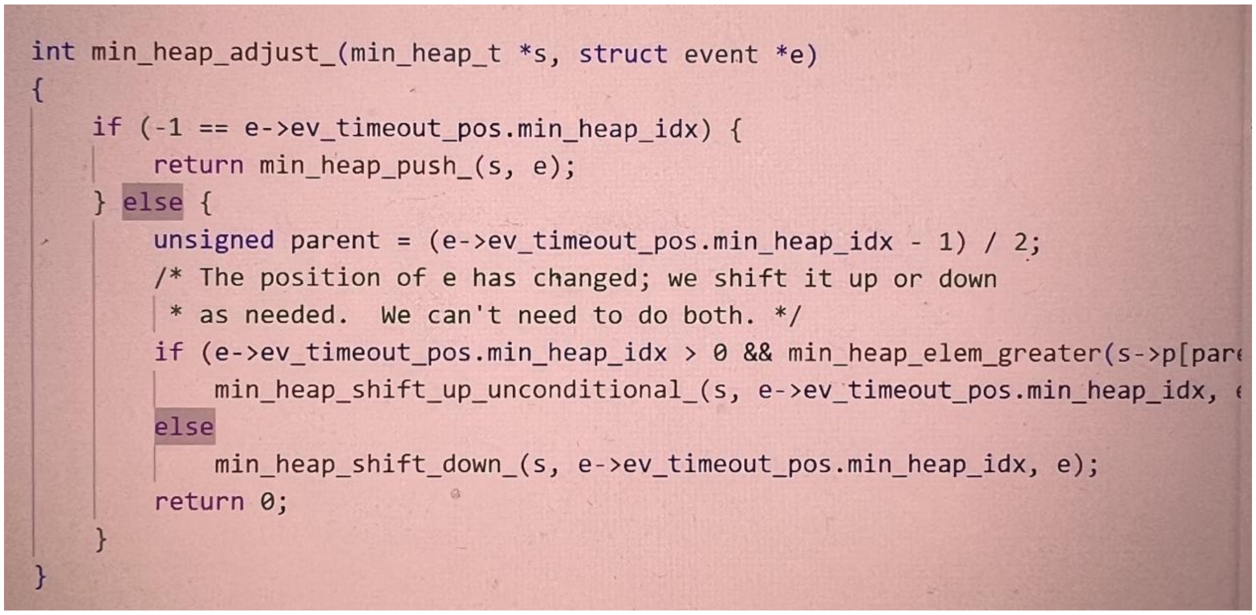
1. This bug deals with a variable whose value is always NULL. Despite this, this variable is still used in an if statement where it will always return true due to the variable being NULL. This can be a problem because the if statement may not function as intended. The impact of this is that the if statement will keep always running making the if statement obsolete, or the unintentional running of the if statement. This may cause the program to do something that it was not intended to do.
    a. If (chain == NULL) {
    b. Chain = evbuffer_chain_new(datlen);
    c. If (!chain)
    d. Goto done;
    e. Evbuffer_chain_insert(buf, chain);
    f. }
    g. buffer.c, line: 1849, line: 1757
        i. There are a few ways to fix this, one way would be to create a new variable that is made specifically for the second if statement so that the NULL variable is only used in the first if statement.
        ii. Severity rating: HIGH

2. This bug deals with a function that is never called. The function is defined with seemingly no errors, but it is never called anywhere in the code. This problem isn't a very big problem because the code is ignoring the function which means Its doing nothing. The impact is very minimal with the function not doing anything.



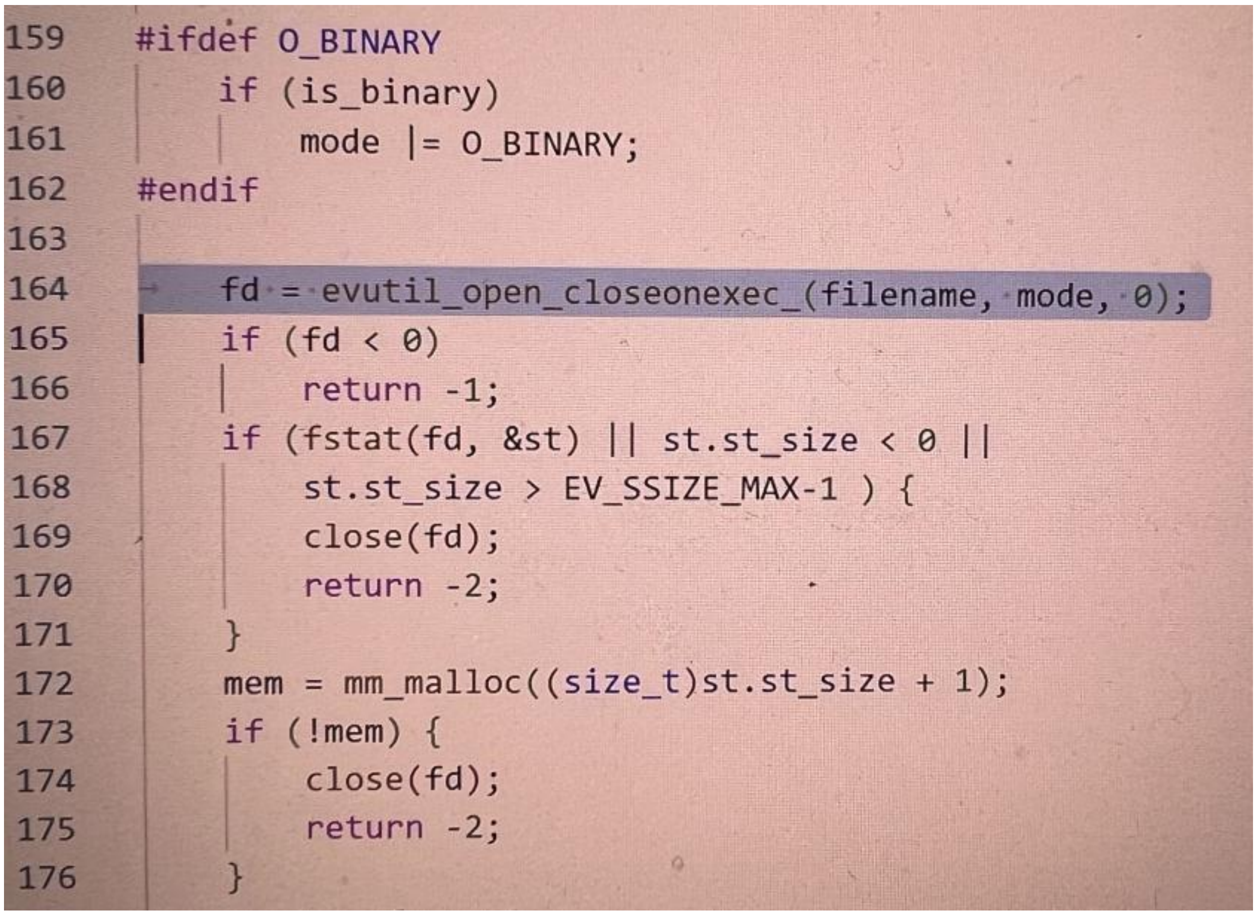
```
int min_heap_adjust_(min_heap_t *s, struct event *e)
{
    if (-1 == e->ev_timeout_pos.min_heap_idx) {
        return min_heap_push_(s, e);
    } else {
        unsigned parent = (e->ev_timeout_pos.min_heap_idx - 1) / 2;
        /* The position of e has changed; we shift it up or down
         * as needed.  We can't need to do both. */
        if (e->ev_timeout_pos.min_heap_idx > 0 && min_heap_elem_greater(s->p[pare
            min_heap_shift_up_unconditional_(s, e->ev_timeout_pos.min_heap_idx,
        else
            min_heap_shift_down_(s, e->ev_timeout_pos.min_heap_idx, e);
        return 0;
    }
}
```

    a.
    b. Minheap-internal.h, line: 117, line: 79

c. There is a way to fix this which is to remove the function completely so that it resolves the issue of not being called in the code.

d. Severity rating: LOW

3. This bug deals with a file that may not be closed at a certain point in the code. There are many points in the code where this file does not close. There are many problems this can cause such as making the file corrupted due the possibility of written data not actually being saved. Another problem is a security one where an attacker could take advantage of the file being open to steal sensitive data.



```
159  #ifdef O_BINARY
160      if (is_binary)
161          mode |= O_BINARY;
162  #endif
163
164      fd = evutil_open_closeonexec_(filename, mode, 0);
165      if (fd < 0)
166          return -1;
167      if (fstat(fd, &st) || st.st_size < 0 ||
168          st.st_size > EV_SSIZE_MAX-1 ) {
169          close(fd);
170          return -2;
171      }
172      mem = mm_malloc((size_t)st.st_size + 1);
173      if (!mem) {
174          close(fd);
175          return -2;
176      }
```

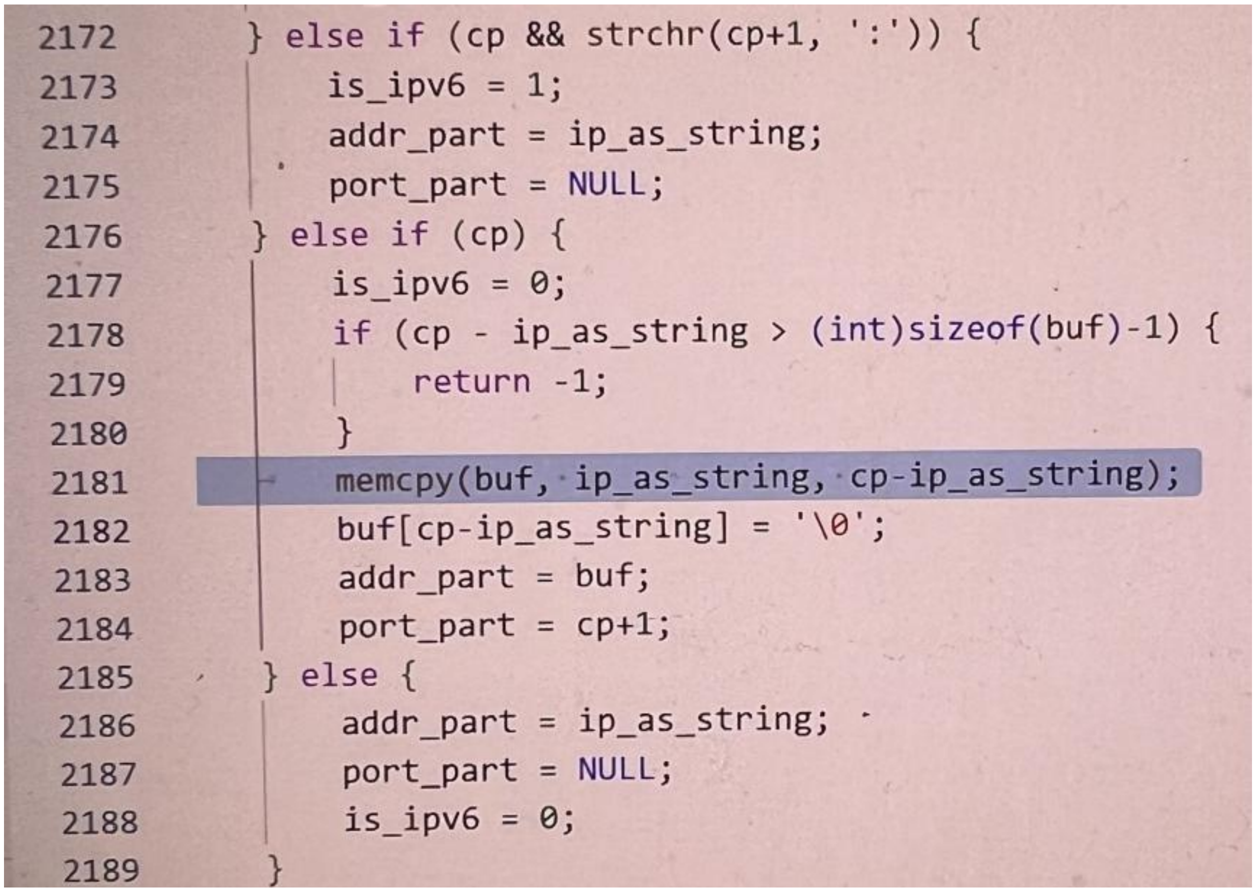
a.

b. Evutil.c, line 164, line 115, line 120

c. One way to fix this is to create a way to make sure that the file is actually closed. In my research, I found that you can simulate a try-finally code sequence in C with goto statements. This ensures that the file will be opened in the try block, and close in the finally block.

d. Severity rating: HIGH

4. This bug deals with there being a potential overflow being caused. This is due to the buffer being bound by the source-buffer size and not the destination-buffer size. The impact buffer overflow can have is that it can lead to memory corruption and could give an attacker access to sensitive information. This can lead to the attacker gaining full control of the system.
    a.

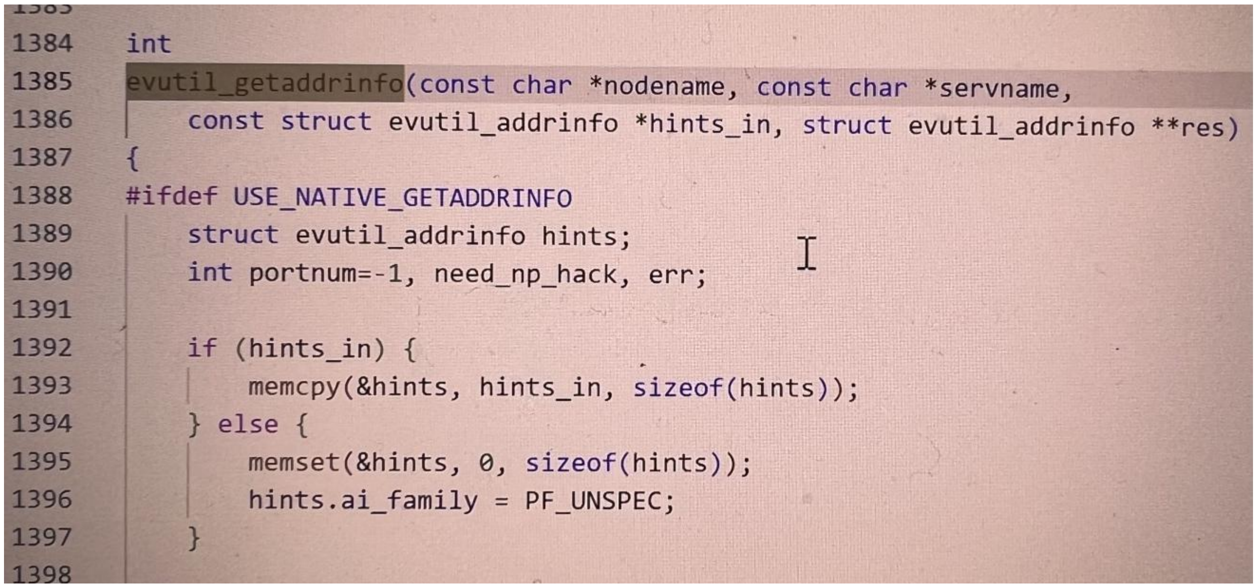


```
2172        } else if (cp && strchr(cp+1, ':')) {
2173            is_ipv6 = 1;
2174            addr_part = ip_as_string;
2175            port_part = NULL;
2176        } else if (cp) {
2177            is_ipv6 = 0;
2178            if (cp - ip_as_string > (int)sizeof(buf)-1) {
2179                return -1;
2180            }
2181            memcpy(buf, ip_as_string, cp-ip_as_string);
2182            buf[cp-ip_as_string] = '\0';
2183            addr_part = buf;
2184            port_part = cp+1;
2185        } else {
2186            addr_part = ip_as_string;
2187            port_part = NULL;
2188            is_ipv6 = 0;
2189        }
```

    b. Evutil.c, line: 2181
    c. One way to fix this as discussed above is to make sure that the buffer is bounded by the destination-buffer and not the source in order to make sure that there is no buffer overflow. It is important to copy data safely in order to prevent buffer overflow.
    d. Severity rating: CRITICAL

5. This bug deals with a variable that may not be initialized when it is called. The variable may be called but due to not being initialized, the code can run into some error. This problem this can cause is that the variable can have some garbage value due to it not being initialized. This can lead to the code performing in a way it is not intended to perform.
    a.
    b. event.c, line: 438, line: 1280, line: 2113
    c. One way to fix this bug is to explicitly initialize the variable. This will make sure that problems don't occur when using this variable. In my research, I found that stderr is

already predefined in C so it should be initialized when the program runs and the compiler may be outputting a false positive. It is always important to make sure that the variables are initialized in order to avoid problems.

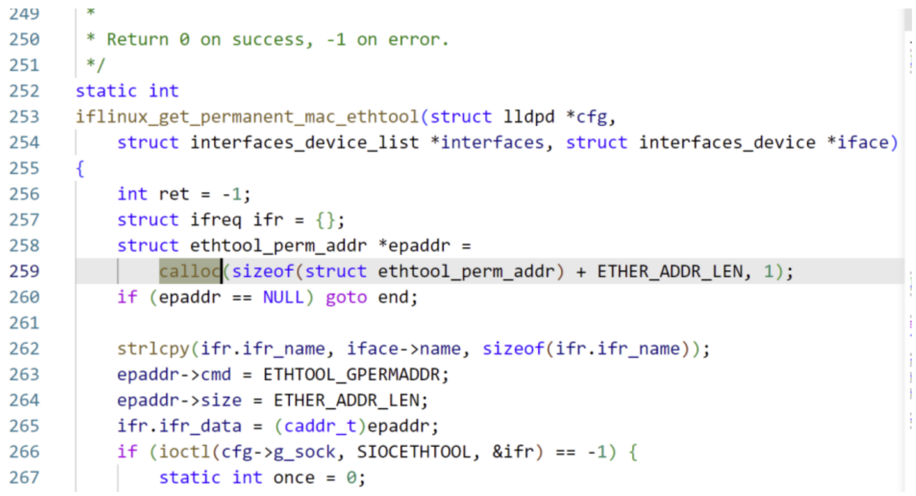


```
int
evutil_getaddrinfo(const char *nodename, const char *servname,
    const struct evutil_addrinfo *hints_in, struct evutil_addrinfo **res)
{
#ifdef USE_NATIVE_GETADDRINFO
    struct evutil_addrinfo hints;
    int portnum=-1, need_np_hack, err;

    if (hints_in) {
        memcpy(&hints, hints_in, sizeof(hints));
    } else {
        memset(&hints, 0, sizeof(hints));
        hints.ai_family = PF_UNSPEC;
    }
```

d. Severity rating: MED

6. This bug deals with the memory allocated not being the right multiple. A variable ethtool_perm_addr has a size of14 bytes, but that generated a warning because it is not a multiple of 8. The problems this can cause are memory leaks, buffer overflows, etc.

a. 

```
249  *
250  * Return 0 on success, -1 on error.
251  */
252 static int
253 iflinux_get_permanent_mac_ethtool(struct lldpd *cfg,
254     struct interfaces_device_list *interfaces, struct interfaces_device *iface)
255 {
256     int ret = -1;
257     struct ifreq ifr = {};
258     struct ethtool_perm_addr *epaddr =
259         calloc(sizeof(struct ethtool_perm_addr) + ETHER_ADDR_LEN, 1);
260     if (epaddr == NULL) goto end;
261
262     strlcpy(ifr.ifr_name, iface->name, sizeof(ifr.ifr_name));
263     epaddr->cmd = ETHTOOL_GPERMADDR;
264     epaddr->size = ETHER_ADDR_LEN;
265     ifr.ifr_data = (caddr_t)epaddr;
266     if (ioctl(cfg->g_sock, SIOCETHTOOL, &ifr) == -1) {
267         static int once = 0;
```

b. Interfaces-linux.c, line: 259

c. A way to fix this error is to simply allocate enough memory so that it is a multiple of 8. This can be done by adding 2 more bytes making it 16 which would then be a multiple of 8. The severity of this bug depends on how the program uses the memory. Examples can be the code only reads from the memory and doesn't write to it which wouldn't have any immediate problems. But problems can occur if the program writes to the memory which can then cause severe problems.

d. Severity rating: MED