Are all BSDs created equally?
A survey of BSD kernel vulnerabilities.

Ilja van Sprundel <ivansprundel@ioactive.com>
Who Am I

• Ilja van Sprundel
• ivansprundel@ioactive.com
• Director of Penetration Testing at IOActive
• Pen test
• Code review
• Break stuff for fun and profit 😊
Outline/Agenda

• Intro
• Data!
  • vulnerabilities over the years
• Test by audit
  • Common attack surface
  • Somewhat less common attack surface
• Some results / conclusions
What is this talk about?

• BSD kernel vulnerabilities
  • Comparison
  • Between different BSD flavors

• Audience
  • Low level security enthusiasts
  • UNIX/BSD geeks
    • I suspect Linux folks might enjoy this too 😊
  • Curious people that like to poke around in OS internals

• Knowledge
  • Some basic knowledge of UNIX / BSD internals
Standing on the shoulders of giants

- Previous interesting BSD kernel security research by:
  - Silvio
  - the noir
  - Esa Etelävuori
  - Patroklos (argp) Argyroudis
  - Christer Oberg
  - Joel Erikkson
  - Clement Lecigne
Re: Theo gave an interview to Forbes Mag. about Linux

Theo de Raadt  Fri, 17 Jun 2005 09:30:05 -0700

> On Fri, Jun 17, 2005 at 04:48:31PM +0200, J. Lievisse Adriaanse wrote:

... 

If the Linux people actually cared about Quality, as we do, they would not have had as many localhost kernel security holes in the last year.

How many is it... 20 so far?
Really? Got Data?

• Somehow that statement has always been stuck in my head
• Is it true?
• Can we look at some data?
## Linux Kernel : Vulnerability Statistics

### Vulnerability Trends Over Time

<table>
<thead>
<tr>
<th>Year</th>
<th># of Vulnerabilities</th>
<th>DoS</th>
<th>Code Execution</th>
<th>Overflow</th>
<th>Memory Corruption</th>
<th>SQL Injection</th>
<th>XSS</th>
<th>Directory Traversal</th>
<th>HTTP Response Splitting</th>
<th>Bypass something</th>
<th>Gain Information</th>
<th>Gain Privileges</th>
<th>CSRF</th>
<th>File Inclusion</th>
<th># of exploits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>19</td>
<td>7</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>23</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>15</td>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>19</td>
<td>8</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2004</td>
<td>51</td>
<td>20</td>
<td>5</td>
<td>12</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>2005</td>
<td>133</td>
<td>90</td>
<td>19</td>
<td>19</td>
<td></td>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>2006</td>
<td>90</td>
<td>81</td>
<td>5</td>
<td>7</td>
<td></td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2007</td>
<td>63</td>
<td>41</td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>2008</td>
<td>70</td>
<td>44</td>
<td>3</td>
<td>17</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2009</td>
<td>105</td>
<td>66</td>
<td>2</td>
<td>22</td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>2010</td>
<td>124</td>
<td>87</td>
<td>3</td>
<td>18</td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>2011</td>
<td>83</td>
<td>52</td>
<td>1</td>
<td>21</td>
<td></td>
<td>10</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>2012</td>
<td>115</td>
<td>83</td>
<td>4</td>
<td>25</td>
<td></td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>2013</td>
<td>189</td>
<td>101</td>
<td>6</td>
<td>41</td>
<td></td>
<td>13</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>2014</td>
<td>133</td>
<td>89</td>
<td>8</td>
<td>21</td>
<td></td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>2015</td>
<td>86</td>
<td>55</td>
<td>6</td>
<td>15</td>
<td></td>
<td>4</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2016</td>
<td>217</td>
<td>153</td>
<td>8</td>
<td>38</td>
<td></td>
<td>18</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>2017</td>
<td>346</td>
<td>80</td>
<td>165</td>
<td>29</td>
<td></td>
<td>14</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74</td>
</tr>
<tr>
<td>Total</td>
<td>1886</td>
<td>1649</td>
<td>234</td>
<td>297</td>
<td></td>
<td>105</td>
<td>2</td>
<td></td>
<td></td>
<td>102</td>
<td>317</td>
<td>243</td>
<td></td>
<td></td>
<td>33</td>
</tr>
</tbody>
</table>

% Of All: 55.1 12.4 15.7 5.6 0.0 0.0 0.1 0.0 5.4 16.8 12.9 0.8 0.0
Data!

• Goes from current back to 1999 for Linux kernel vulnerabilities
• Cvedetails.com doesn’t seem to provide data for OBSD/NBSD/FBSD
• Manually grab it from
  • https://www.freebsd.org/security/advisories.html
  • http://netbsd.org/support/security/advisory.html
  • https://www.openbsd.org/errata*.html
BSD kernel vulnerabilities over the years

- Looking at these numbers, that was an astute observation by Theo.
  - 20 was a very low estimate
- But are these numbers on equal footing?
- Many eyeballs?
  - Yea, yea, I know .... But is there some truth to it in this case?

<table>
<thead>
<tr>
<th>Year</th>
<th>FreeBSD</th>
<th>NetBSD</th>
<th>OpenBSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>3</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>2000</td>
<td>8</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>2001</td>
<td>6</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>2002</td>
<td>11</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>2003</td>
<td>7</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>2004</td>
<td>8</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>2005</td>
<td>11</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>2006</td>
<td>9</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2007</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2008</td>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2009</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2012</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2013</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2014</td>
<td>7</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2015</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2016</td>
<td>12</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>2017</td>
<td>1</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>Total</td>
<td>118</td>
<td>96</td>
<td>163</td>
</tr>
</tbody>
</table>
Test by audit!

- Silvio Cesare did some interesting work in ~2002 that gives some answers
- [https://www.blackhat.com/presentations/bh-usa-03/bh-usa-03-cesare.pdf](https://www.blackhat.com/presentations/bh-usa-03/bh-usa-03-cesare.pdf)
- His results seem to indicate there isn’t really that much of a quality difference. However:
  - that was well over a decade ago.
    - Have things changed?
  - Time spend on the BSDs was only a couple of days compared to Linux
    - If more time would’ve been spend, would more bugs have been found?
  - bugs are mostly int overflows and info leaks
    - Other kinds of issues that can ‘easily’ be found?
Test by Audit redux.

- Spend April-May-June auditing BSD source code.
- Asked myself, “where would the bugs be?”
- Attack surface
  - Very common
    - Syscalls
    - TCP/IP stack
  - Somewhat less common (in ascending order, more or less)
    - Drivers (ioctl interface)
    - compat code
    - Trap handlers
    - Filesystems
    - Other networking (BT, wifi, IrDA)
Syscalls
Attack surface entrypoint

- The obvious attack surface
- Syscalls are how userland gets anything done from kernel
- Hundreds of them
  - FreeBSD: ~550
  - OpenBSD: ~330
  - NetBSD: ~480
- Assumption: given that they’re obvious, and well tested, less likely to contain security bugs
```c
int sys_sendsyslog(struct proc *p, void *v, register_t *retval)
{
    struct sys_sendsyslog_args /*
        {syscallarg(const void *) buf;
        syscallarg(size_t) nbyte;
        syscallarg(int) flags;
    }*/ uap = v;
    int error;
    static int dropped_count, orig_error;
    ...
    error = dosendsyslog(p, SCARG(uap, buf),
                        SCARG(uap, nbyte),
                        SCARG(uap, flags), UIO_USERSPACE);
    ...
    return (error);
}

int dosendsyslog(struct proc *p, const char *buf, size_t nbyte, int flags,
                  enum uio_seg sflg)
{
    ...
    struct iovec aiov;
    struct uio auio;
    size_t i, len;
    ...
    aiov.iov_base = (char *) buf;
    aiov.iov_len = nbyte; // user controlled size_t. never capped anywhere
    auio.uio_resid = aiov.iov_len;
    ...
    len = auio.uio_resid;
    ...
    if (fp) {
        ...
    } else if (consy) {
        ...
    } else {
        ...
    }
}
```
Sample bug

• sendsyslog system call
• OpenBSD 6.1
  • Been there since OpenBSD 6.0
• Unbound length passed to malloc() from userland
• Will trigger a kernel panic
int sys_kldstat(struct thread *td, struct kldstat_args *uap)
{
    struct kld_file_stat stat;
    int error, version;

    /*
     * Check the version of the user's structure.
     */
    if ((error = copyin(&uap->stat->version, &version, sizeof(version)))
        != 0)
        return (error);
    if (version != sizeof(struct kld_file_stat_1) &&
        version != sizeof(struct kld_file_stat))
        return (EINVAL);

    error = kern_kldstat(td, uap->fileid, &stat); /* doesn't full init stat struct
    if (error != 0)
        return (error);
    return (copyout(&stat, uap->stat, version)); /* uninit stat struct copied to user
}
Sample bug 2

• kldstat system call
• FreeBSD 11.0
  • Been there for almost 10 years (Modified Mon Oct 22 04:12:57 2007 UTC (9 years, 9 months ago))
• Doesn’t fully initialize a structure. Sends it back to userland
• infoleak

• Previous assumption is not true: bugs in syscalls do occur with some frequency
  • Especially newly added syscalls
Attack surface entrypoint

- TCP/IP stack
  - IPv4/6
  - UDP/TCP/ICMP
  - IPsec
  - ...
- Obvious and well known attack surface
- Has been around forever
- Assumption: well tested and less likely to find bugs there
static void pppoe_dispatch_disc_pkt(struct mbuf *m, int off)
{
    ...
    int noff, err, errortag;
    u_int16_t *max_payload;
    u_int16_t *tag, len;
    ...
    while (off + sizeof(*pt) <= m->m_pkthdr.len) {
        n = m_pulldown(m, off, sizeof(*pt), &noff);
        ...
        pt = (struct pppoeitag *)(mtod(n, caddr_t) + noff);
        tag = ntohs(pt->tag);
        len = ntohs(pt->len);
        ...
        switch (tag) {
        ...
        case PPPOE_TAG_SNAME_ERR:
            err_msg = "SERVICE NAME ERROR";
            errortag = 1;
            break;
        ...
        }
        if (err_msg) {
            log(LOG_INFO, "%s: %s: ", devname, err_msg);
            if (errortag && len) {
                n = m_pulldown(m, off, len,
                               &noff);  \<-- if m_pulldown() fails, it will mfreem(m)
                if (n) {
                    u_int8_t *et = mtod(n, caddr_t) + noff;
                    while (len--)
                        addlog("%c", *et++);
                   }  \<-- should have else case that sets m to NULL
            }
            addlog("\n");
            goto done;  \<-- will end up mfreem(m) again
        }
        ...
        off += len;
    }
    ...
    done:
    m_freem(m); \<-- possible double free!
}
Sample bug

• `pppoe_dispatch_disc_pkt()`
• Double free when parsing packet

• Affects OpenBSD 6.1
  • Been there since 2004
  • Fixed in NetBSD a couple of months ago

• Previous assumption is not [entirely] true: bugs in TCP/IP stack do occur with some frequency
  • newer code
  • *mbuf handling is complicated and error prone*
Drivers
Attack surface entrypoint

- Lots and lots of drivers
- For all sorts of things
- UNIX: everything is a file
  - Most expose entrypoints in /dev
- File operations
  - Open
  - ioctl
  - Read
  - Write
  - Close
  - ...
- ioctl is where most of the attack surface is!
int cryptof_ioctl(struct file *fp, u_long cmd, void *data) {
    ...
    switch (cmd) {
        ...
        mutex_enter(&crypto_mtx);
        fcr->mtime = fcr->atime;
        mutex_exit(&crypto_mtx);
        mkop = (struct crypt_mkop *)data;
        knop = kmalloc((mkop->count * sizeof(struct crypt_n_kop)),
                        KM_SLEEP);
        error = copyin(mkop->reqs, knop,
                        (mkop->count * sizeof(struct crypt_n_kop)));
        if (!error) {
            error = cryptodev_mkey(fcr, knop, mkop->count);
            if (!error)
                error = copyout(knop, mkop->reqs,
                                (mkop->count * sizeof(struct crypt_n_kop)));
        }
        kmem_free(knop, (mkop->count * sizeof(struct crypt_n_kop)));
        break;
    }
}
Sample bug

- Crypto device CIOCNFKEYM ioctl
- NetBSD 7.1
  - Been there since NetBSD 4.0.1? Thu Apr 10 22:48:42 2008
- Classic integer overflow → memory corruption
static int
ksyms_open(struct cdev *dev, int flags, int fmt __unused, struct thread *td)
{
    ...
    struct ksyms_softc *sc;
    ...
    sc = (struct ksyms_softc *)malloc(sizeof(*sc), M_KSYMS, M_NOWAIT|M_ZERO);
    ...
    sc->sc_proc = td->td_proc;
    sc->sc_pmap = &td->td_proc->p_vmspace->vm_pmap;
    
    /*
     * XXX mmap() will actually map the symbol table into the process
     * address space again.
     */
    if (offset > round_page(sc->sc_usize) ||
        (*paddr = pmap_extract(sc->sc_pmap, /* can be expired pointer! */
        (vm_offset_t)sc->sc_uaddr + offset)) == 0)
        return (-1);
    
    return (0);
}

static int
ksyms_mmap(struct cdev *dev, vm_ooffset_t offset, vm_paddr_t *paddr,
        int prot __unused, vm_memattr_t *memattr __unused)
{
    struct ksyms_softc *sc;
    int error;
    
    error = devfs_get_cdevpriv((void **)&sc);
    if (error)
        return (error);
    
    /* XXX mmap() will actually map the symbol table into the process
     * address space again.
     */
    if (offset > round_page(sc->sc_usize) ||
        (*paddr = pmap_extract(sc->sc_pmap, /* can be expired pointer! */
        (vm_offset_t)sc->sc_uaddr + offset)) == 0)
        return (-1);
    
    return (0);
Sample bug 2

- Ksyms device
- FreeBSD 11
  - Been there since FreeBSD 8.0 *Tue May 26 21:39:09 2009*
- Expired pointer
  - open() callback saves pointer to pmap
  - mmap() callback uses saved pointer in private fd/device storage
  - So how is this a problem?
    - What if we hand fd off to another process (e.g. send over socket or fork/execve)
    - And then we exit
    - If other process now does mmap, it will be using an expired pmap!

Unread portion of the kernel message buffer:

```
Fatal trap 12: page fault while in kernel mode
cpuid = 0; apic id = 0
fault virtual address = 0xfffffffff8000e80
fault code = supervisor read data, page not present
instruction pointer = 0x20:0xfffffffff80f8f2e
frame pointer = 0x20:0xfffffffff80da5c0
code segment = base 0x0, limit 0xffffffff, type 0x1
  - DPL 0, pres 1, long 1, def32 0, gran 1
processor eflags = interrupt enabled, resume, IOMAP = 0
current process = 62550 (ksyms_serv)
trap number = 12
panic: page fault
cpuid = 0
KDB: stack backtrace:
#0 0xfffffffff80b2a077 at kdb_backtrace+0x67
#1 0xfffffffff80b293e2 at vpanic+0x182
#2 0xfffffffff80b29b33 at panic+0x43
#3 0xfffffffff80fda0d at trap_fatal+0x351
#4 0xfffffffff80fda23 at trap_ptault+0x183
#5 0xfffffffff80fda4cc at trap+0x26c
#6 0xfffffffff8104141 at calltrap+0x8
#7 0xfffffffff8221d00c at ksysms_map+0x4c
#8 0xfffffffff80e00cf5 at old_dev_pager_ctor+0x65
#9 0xfffffffff80e0288 at dev_pager_alloc+0x865
#10 0xfffffffff80dd8d8 at dev_pager_alloc+0x228
#11 0xfffffffff80e214e6 at vm_mmap dev+0x96
#12 0xfffffffff80e967e5 at devfs_mmap fd+0x135
#13 0xfffffffff80e9f86 at sys_mmap+0x4c
#14 0xfffffffff80e188e at amd64_syscall+0x4ce
#15 0xfffffffff80e8442 at Xfast_syscall+0xfb
Uptime: 92d6m10s
Dumping stack 253 out of 991 MB: ..., 7.1%, 14%, 21%, 34%, 41%, 54%, 61%, 74%, 81%, 95%
Reading symbols from /usr/lib/debug/boot/kernel/uhid.ko.debug...done.
Loaded symbols from /usr/lib/debug/boot/kernel/uhid.ko.debug
Reading symbols from /usr/lib/debug/boot/kernel/uhid.ko.debug...done.
Loaded symbols from /usr/lib/debug/boot/kernel/uhid.ko.debug
Reading symbols from /usr/lib/debug/boot/kernel/ksyms.ko.debug...done.
Loaded symbols from /usr/lib/debug/boot/kernel/ksyms.ko.debug
#0 dawump (textdumpvalue optimized out?) at pcpu.h:221
221 pcpu.h: No such file or directory.
in pcpu.h
(kjdbc)
```
Compat code
Attack surface entrypoint

• The BSDs have binary compatibility [compat] support for some binaries:
  • Older versions of the OS
  • 32bit versions of a program (on a 64bit version of the OS)
  • Other operating system (e.g. Linux)
• Has to emulate a bunch of stuff (e.g. syscalls)

“The people who rely on the compat layers don't care enough to maintain it. The people who work on the mainline system don't care about the compat layers because they don't use them. The cultures aren't aligned in the same direction. **Compat layers rot very quickly.**” – Theo De Raadt
static int
ti_bind(file_t *fp, int fd, struct svr4_strioctl *ioc, struct lwp *l)
{
    ...
    struct svr4_strmcmd bnd;
    ...
    if (ioc->len > sizeof(bnd))
        return EINVAL;
    if ((error = copyin(NETBSD32PTR(ioc->buf), &bnd, ioc->len)) != 0)
        return error;
    ...
    switch (st->s_family) {
    case AF_INET:
        ...
        netaddr_to_sockaddr_in(&sain, &bnd);
        ...
    }
    ...
}

#define SVR4_C_ADDROF(sc) (const void *) (((const char *) (sc)) + (sc)->offs)

static void netaddr_to_sockaddr_in
(struct sockaddr_in *sain, const struct svr4_strmcmd *sc)
{
    const struct svr4_netaddr_in *na;
    na = SVR4_C_ADDROF(sc); ← could point to anywhere in memory
    memset(sain, 0, sizeof(*sain));
    sain->sin_len = sizeof(*sain);
    sain->sin_family = na->family; ← crash or info leak
    sain->sin_port = na->port; ← crash or info leak
    sain->sin_addr.s_addr = na->addr; ← crash or info leak
    ...
    /*
     * Pretend that we have streams...
     * Yes, this is gross.
     ...
     */
Sample bug

• SVR 4 streams compat code
• NetBSD 7.1
  • Been there since NetBSD 1.2 *Thu Apr 11 12:49:13 1996*
• Uses offset that comes from userland
  • Without any validation
• Can read arbitrary(-ish) kernel memory
  • Panic
  • Info leak
• CVS commit message on the bugfix:

Log Message:
Fix some of the multitudinous holes in svr4 streams.
We should never have enabled this by default; it is a minefield.
Trap handlers
Attack surface entrypoint

- Trap handlers handle some kind of exception or fault
  - Div by zero
  - Syscall
  - Breakpoint
  - Invalid memory access
  - ...

- Some can be triggered by userland, and the kernel has to handle them correctly

- Due to their nature, they are ugly and highly architecture specific
Fuzz it!

• what would happen if you simply executed a bunch of random bytes as instructions?

• Surely a bunch of traps will get generated, and the kernel would have to handle them.

```c
int rfd;

void execute_code(unsigned char *p) {
    int (*fn)();
    fn = p;
    fn();
    return;
}

void fuzz() {
    unsigned char *code = mmap(NULL, lenbuf, PROT_EXEC | PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    while(1) {
        read(rfd, code, lenbuf);
        int pid = fork();
        if (pid == -1) {
            exit(0);
        } else if (pid == 0) {
            execute_code(code);
        } else {
            int status;
            pid_t r;
            r = waitpid(pid, &status, 0);
            if (r == -1) {
                kill(pid, 9);
                sleep(1);
                waitpid(pid, &status, WNOHANG);
            }
        }
    }
}

int main(void) {
    rfd = open("/dev/urandom", O_RDONLY);
    fuzz();
    return 0;
}
```
demo!
Hit trap bugs

• Xen
td sendsignal()
invalid signal 0

cpu = 0

KDB: stack backtrace:
#0 0xffffffff80b24077 at kdb_backtrace+0x57
#1 0xffffffff80a93e22 at vpsnic4x+0x182
#2 0xffffffff80a9253 at panic+0x4d3
#3 0xffffffff8063d47 at tdsendsignal+0x1a07
#4 0xffffffff80adf2920 at trap+0x230
#5 0xffffffff80a9f948 at trap+0x6e8
#6 0xffffffff80bf1d81 at calltrap+0x8

Uptime: 15m47s
Dumping 138 out of 991 MB: 0.69% 2.44% 3.35% 4.78% 58.76% 81.93% 93%

Reading symbols from /usr/lib/debug/boot/kernel/uhid.ko.debug...done.
Loaded symbols for /usr/lib/debug/boot/kernel/uhid.ko.debug
#0 doadump (textdump=value optimized out) at pcpu.h:221
221  pcpu.h: No such file or directory.

(kd) bt
#0 0xffffffff80ad69 in kern_reboot (howto=265) at /usr/src/sys/kern/kern_shutdown.c:366
#1 0xffffffff80a94b1 in vpanic (fmt=value optimized out, ap=value optimized out) at /usr/src/sys/kern/kern_shutdown.c:759
#2 0xffffffff80a89c53 in panic (fmt=0x80) at /usr/src/sys/kern/kern_shutdown.c:889
#3 0xffffffff80604487 in tdsendsignal (p=value optimized out, td=value optimized out, sig=value optimized out, ksi=value optimized out) at /usr/src/sys/kern/kern_sig.c:2104
#4 0xffffffff80a8129b in trapsignal (td=0xffffffff80a9d4f0, t=40, ksi=0xffffffff80a8134) at /usr/src/sys/kern/kern_sig.c:1988
#5 0xffffffff80f9a0484 in trap (frame=0xffffffff80003ad4) at /usr/src/sys/amd64/amd64/trap.c:611
#6 0xffffffff80f9a01141 in calltrap () at /usr/src/sys/amd64/amd64/exception.c:523
#7 0x0000000000604187 in ?? ()

Previous frame inner to this frame (corrupt stack?)
Current language: auto; currently minimal
(kd)
File systems
Attack surface entrypoint

• Filesystem attack surface seems easy enough.
  • Malicious fs image that gets mounted
    • Also do file operations on them once mounted
  • Is certainly attack surface

• However, there is more!

• In recent years all 3 BSDs support fuse
• VFS layer now has to deal with malicious data that comes from userland
  • Before it always came from a trusted file system driver
Attack surface entrypoint [fuse]

• FBSD/OBSD/NBSD all have different fuse implementations (no shared code whatsoever)
  • NBSD: most complete (allows for the most file operations)
  • FBSD: most controlled arguments passed back and forth (getattr, readdir) less opportunity for consumers to make mistakes, but more parsing/processing in fusefs itself, more potential for bugs in fuse code itself
  • OBSD: minimal functional implementation (compared to the previous two)

• none implement ioctl

• all do:
  • read
  • write
  • readdir
  • getattr
  • setattr
  • ...

```c
int vfs_getcwd_scandir(struct vnode **lvpp, struct vnode **uvpp, char **bpp, char *bufp, struct proc *p) {
    int eofflag, tries, dirbuflen, len, reclen, error = 0;
    ...
    struct vaJr va;
    ...
    error = VOP_GETATTR(lvp, &va, p->p_ucred, p);
    ...
    data can come from fusefs...
    dirbuflen = DIRBLKSIZ;
    if (dirbuflen < va.va_blocksize)
        dirbuflen = va.va_blocksize;
        fusefs can make this really big
    dirbuf = malloc(dirbuflen, M_TEMP, M_WAITOK);
        malloc() will panic on very large values...
    error = VOP_READDIR(uvp, &uio, p->p_ucred, &eofflag);
        fusefs can provide arbitrary content...
    cpos = dirbuf;
    ...
    int eofflag;
    ...
    struct va;
    ...
    for (len = (dirbuflen - uio.uio_resid); len > 0; len -= reclen) {
        dp = (struct dirent *)cpos;
        reclen = dp->d_reclen;
        /* Check for malformed directory */
        if (reclen < DIRENT_RECSIZE(1)) {
            error = EINVAL;
            goto out;
        }
        if (dp->d_fileno == fileno) {
            char *bp = *bpp;
            bp -= dp->d_namlen;  /* fusefs can lie about d_namlen
                                    if (bp <= bufp) {
                                        error = ERANGE;
                                        goto out;
                                    }
        }
        memmove(bp, dp->d_name, dp->d_namlen);  /* out of bound read.
```
Sample bug

- Unbound malloc and out of bound read (could panic or info leak)
- OpenBSD 6.1
  - Been there since OpenBSD 4.0 Fri Apr 28 08:34:31 2006
- getcwd syscall when taking data from fuse / userland
static daddr_t
ext2_nodealloccg(struct inode *ip, int cg, daddr_t ipref, int mode)
{
    ...
    error = bread(ip->i_devvp, fsbtodb(fs,
        fs->e2fs_gd[cg].ext2bgd_i_bitmap),
        (int)fs->e2fs_bsize, NOCREDS, &bp); ← read from filesystem
    ...
    ibp = (char *)bp->b_data;
    ...
    len = howmany(fs->e2fs->e2fs_ipg - ipref, NBBY);
    loc = memchr(&ibp[start], 0xff, len);
    if (loc == NULL) {
        len = start + 1;
        start = 0;
        loc = memchr(&ibp[start], 0xff, len); ← logic driven by fs data
        if (loc == NULL) {
            printf("cg = %d, ipref = %lld, fs = %s\n",
                cg, (long long)ipref, fs->e2fs_fsmnt);
            panic("ext2fs_nodealloccg: map corrupted"); ← panic driven by fs data
            /* NOTREACHED */
        }
    }
    ...
}
Sample bug 2

- panic() driven by filesystem data
- FreeBSD 11
  - Been there since FreeBSD 8.1 *Thu Jan 14 14:30:54 2010*
- Ext2 file system code
Networking (bt, wifi, irda)
Wifi Attack surface entrypoint

• Stack itself
  • 802.11 network data
  • Parsing
  • Info leaks

• Wifi drivers
  • Data send by device to host
802.11 stack

• One 802.11 stack for all wifi drivers
• Much easier to maintain
  • Need to fix in only 1 place if bugs are found
• ieee80211_input() is main parsing input
  • Called from all wifi drivers
```c
ieee80211_eapol_key_input(struct ieee80211com *ic, struct mbuf *m,
                        struct ieee80211_node *ni)
{
    struct ifnet *ifp = &ic->ic_if;
    struct ether_header *eh;
    struct ieee80211_eapol_key *key;
    ...
    eh = mtod(m, struct ether_header *);
    ...
    if (m->m_len < sizeof(*key) &&
        (m = m_pullup(m, sizeof(*key))) == NULL) { /* guarantees that there are sizeof(struct ieee80211_eapol_key) continuous bytes in the mbuf */
        ...
        key = mtod(m, struct ieee80211_eapol_key *);
        ...
        if (m->m_pkthdr.len < 4 + BE_READ_2(key->len)) /* assume key->len is larger than key->payload */
            goto done;
    }
    /* check key data length */
    totlen = sizeof(*key) + BE_READ_2(key->paylen); /* assume key->len is larger than key->payload */
    if (m->m_pkthdr.len < totlen || totlen > MCLBYTES)
        goto done;
    /* make sure the key data field is contiguous */
    if (m->m_len < totlen && (m = m_pullup(m, totlen)) == NULL) { /* not enough data pulled up if key->len is larger than key->payload! */
        ...
        key = mtod(m, struct ieee80211_eapol_key *);
        ...    ieee80211_recv_4way_msg3(ic, key, ni); /* can crash in here if not enough data is pulled up. */
        ...
    }
}```
802.11 Stack sample bug

• mbuf mishandling, leading to crash
  • Doesn’t guarantee it pulls up enough mbuf data

• OpenBSD 6.1
  • Bug has been there for almost 9 years

• Parsing EAPOL frames
802.11 Drivers

- Wifi drivers are either PCI or USB
- Do you trust the radio?
  - What if it does get compromised?
- Assume PCI cards cause total compromise (they can do DMA)
  - Well, actually, with IOMMU that’s no longer the case …
- USB is packet based protocol
  - Host USB parsers should be able to parse safely
    - Currently BSD wifi drivers do not do this!
      - Leads to trivial heap smashes
void run_rx_frame(struct sc *sc, uint8_t *buf, int dmalen) {
    struct rt2860_rxwi *rxwi;
    uint16_t len;
    rxwi = (struct rt2860_rxwi *)buf;
    len = letoh16(rxwi->len) & 0xfff;
    can be at most 4095
    /* could use m_devget but net80211 wants config mgmt frames */
    MGETHDR(m, M_DONTWAIT, MT_DATA);
    if (__predict_false(m == NULL)) {
        ifp->if_ierrors++;
        return;
    }
    if (len > MHLEN) {
        MCLGET(m, M_DONTWAIT);
        if (__predict_false(!(m->m_flags & M_EXT))) {
            ifp->if_ierrors++;
            m_freem(m);
            return;
        }
    }
    /* finalize mbuf */
    memcpy(mtod(m, caddr_t), wh, len);
    m->m_pkthdr.len = m->m_len = len;
}

/* Finalize mbuf. */
memcpy(mtod(m, caddr_t), wh, mlen);
/* len can be ~8k. can cause memory corruption. */

/* A frame has been uploaded: pass the resulting mbuf chain up to
the higher level protocols. */
void attu_rxeof(struct usbd_xfer *xfer, void *priv, usbd_status status) {
    h = (struct attu_rx_hdr *)c->atu_buf;
    len = UGETW(h->length) - 4; /* XXX magic number */
    m = c->atu_mbuf;
    memcpy(mtod(m, char *), c->atu_buf + ATU_RX_HDRLEN, len);
    /* need to validate len before copy. can cause memory corruption */
    usbd_setup_xfer(c->atu_xfer, sc->atu_ep[ATU_ENDPT_RX], c, c->atu_buf, ATU_RX_BUFSIZE, USBD_SHORT_XFER_OK | USBD_NO_COPY, USBD_NO_TIMEOUT, attu_rxeof);
    usbd_transfer(c->atu_xfer);
}

void otus_sub_rxeof(struct otus_softc *sc, uint8_t *buf, int len) {
    len comes from usb. can be ~8k
    ...
    plcp = buf;
    ...  
    mlen = len - AR_PLCP_HDR_LEN - sizeof(*tail);
    mlen -= IEEE80211_CRC_LEN; /* strip 802.11 FCS */
    wh = (struct ieee80211_frame *)(plcp + AR_PLCP_HDR_LEN);
    /* Build a fake beacon frame to let net80211 do all the parsing. */
    pktlen = sizeof(*wh) + letoh32(bss->ieslen); /* could int overflow */
    if (__predict_false(pktlen > MCLBYTES)) /* signedness issue */
        return;
    MGETHDR(m, M_DONTWAIT, MT_DATA);
    if (__predict_false(m == NULL))
        return;
    if (pktlen > MHLEN) {
        MCLGET(m, M_DONTWAIT);
        if (!m->m_flags & M_EXT) {
            m_free(m);
            return;
        }
    }
    wh = mtod(m, struct ieee80211_frame *);
    ...  
    memcpy(&wh[1], (uint8_t *)&bss[1], letoh32(bss->ieslen)); /* memory corruption */
    ...  
    }
802.11 drivers sample bug

• Wide open attack surface
  • Atmel AT76C50x IEEE 802.11b wireless network device [atu(4)]
  • Atheros USB IEEE 802.11a/b/g/n wireless network device [otus(4)]
  • Realtek RTL8188SU/RTL8192SU USB IEEE 802.11b/g/n wireless network device [rsu(4)]
  • Ralink Technology/MediaTek USB IEEE 802.11a/b/g/n wireless network device [run(4)]
  • Atheros USB IEEE 802.11a/b/g wireless network device [uath(4)]

• Across all BSDs

• They didn’t think about the attack surface on this one
miscellaneous

• NULL derefs
  • malloc(len, type, M_NOWAIT/M_CANFAIL)
  • Not checking return value
    • Relatively frequent bug
    • M_WAITOK (== can never fail) is a very common case
      • Developers treating M_NOWAIT/M_CANFAIL code as if it was M_WAITOK

• DRM/DRI
  • DRM/DRI code base is part of linux kernel source tree.
    • BSD folks fork it
      • Code quality is about as big a shit sandwich as it is in linux DRM/DRI code base

“All this linux code that we are importing ... is not going to be reviewed by any of the other OpenBSD kernel developers ... because they refuse to read any code that is not conformant to the BSD KNF standard” – Matthieu Herrb
Results

• results:
  • About ~115 kernel bugs so far
  • FBSD: ~30
  • OBSD: ~25
  • NBSD: ~60

• types of bugs seen:
  • Straight heap/stack smash
  • race conditions
  • expired pointers
  • Double frees
  • recursion issues
  • integer issues
    • Underflows, overflows, signedness
  • Refcount issues
    • Overflows
    • Reference dropped too soon → use after free
  • info leaks
  • out of bound read
  • NULL deref
  • Logic bugs
  • typo
  • Division by zero
  • kernel panics driven by userland
  • Memory leaks
Conclusions

• Bugs were found in all 3 of the examined BSDs
  • Among all of the attack surfaces mentioned above

• Winner / loser
  • OBSD clear winner (they have massively reduced their attack surface over the years):
    • Attack surface reduction
      • no loadable modules
      • relatively few devices
      • Virtually no compat code (they removed Linux a couple of years ago)
      • removed entire Bluetooth stack
      • Significantly less syscalls (e.g. 200+ syscalls less than FBSD)
      • Cut support for some older architectures
  • Code Quality
    • int overflows / signedness bugs, as good as gone in most places
    • Few info leaks
  • NBSD clear loser
    • Tons of legacy and compat code (who the hell still needs the ISO protocols ??? Really?)
    • seems to be less consistent with security code quality
      • Too many signedness bugs.
    • This is NOT a dis.
      • if you think building/maintaining/improving and OS is easy, go ahead, try it. See how far you get.
  • FBSD is somewhere in between
Conclusions

• Security team responses

  • OpenBSD
    • About a week or so to get response (Theo said it took so long because he was on vacation)
    • Bugfixes rolled out in the next couple of days

  • FreeBSD
    • Response in about a week or so
    • The filed bugs internally
      • Don’t know what the status of those bugs is

  • NetBSD
    • They fixed virtually all bugs submitted. Pretty much **overnight !!!**
    • That is ridiculously **impressive.**
    • They also turned off the SVR4 subsystem for i386 by default

  “We have also disabled COMPAT_SVR4 by default on i386, something that should have been done a long time ago.” – Response from NetBSD developers
More conclusions

• Bugs are still easy to find in those kernels. Even OpenBSD.

• Varying level of quality depending on age and who wrote it
  • Most consistent quality was observed with OpenBSD

• The maintainers of various BSDs should talk more among each other
  • Several bugs in one were fixed in the other
    • OpenBSD expired proc pointer in midioctl() fixed in NetBSD
    • NetBSD signedness bug in ac97_query_devinfo() fixed in OpenBSD
More conclusions

• Code base size
  • OpenBSD: 2863505 loc
  • NetBSD: 7330629 loc
  • FreeBSD: 8997603 loc

• Obviously this plays a part
  • Can’t have a bug in code you don’t have

• Accidental vs. planned
  • Haven’t gotten to implementing something yet or ...
  • Choice made on purpose to delete code
    • Attack surface reduction
More conclusions

• Many eyeballs ...

• Gut feeling, I suspect this is a factor.

• Based on my result, code quality alone can’t account for the discrepancy between the bug numbers (BSD vs. Linux).

• Say what you will about the people reviewing the Linux kernel code, there are simply orders of magnitude more of them. And it shows in the numbers.
Questions / comments from the internet

• Defcon releases presentation before you actually perform the presentation
  • This is kindof annoying....
  • People saw it, and commented on it before I had a chance to stand up here ...
    • Surprisingly little hate / trolls (yet?)
    • Got some from the OS zealots

• Why didn’t you do the same for linux?
  • The numbers presented earlier speak for themselves IMO. Assumed them (and what they imply) to be accurate
  • You conclude that linux is better! And its not even the subject of the presentation
    • I did not.
  • You should’ve added a comparison of protection mechanisms between linux and the BSDs
    • While that would be interesting, its far outside of the scope. Doesn’t relate to code quality. Also time constraints.
• What about dragonflyBSD? HardenedBSD?
  • I Considered it. Maybe I should have too. I had limited amount of time and picked the 3 most commonly used BSDs
Questions / comments from the internet

• Interesting related links I got from the internet
  • *How to find 56 potential vulnerabilities in FreeBSD code in one evening, PVS-Studio delved into the FreeBSD kernel*
    • [https://www.viva64.com/en/b/0496/](https://www.viva64.com/en/b/0496/)
    • [https://www.viva64.com/en/b/0377/](https://www.viva64.com/en/b/0377/)
    • [https://www.viva64.com/en/b/0487/](https://www.viva64.com/en/b/0487/)

• FreeBSD – A lesson in poor defaults
  • [https://vez.mrsk.me/freebsd-defaults.txt](https://vez.mrsk.me/freebsd-defaults.txt)
Questions ?

That’s all Folks!