Semmle™

OSS security, here be dragons!

Fermin J. Serna - @fjserna

Chief Security Officer @ Semmle

Distinguished Engineer @ GitHub

Previously:

- Head of Product Security @ Google

Other:

- Two times Pwnie Award nomination
  - 2016 winner: glibc getaddrinfo()
- EMET initial main developer
- I once owned Charlie Miller's iPhone at Pwn2Own

@fjserna

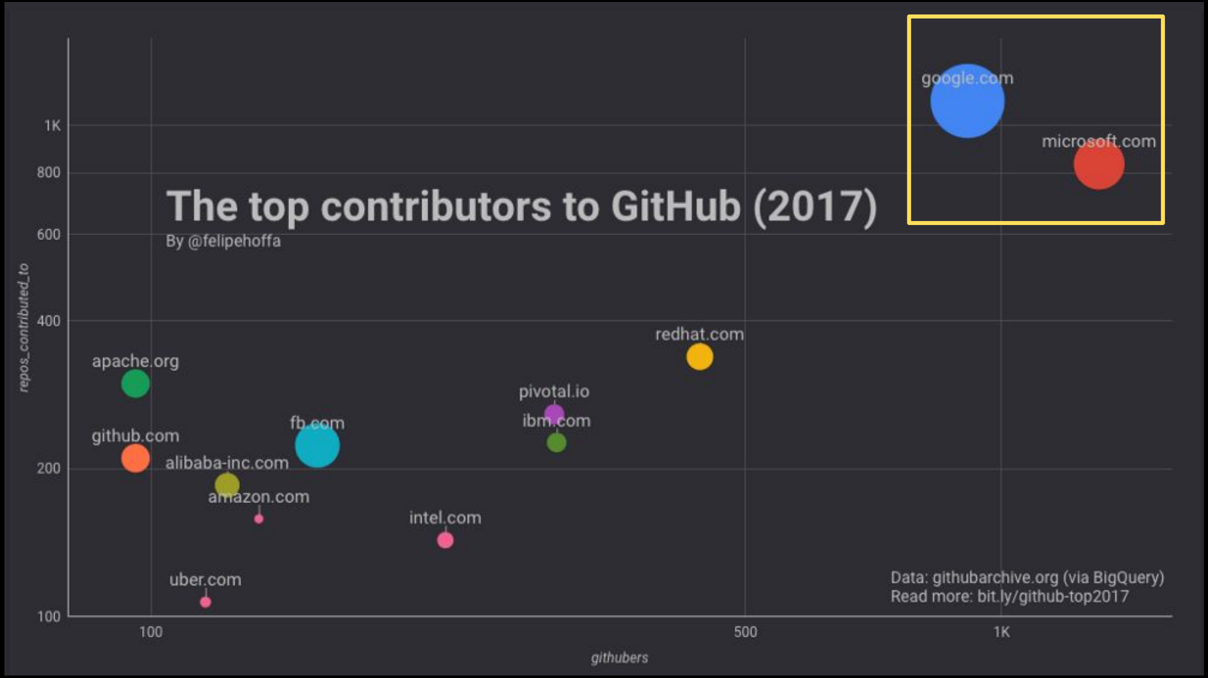
The Problem

OSS has won! What about security?

99% of software uses open source

Open Source has won!
But it comes with challenges:

- Code quality
- Supply chain
- Common development best practices:
  - Code review
  - Testing
  - ...
- Security: CVE tagging

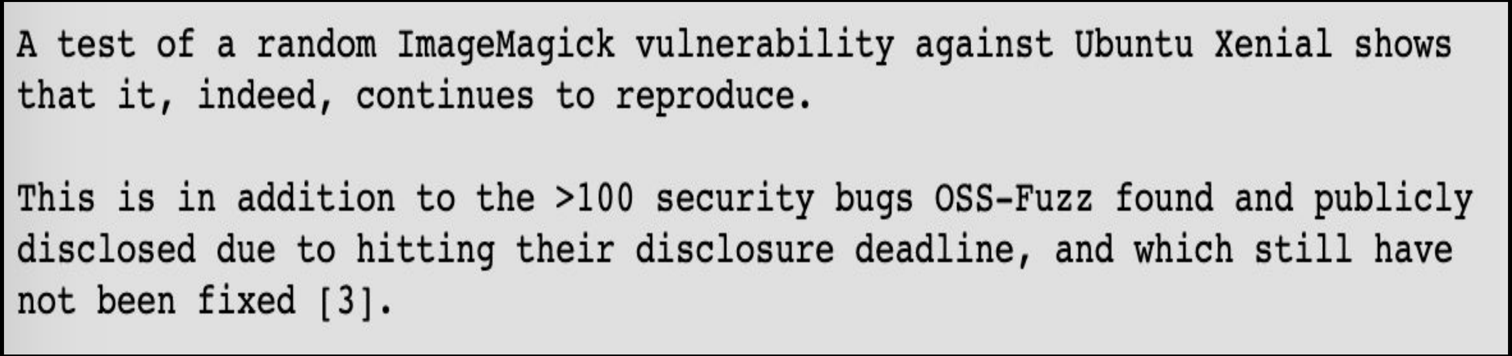


https://resources.whitesourcesoftware.com/blog-whitesource/git-much-the-top-10-companies-contributing-to-open-source

CVEs in OSS

Common Vulnerabilities and Exposures

The Standard for Information Security Vulnerability Names

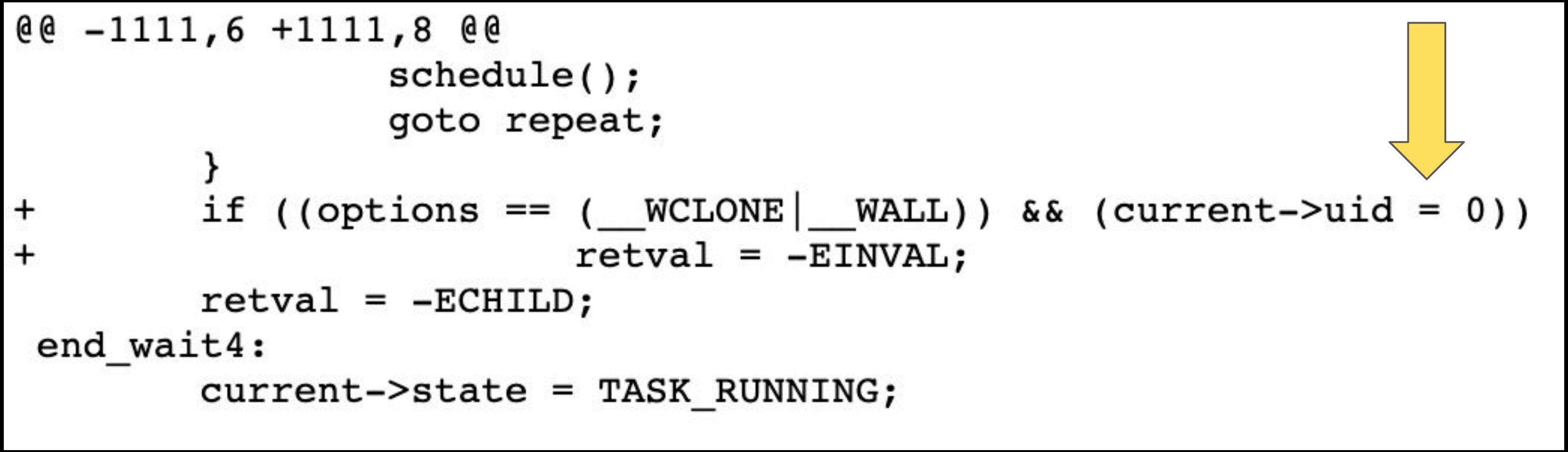


A test of a random ImageMagick vulnerability against Ubuntu Xenial shows that it, indeed, continues to reproduce.

This is in addition to the >100 security bugs OSS-Fuzz found and publicly disclosed due to hitting their disclosure deadline, and which still have not been fixed [3].

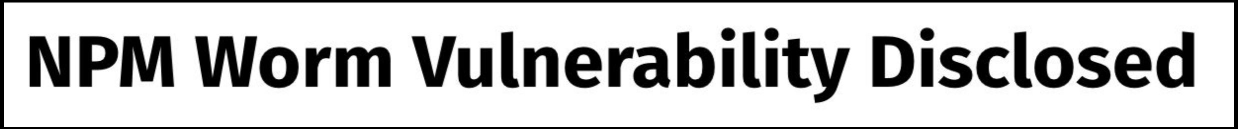
https://www.openwall.com/lists/oss-security/2019/06/15/2

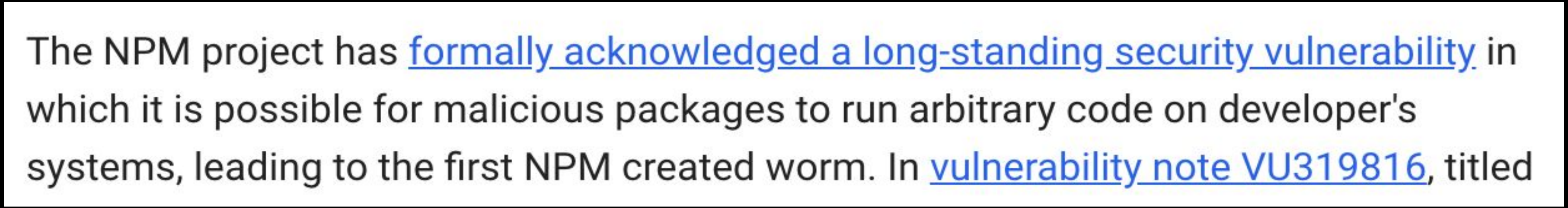
Linux Kernel Backdoor (2003)



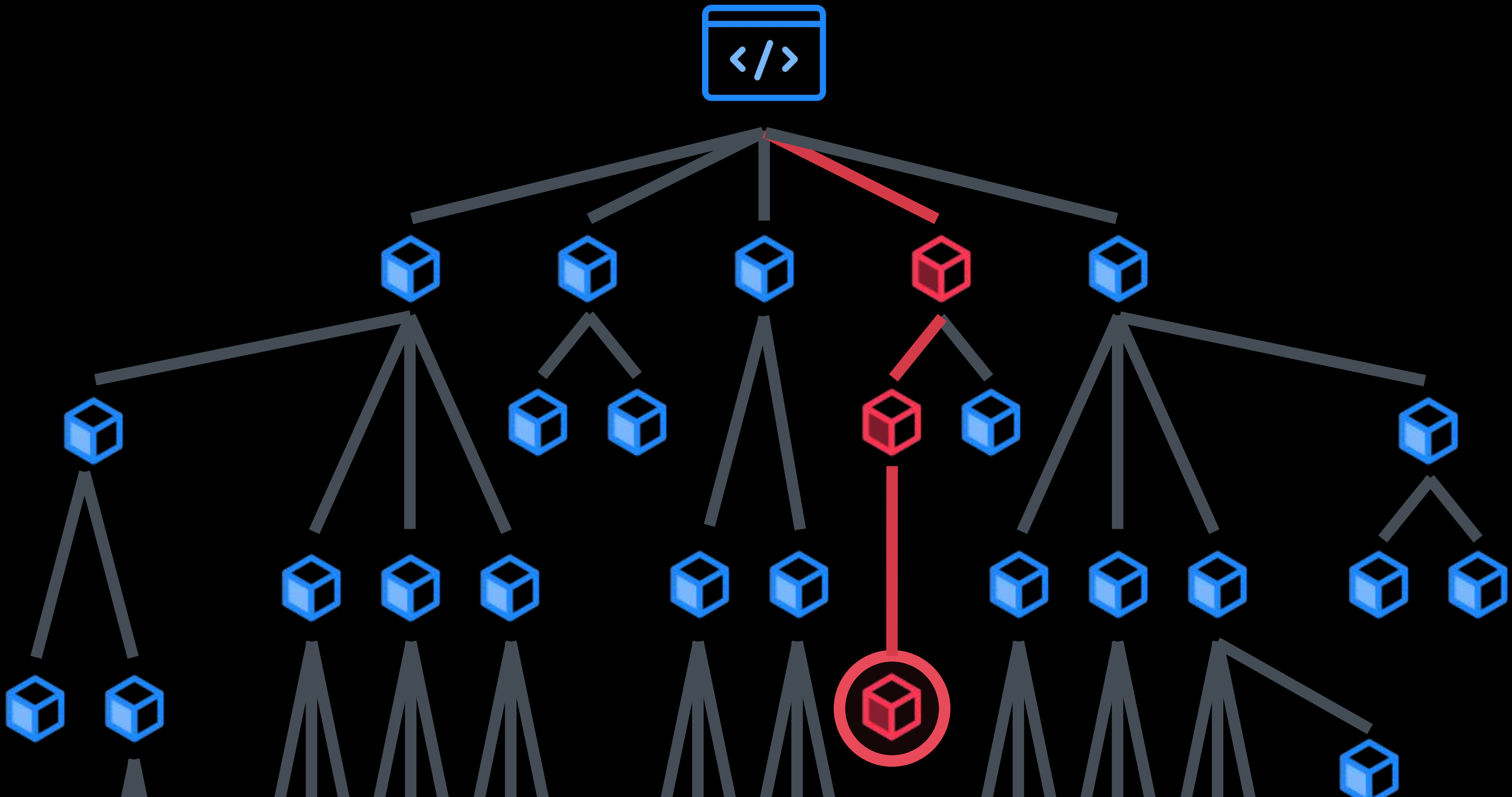
```
@@ -1111,6 +1111,8 @@
                 schedule();
                 goto repeat;
         }
+        if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+                        retval = -EINVAL;
         retval = -ECHILD;
 end_wait4:
         current->state = TASK_RUNNING;
```

Supply chain problems...


NPM Worm Vulnerability Disclosed

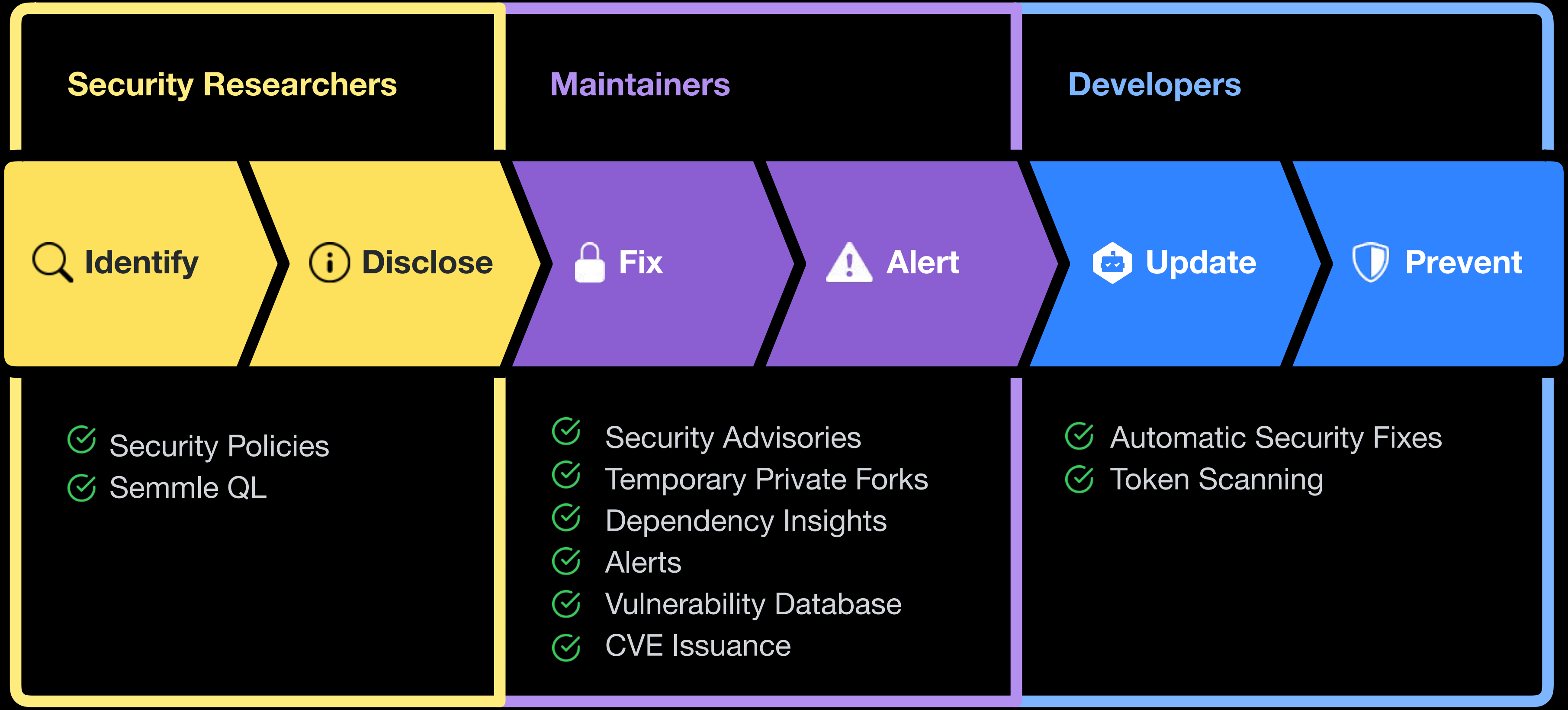

The NPM project has formally acknowledged a long-standing security vulnerability in which it is possible for malicious packages to run arbitrary code on developer's systems, leading to the first NPM created worm. In vulnerability note VU319816, titled

https://www.infoq.com/news/2016/03/npm-infection/



What is Github doing?

Open Source Security Workflow



Security Researchers

Identify → Disclose

- ✓ Security Policies
- ✓ Semmle QL

Maintainers

Fix → Alert

- ✓ Security Advisories
- ✓ Temporary Private Forks
- ✓ Dependency Insights
- ✓ Alerts
- ✓ Vulnerability Database
- ✓ CVE Issuance

Developers

Update → Prevent

- ✓ Automatic Security Fixes
- ✓ Token Scanning

Semmle QL - Automating Variant Analysis

QL is an object oriented **query language** for exploring code as data

Treat code as data

Security engineers use QL to

- Explore code iteratively
- Map attack surfaces
- Automate variant analysis

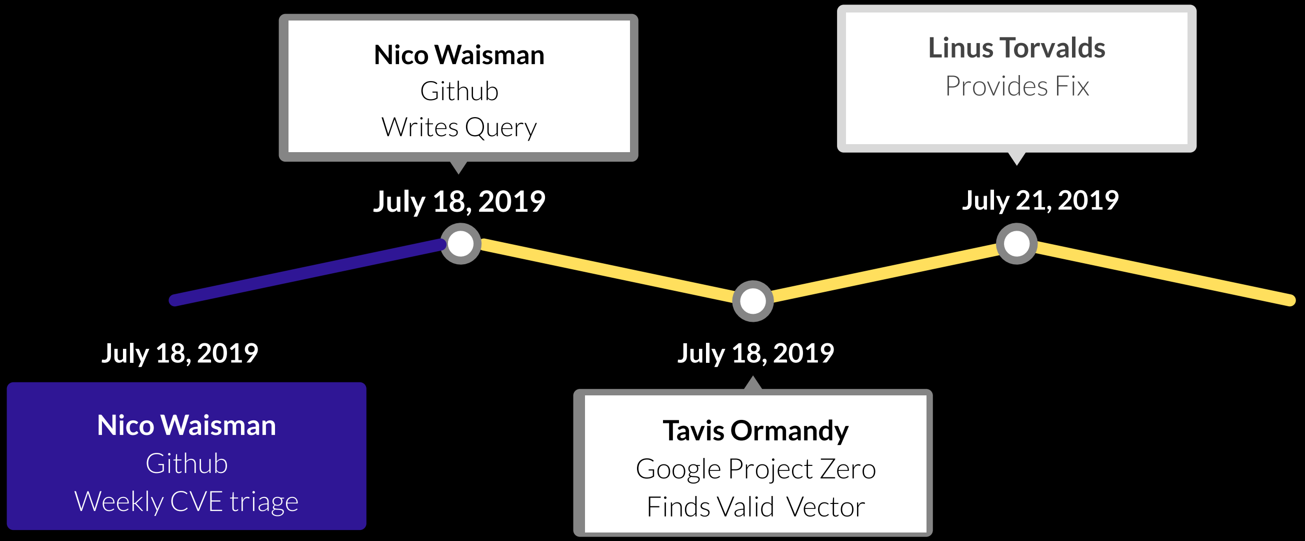
```
from AddExpr a, Variable v, RelationalOperation cmp
where a.getAnOperand() = v.getAnAccess()
  and cmp.getAnOperand() = a
  and cmp.getAnOperand() = v.getAnAccess()
  and forall(Expr op | op = a.getAnOperand() |
      op.getType().getSize() < 4)
  and not a.getExplicitlyConverted().getType().getSize() < 4
select cmp, "Bad overflow check"
```

Sample QL query identifying common mistakes made checking for integer overflows

What is Github “also” doing?

Variant Analysis (CVE triage)

Linux Kernel
Buffer Overflow

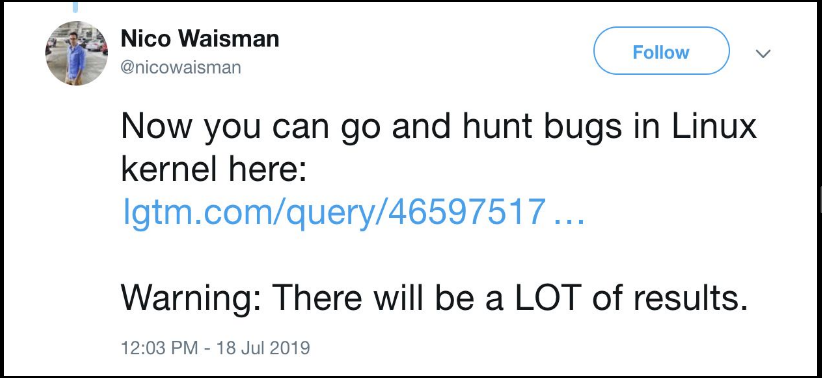


Array write inside loop

We are looking for:

```
while (condition) {

    [...]

    array[x] = y;

    [...]

}
```

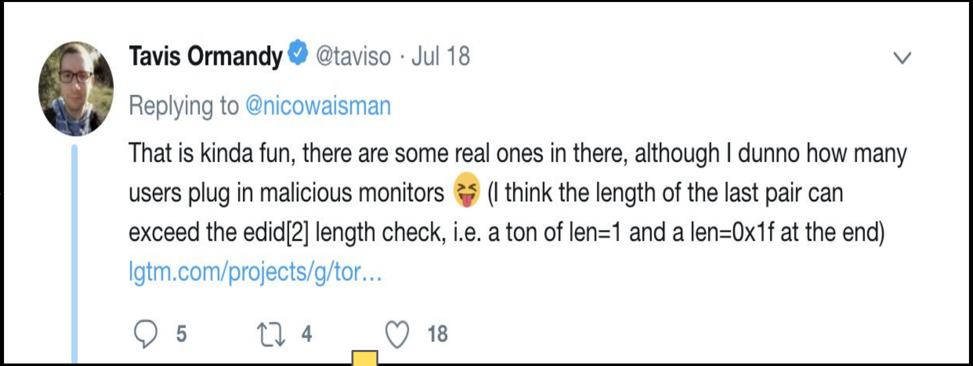
```
import cpp

from Variable buffer, Expr bufferWriteBase, ArrayExpr bufferWrite,
Assignment a, Loop loop
where
      buffer.getType() instanceof ArrayType and
    bufferWriteBase = buffer.getAnAccess() and
    bufferWriteBase = bufferWrite.getArrayBase() and
    bufferWrite = a.getLValue() and
    loop = a.getEnclosingStmt().getParent*()
select a
```


Nico Waisman
@nicowaisman

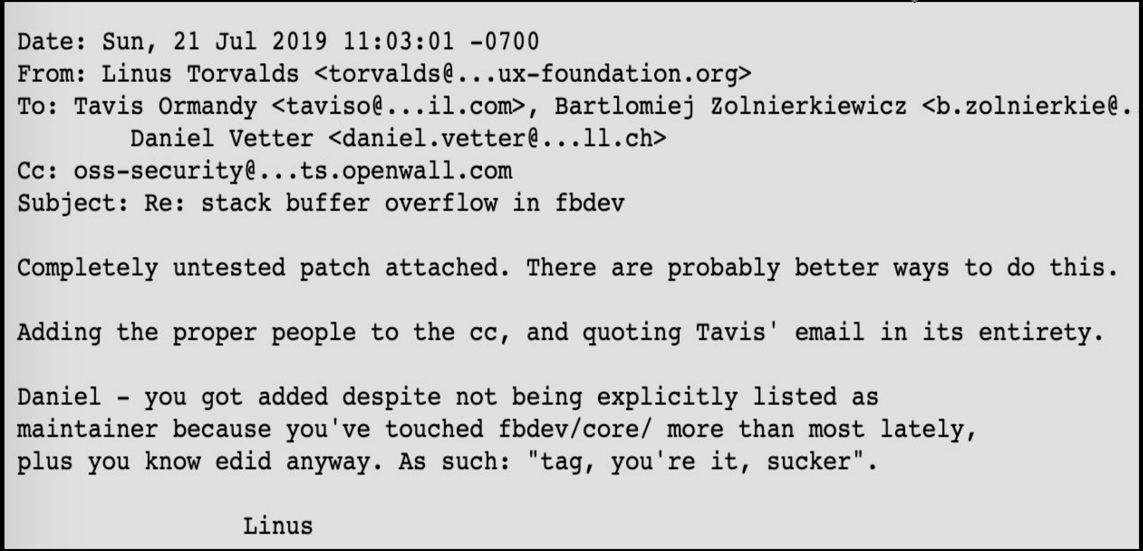
Now you can go and hunt bugs in Linux kernel here: lgtm.com/query/46597517 …

Warning: There will be a LOT of results.

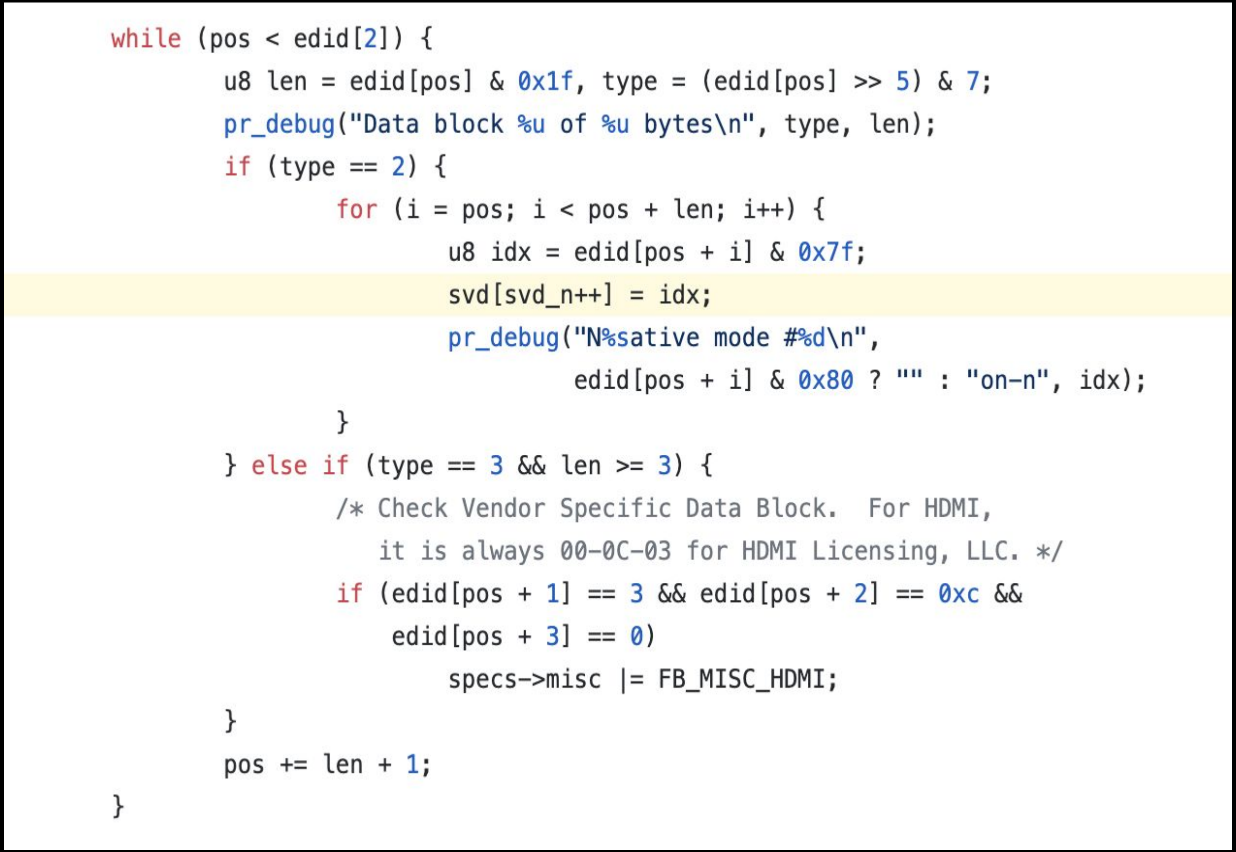
12:03 PM - 18 Jul 2019


Tavis Ormandy @taviso · Jul 18
Replying to @nicowaisman

That is kinda fun, there are some real ones in there, although I dunno how many users plug in malicious monitors 😝 (I think the length of the last pair can exceed the edid[2] length check, i.e. a ton of len=1 and a len=0x1f at the end) lgtm.com/projects/g/tor...


```
Date: Sun, 21 Jul 2019 11:03:01 -0700
From: Linus Torvalds <torvalds@...ux-foundation.org>
To: Tavis Ormandy <taviso@...il.com>, Bartlomiej Zolnierkiewicz <b.zolnierkie@.
        Daniel Vetter <daniel.vetter@...ll.ch>
Cc: oss-security@...ts.openwall.com
Subject: Re: stack buffer overflow in fbdev

Completely untested patch attached. There are probably better ways to do this.

Adding the proper people to the cc, and quoting Tavis' email in its entirety.

Daniel - you got added despite not being explicitly listed as
maintainer because you've touched fbdev/core/ more than most lately,
plus you know edid anyway. As such: "tag, you're it, sucker".

            Linus
```

Linux Kernel, never plug untrusted monitors...

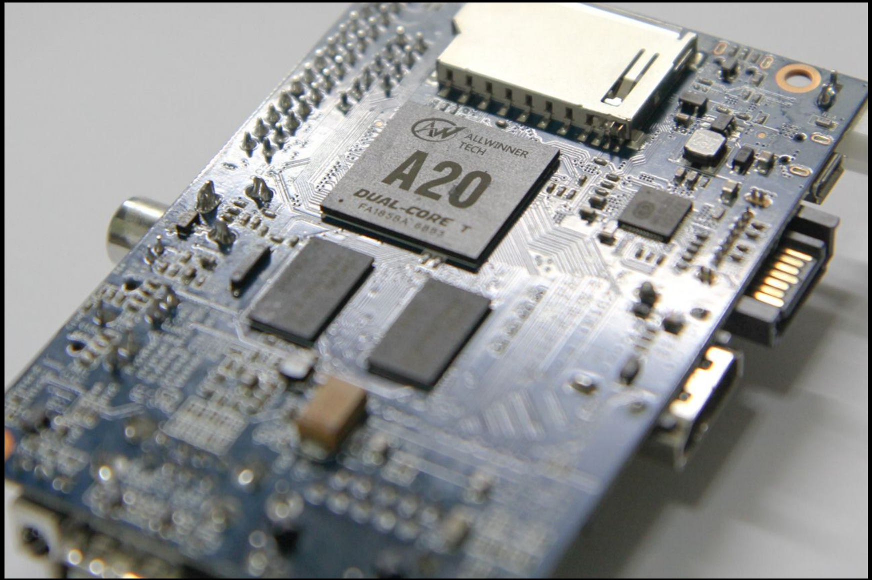


```
while (pos < edid[2]) {
        u8 len = edid[pos] & 0x1f, type = (edid[pos] >> 5) & 7;
        pr_debug("Data block %u of %u bytes\n", type, len);
        if (type == 2) {
                for (i = pos; i < pos + len; i++) {
                        u8 idx = edid[pos + i] & 0x7f;
                        svd[svd_n++] = idx;
                        pr_debug("N%sative mode #%d\n",
                                 edid[pos + i] & 0x80 ? "" : "on-n", idx);
                }
        } else if (type == 3 && len >= 3) {
                /* Check Vendor Specific Data Block.  For HDMI,
                   it is always 00-0C-03 for HDMI Licensing, LLC. */
                if (edid[pos + 1] == 3 && edid[pos + 2] == 0xc &&
                    edid[pos + 3] == 0)
                        specs->misc |= FB_MISC_HDMI;
        }
        pos += len + 1;
}
```

U-Boot: A variant analysis journey

What is U-Boot?

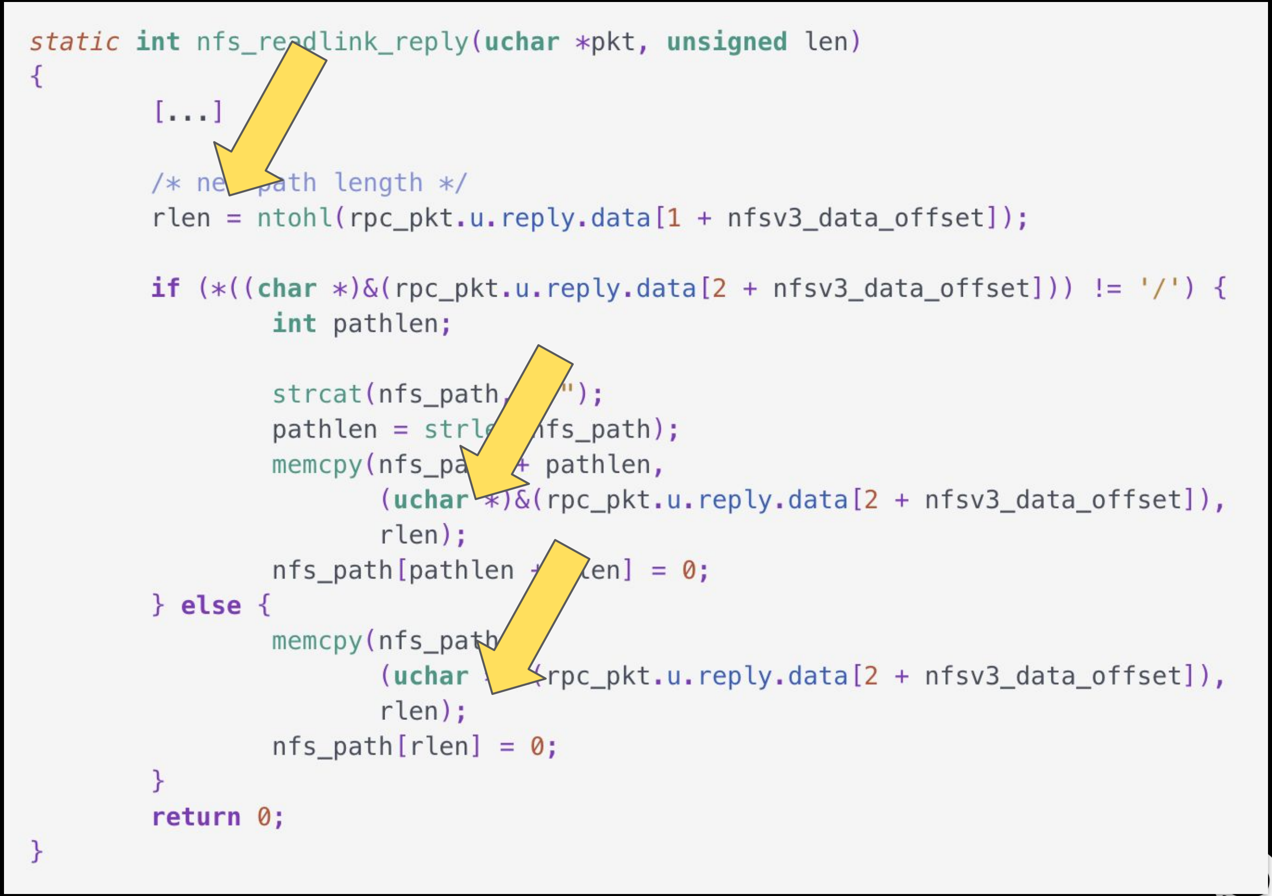
- Open source bootloader
- Used by
  - Kindle devices
  - ChromeOS ARM devices
  - IoT
- Supports verifiable boot
  - Any vulnerability before the signature check is a potential jailbreak
  - Filesystems should then be considered untrusted
  - ……Not to mention any networking :)



Vulnerability counter: 2

Seed Vulnerability:

- 2x memcpy() size argument is attacker controlled
- size comes from the NFS packet
- No size constraints



```
static int nfs_readlink_reply(uchar *pkt, unsigned len)
{
        [...]

        /* new path length */
        rlen = ntohl(rpc_pkt.u.reply.data[1 + nfsv3_data_offset]);

        if (*((char *)&(rpc_pkt.u.reply.data[2 + nfsv3_data_offset])) != '/') {
                int pathlen;

                strcat(nfs_path, "/");
                pathlen = strlen(nfs_path);
                memcpy(nfs_path + pathlen,
                       (uchar *)&(rpc_pkt.u.reply.data[2 + nfsv3_data_offset]),
                       rlen);
                nfs_path[pathlen + rlen] = 0;
        } else {
                memcpy(nfs_path,
                       (uchar *)&(rpc_pkt.u.reply.data[2 + nfsv3_data_offset]),
                       rlen);
                nfs_path[rlen] = 0;
        }
        return 0;
}
```

Vulnerability counter: 2

Query initial steps: find all memcpy callers

- **596** instances. Find usage of rpc_pkt.u.**reply**
- **191** instances

```
import cpp


from FunctionCall call
where call.getTarget().getName() = "memcpy"
select call
```

```
import cpp


from FieldAccess access, Field f
where f = access.getTarget() and f.hasName("reply")
select access
```

Given a (problem-specific) set of sources and sinks, is there a path in the data flow graph from some source to some sink?

Vulnerability counter: 2

Query refining process

- Data Flow analysis
- Source: nfs data packet read from the socket
- Sink: memcpy()

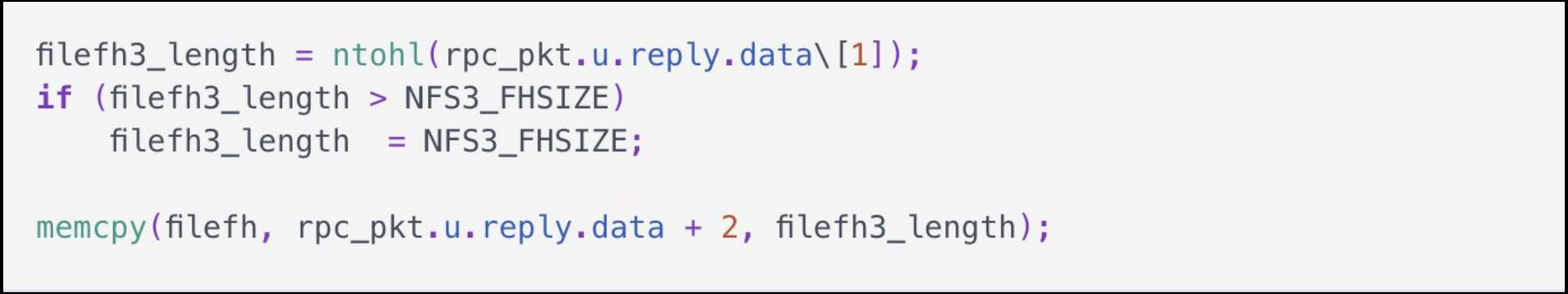
Results:

- 4 instances
- We found the 2 original seed vulnerabilities

```
import cpp
import semmle.code.cpp.dataflow.TaintTracking

class NetworkToMemFuncLength extends TaintTracking::Configuration {
  NetworkToMemFuncLength() { this = "NetworkToMemFuncLength" }

  override predicate isSource(DataFlow::Node source) {
    exists (FieldAccess access, Field f |
      source.asExpr() = access and f = access.getTarget() and
f.hasName("reply"))
  }

  override predicate isSink(DataFlow::Node sink) {
    exists (FunctionCall fc, Expr argument |
      sink.asExpr() = argument and (argument = fc.getArgument(2)
and fc.getTarget().hasQualifiedName("memcpy"))
            )
  }
}

from Expr socket_buffer, Expr sizeArg, NetworkToMemFuncLength
config
where config.hasFlow(DataFlow::exprNode(socket_buffer),
DataFlow::exprNode(sizeArg))
select sizeArg
```

Vulnerability counter: 3

Findings triage



```
filefh3_length = ntohl(rpc_pkt.u.reply.data\[1]);
if (filefh3_length > NFS3_FHSIZE)
    filefh3_length  = NFS3_FHSIZE;

memcpy(filefh, rpc_pkt.u.reply.data + 2, filefh3_length);
```

Looks like they are doing the correct by checking the length

Until you discover that filefh3_length is a signed integer!!!

Vulnerability Counter += 1

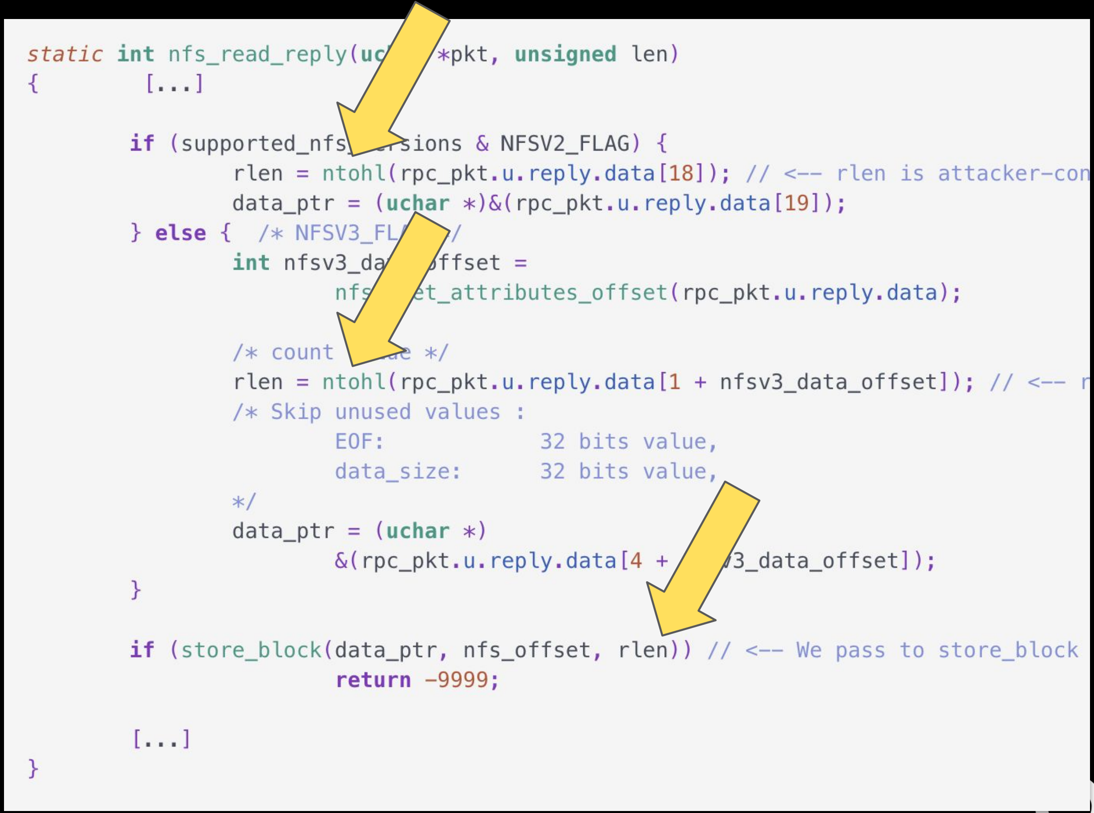
Vulnerability counter: 5

Findings triage

Same story:

- rlen comes directly from the NFS packet
- rlen is consumed as the size for the memcpy() operation
- No checks!

memcpy() happens inside store_block() in two different locations



```
static int nfs_read_reply(uc    *pkt, unsigned len)
{         [...]

        if (supported_nfs    sions & NFSV2_FLAG) {
                rlen = ntohl(rpc_pkt.u.reply.data[18]); // <-- rlen is attacker-con
                data_ptr = (uchar *)&(rpc_pkt.u.reply.data[19]);
        } else {  /* NFSV3_FL    /
                int nfsv3_da    ffset =
                        nfs    et_attributes_offset(rpc_pkt.u.reply.data);

                /* count     ue */
                rlen = ntohl(rpc_pkt.u.reply.data[1 + nfsv3_data_offset]); // <-- r
                /* Skip unused values :
                        EOF:            32 bits value,
                        data_size:      32 bits value,
                */
                data_ptr = (uchar *)
                        &(rpc_pkt.u.reply.data[4 +    v3_data_offset]);
        }

        if (store_block(data_ptr, nfs_offset, rlen)) // <-- We pass to store_block
                return -9999;

        [...]
}
```

Let's think bigger beyond NFS:

- Data Flow analysis
- Source: ntohl(), ntohs() and friends
- Sink: memcpy()

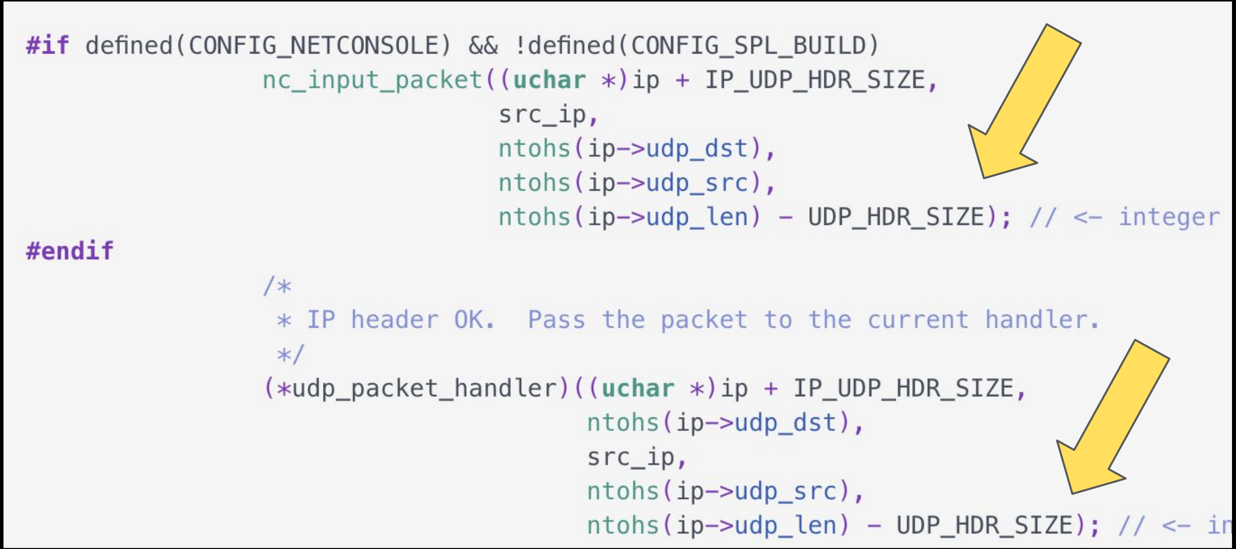
Results:

- 8 instances
- Finds the all previous vulnerabilities

```
import cpp
import semmle.code.cpp.dataflow.TaintTracking

class NetworkByteOrderTranslation extends Expr {
  NetworkByteOrderTranslation() {
    this = any(MacroInvocation mi |
mi.getOutermostMacroAccess().getMacroName().regexpMatch("(?i)(^|.*_)nt
oh(l|ll|s)")
      ).getExpr()
  }
}

[...]

  override predicate isSource(DataFlow::Node source) {
    exists (FieldAccess access, Field f |
      source.asExpr() instanceof NetworkByteOrderTranslation
  }

[...]

from Expr socket_buffer, Expr sizeArg, NetworkToMemFuncLength config
where config.hasFlow(DataFlow::exprNode(socket_buffer),
DataFlow::exprNode(sizeArg))
select sizeArg
```

Vulnerability counter: 7

Findings triage



```
#if defined(CONFIG_NETCONSOLE) && !defined(CONFIG_SPL_BUILD)
                nc_input_packet((uchar *)ip + IP_UDP_HDR_SIZE,
                                src_ip,
                                ntohs(ip->udp_dst),
                                ntohs(ip->udp_src),
                                ntohs(ip->udp_len) - UDP_HDR_SIZE); // <- integer
#endif
                /*
                 * IP header OK.  Pass the packet to the current handler.
                 */
                (*udp_packet_handler)((uchar *)ip + IP_UDP_HDR_SIZE,
                                      ntohs(ip->udp_dst),
                                      src_ip,
                                      ntohs(ip->udp_src),
                                      ntohs(ip->udp_len) - UDP_HDR_SIZE); // <- in
```

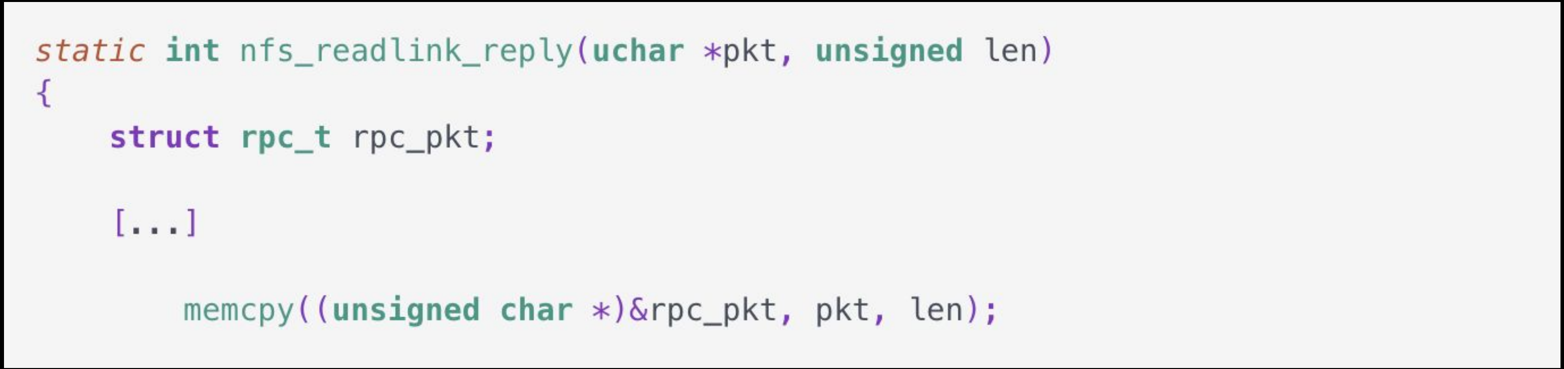
TCP/IP stack

2 integer underflows with no bounds checking later ending up in memcpy()

Vulnerability Counter += 2

Vulnerability counter: 12

Other vulnerabilities



```
static int nfs_readlink_reply(uchar *pkt, unsigned len)
{
    struct rpc_t rpc_pkt;

    [...]

        memcpy((unsigned char *)&rpc_pkt, pkt, len);
```

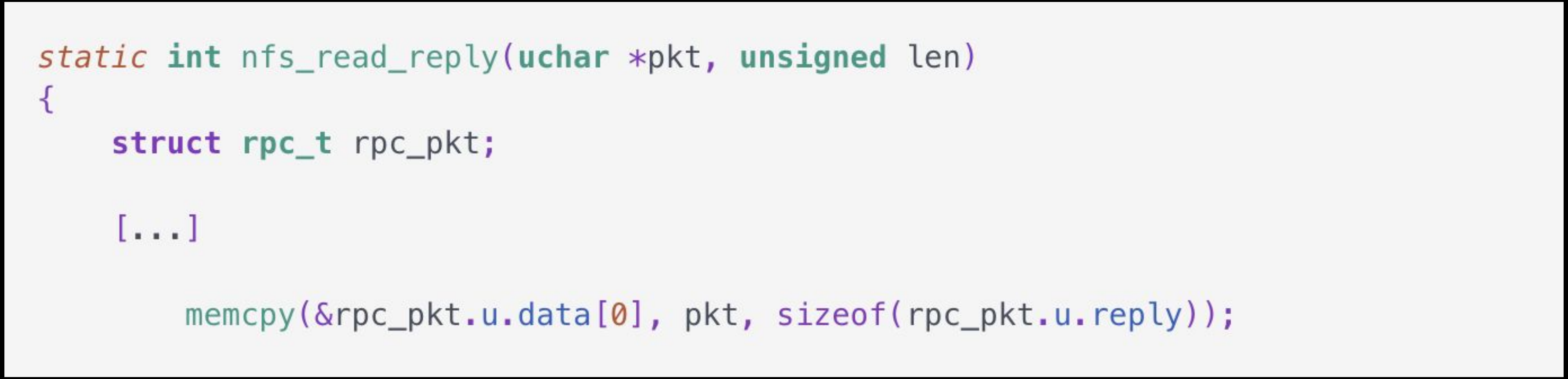
Plain stack overflow

Also happens in 4 other different functions

Vulnerability Counter += 5

Vulnerability counter: 13

Other vulnerabilities



```
static int nfs_read_reply(uchar *pkt, unsigned len)
{
    struct rpc_t rpc_pkt;

    [...]

        memcpy(&rpc_pkt.u.data[0], pkt, sizeof(rpc_pkt.u.reply));
```

Clearly, someone is trying to prevent the overflow

But, forgot to check the lower bound… would be a READ OOB

Vulnerability Counter += 1

CVE-2019-14192, CVE-2019-14193, CVE-2019-14194, CVE-2019-14195, CVE-2019-14196, CVE-2019-14197, CVE-2019-14198, CVE-2019-14199, CVE-2019-14200, CVE-2019-14201, CVE-2019-14202, CVE-2019-14203 and CVE-2019-14204

Wifi drivers and trusting lengths

Linux Kernel
Buffer Overflow

August 28, 2019

huangwen
ADLab
CVE-2019-1481[4-6]

August 29, 2019

Nico Waisman
Github
Writes Query

August 30, 2019

Nico Waisman
Github
Report 2 bugs

Sep 20, 2019

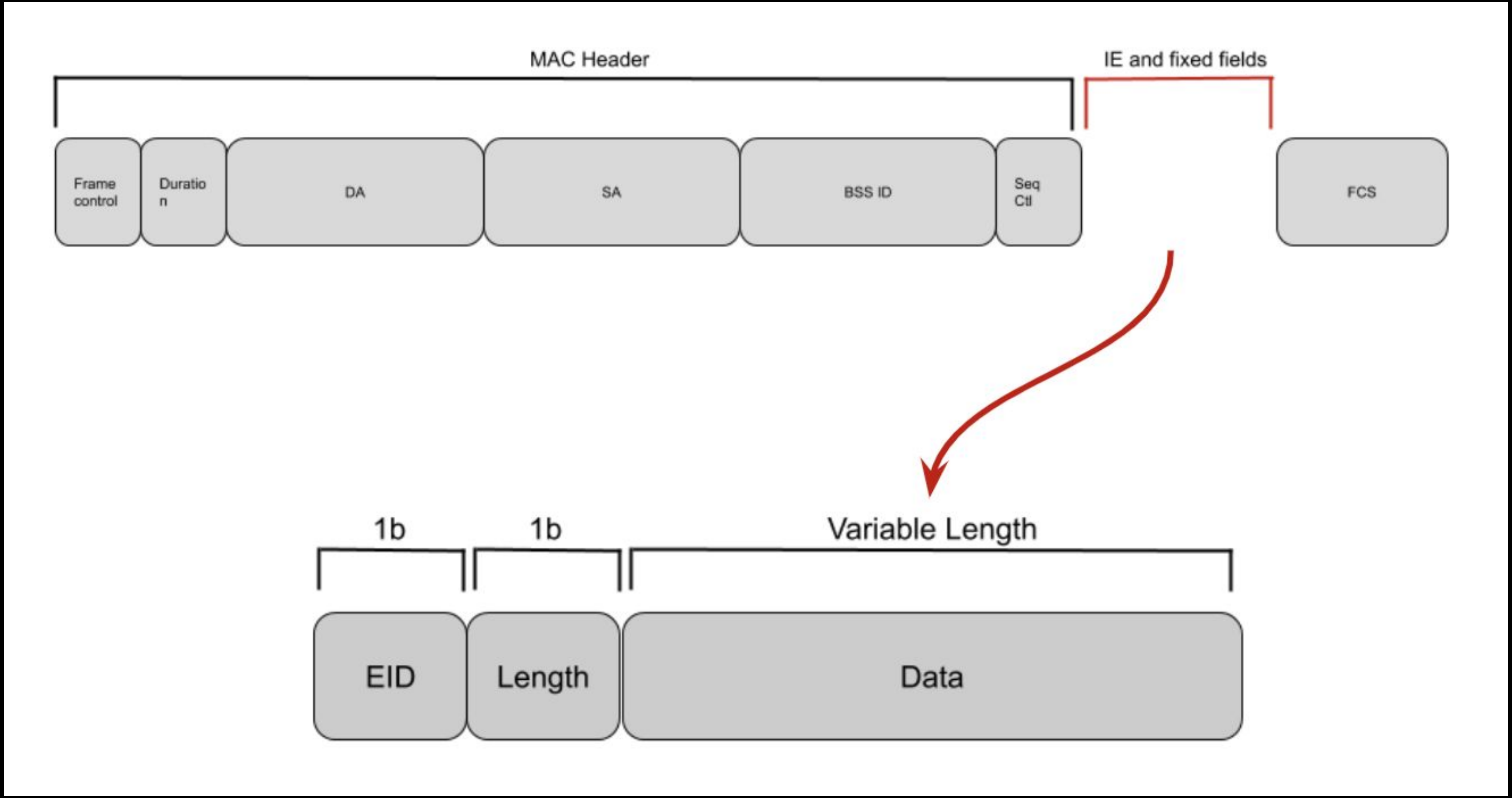
Linux Team
Provides Fix for
nl80211/cfg80211

Oct 4, 2019

Linux Team
Provides Fix for
mac80211/wext-sme

Vulnerability counter: 0

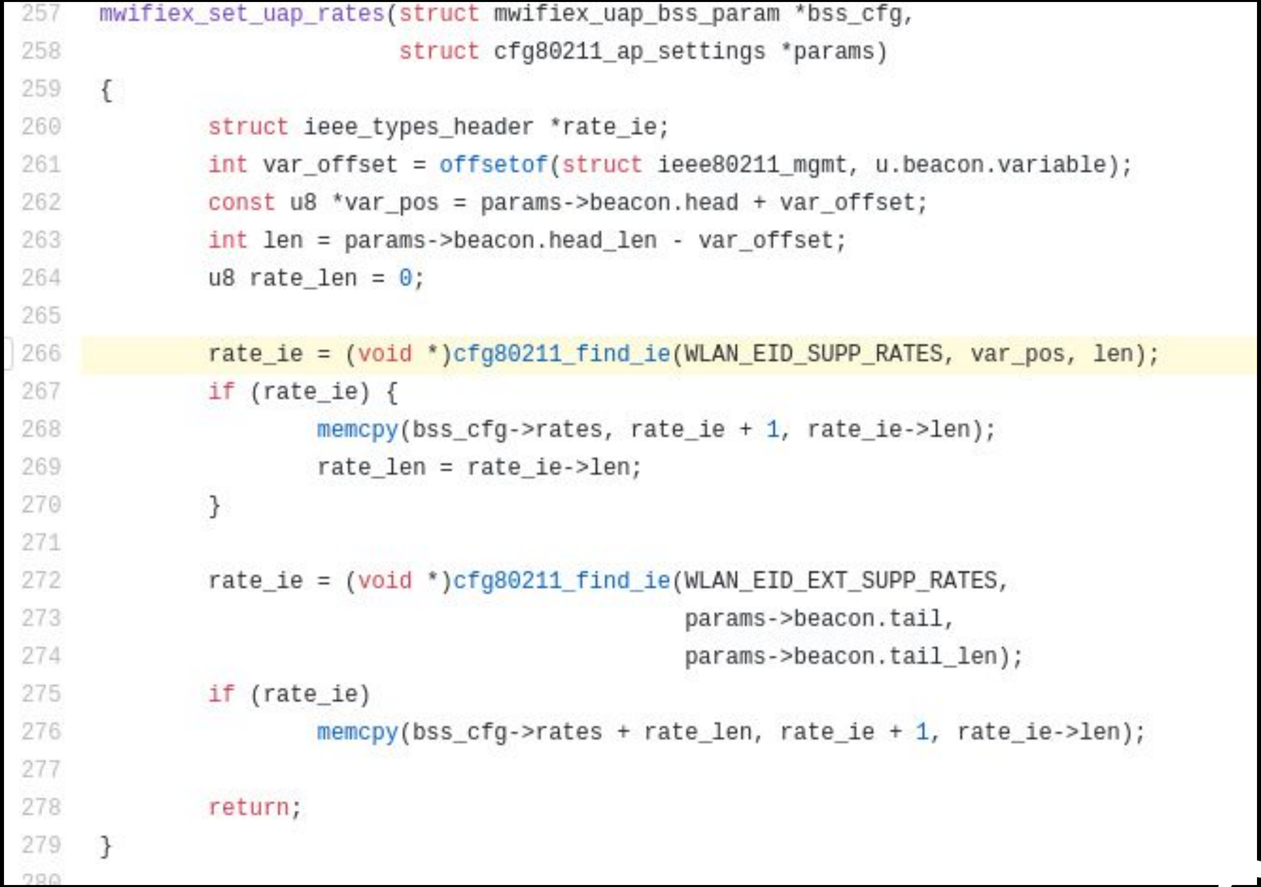
Crash course on 802.11 framing



Vulnerability counter: 0

CVE-2019-1481[4-6]

Found by huangwen of ADLab of Venustech



```
mwifiex_set_uap_rates(struct mwifiex_uap_bss_param *bss_cfg,
                      struct cfg80211_ap_settings *params)
{
        struct ieee_types_header *rate_ie;
        int var_offset = offsetof(struct ieee80211_mgmt, u.beacon.variable);
        const u8 *var_pos = params->beacon.head + var_offset;
        int len = params->beacon.head_len - var_offset;
        u8 rate_len = 0;

        rate_ie = (void *)cfg80211_find_ie(WLAN_EID_SUPP_RATES, var_pos, len);
        if (rate_ie) {
                memcpy(bss_cfg->rates, rate_ie + 1, rate_ie->len);
                rate_len = rate_ie->len;
        }

        rate_ie = (void *)cfg80211_find_ie(WLAN_EID_EXT_SUPP_RATES,
                                           params->beacon.tail,
                                           params->beacon.tail_len);
        if (rate_ie)
                memcpy(bss_cfg->rates + rate_len, rate_ie + 1, rate_ie->len);

        return;
}
```

Vulnerability counter: 0

We start with cfg80211_find_ie:

- Data Flow analysis
- Source: cfg80211_find_ie
- Sink: memcpy() size

Results:

- 13 instances
- No bugs. Code looks buggy, but there were sanitized
- What other function deal with IE?

```
class GetIE extends TaintTracking::Configuration {
  GetIE() { this="cfg80211_find_ie" }

  override predicate isSource(DataFlow::Node source) {
     exists( Function sl |
      source.asExpr().(FunctionCall).getTarget() = sl
      and
      sl.hasQualifiedName("cfg80211_find_ie")

     )
  }
  override predicate isSink(DataFlow::Node sink) {
     exists(FunctionCall fc |
      sink.asExpr() = fc.getArgument(2)
      and
      fc.getTarget().hasQualifiedName("memcpy")
     )
  }
}
```

Vulnerability counter: 0

A new IE based function:

- Data Flow analysis
- Source: ieee80211_bss_get_ie
- Sink: memcpy() size

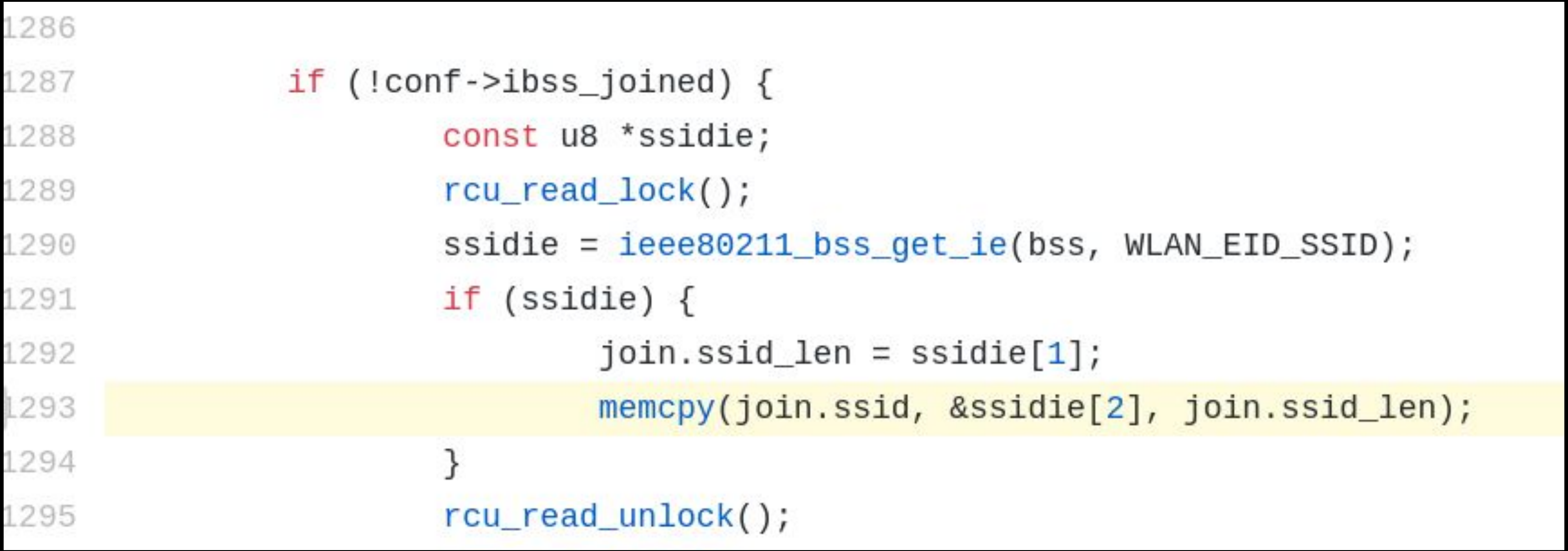
Results:

- 10 instances
- Most of the match looks promising, but a big chunk where sanitized.

```
class GetIE extends TaintTracking::Configuration {
  GetIE() { this="ieee80211_bss_get_ie" }

  override predicate isSource(DataFlow::Node source) {
     exists( Function sl |
      source.asExpr().(FunctionCall).getTarget() = sl
      and
      sl.hasQualifiedName("ieee80211_bss_get_ie")

     )
  }
  override predicate isSink(DataFlow::Node sink) {
     exists(FunctionCall fc |
      sink.asExpr() = fc.getArgument(2)
      and
      fc.getTarget().hasQualifiedName("memcpy")
     )
  }
}
```

CVE-2019-16746

Vulnerability counter: 1

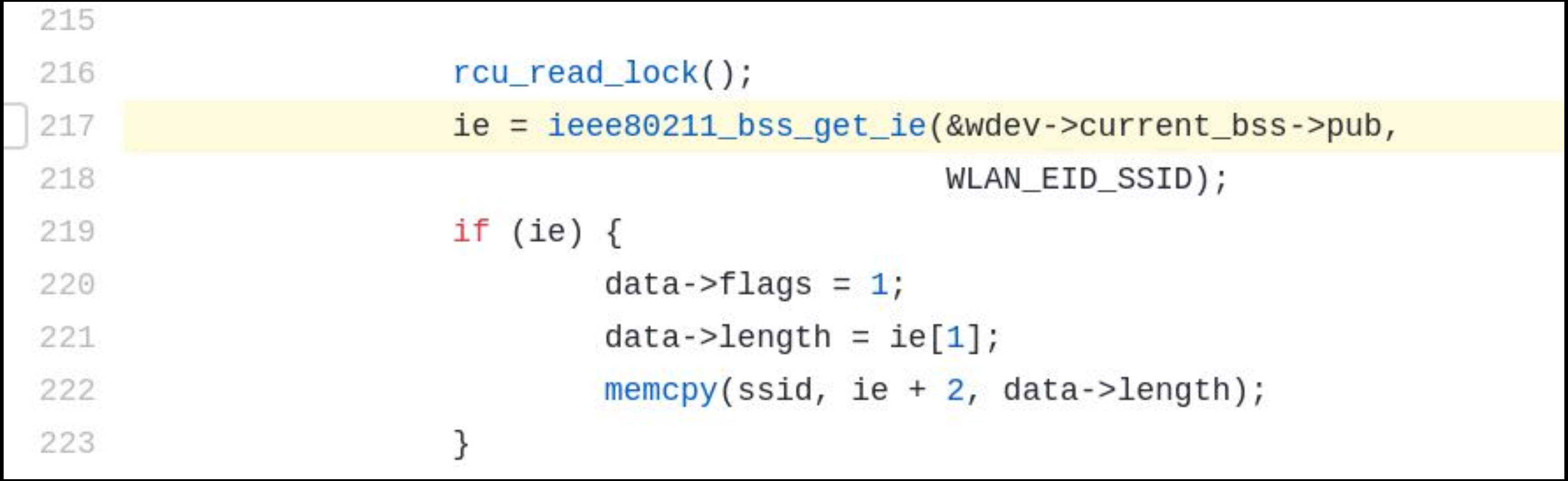


```
	if (!conf->ibss_joined) {
		const u8 *ssidie;
		rcu_read_lock();
		ssidie = ieee80211_bss_get_ie(bss, WLAN_EID_SSID);
		if (ssidie) {
			join.ssid_len = ssidie[1];
			memcpy(join.ssid, &ssidie[2], join.ssid_len);
		}
		rcu_read_unlock();
```

- cw1200 wireless driver
- Remotely exploitable through a beacon

CVE-2019-17133

Vulnerability counter: 2



```
	rcu_read_lock();
	ie = ieee80211_bss_get_ie(&wdev->current_bss->pub,
				  WLAN_EID_SSID);
	if (ie) {
		data->flags = 1;
		data->length = ie[1];
		memcpy(ssid, ie + 2, data->length);
	}
```

- cfg80211 wext compat for managed mode.
- Remotely exploitable through a beacon

Q & A

Thanks!
@fjserna