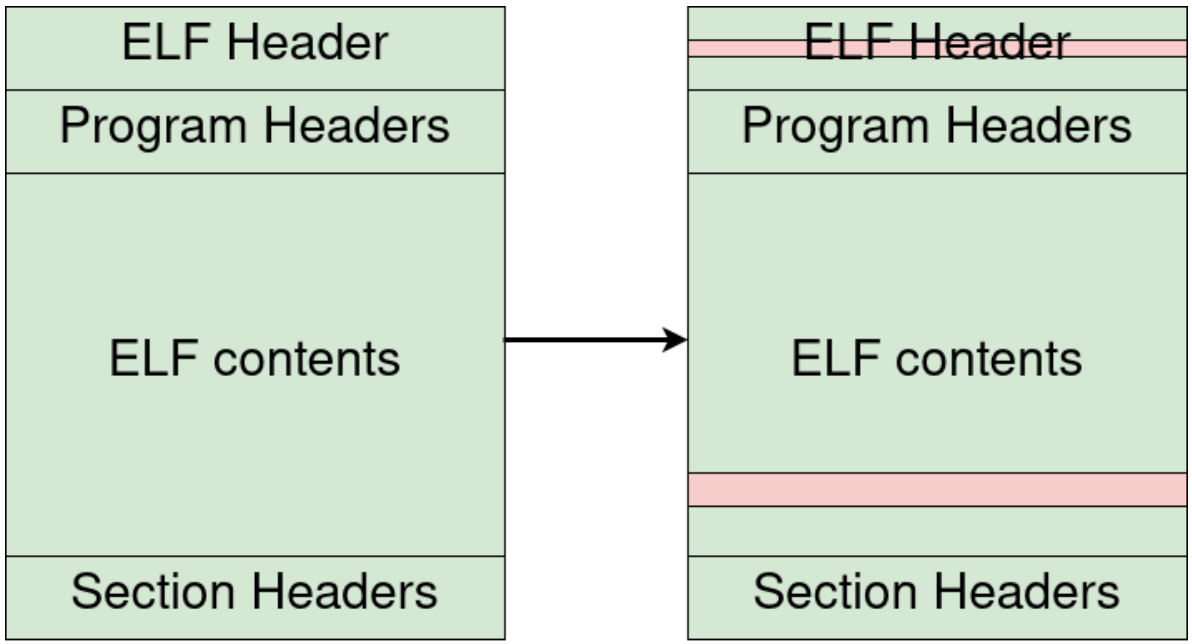
Exploit 1 - overwrite setuid ELF



Exploit 2 - backdoor setuid ELF

- inject our malicious code somewhere inside the binary
- overwrite the ELF entry address to point to our code
- the code only executes a shell if some conditions are met, otherwise jumps to the original entry point
- exploit in `exploit_backdoor`

Exploit 2 - backdoor setuid ELF



Outline

Resources

- support archive: `http://elf.cs.pub.ro/cns/res/lectures/11-exploit-demo-1-support.zip`

References

- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0847
- https://dirtypipe.cm4all.com/