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?

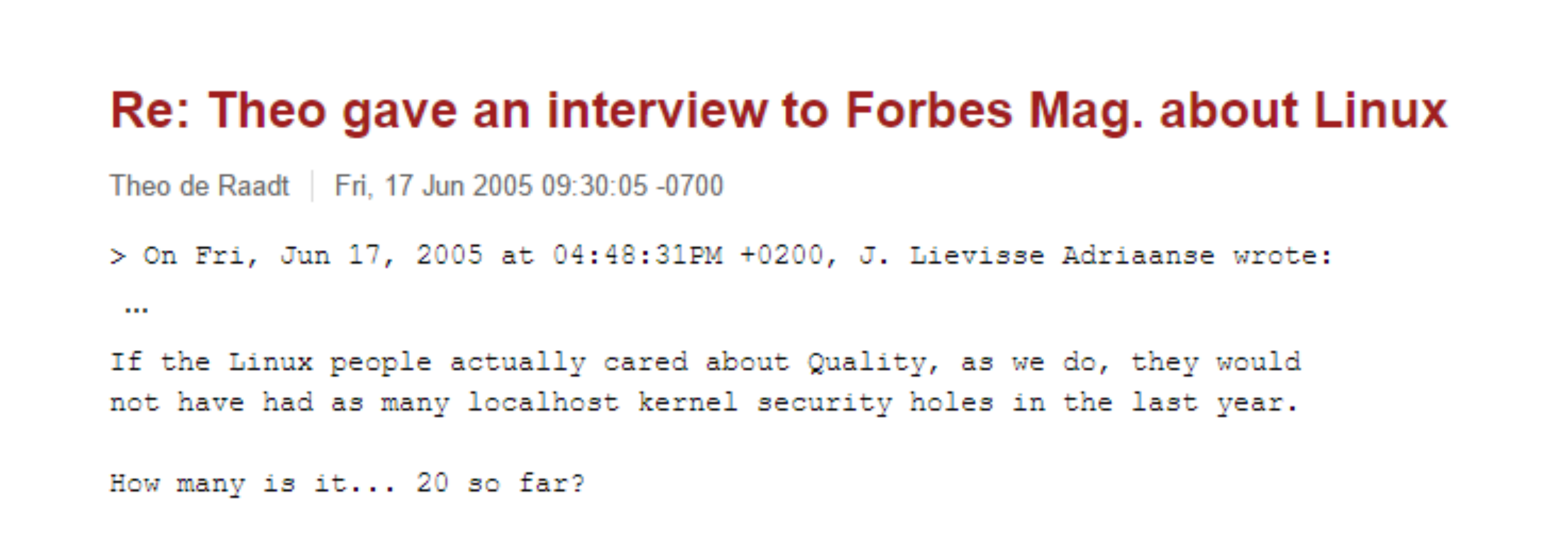
- BSD kernel vulnerabilities
    - Comparison
    - Between different BSD flavors

- 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 Etelavuori
  - 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

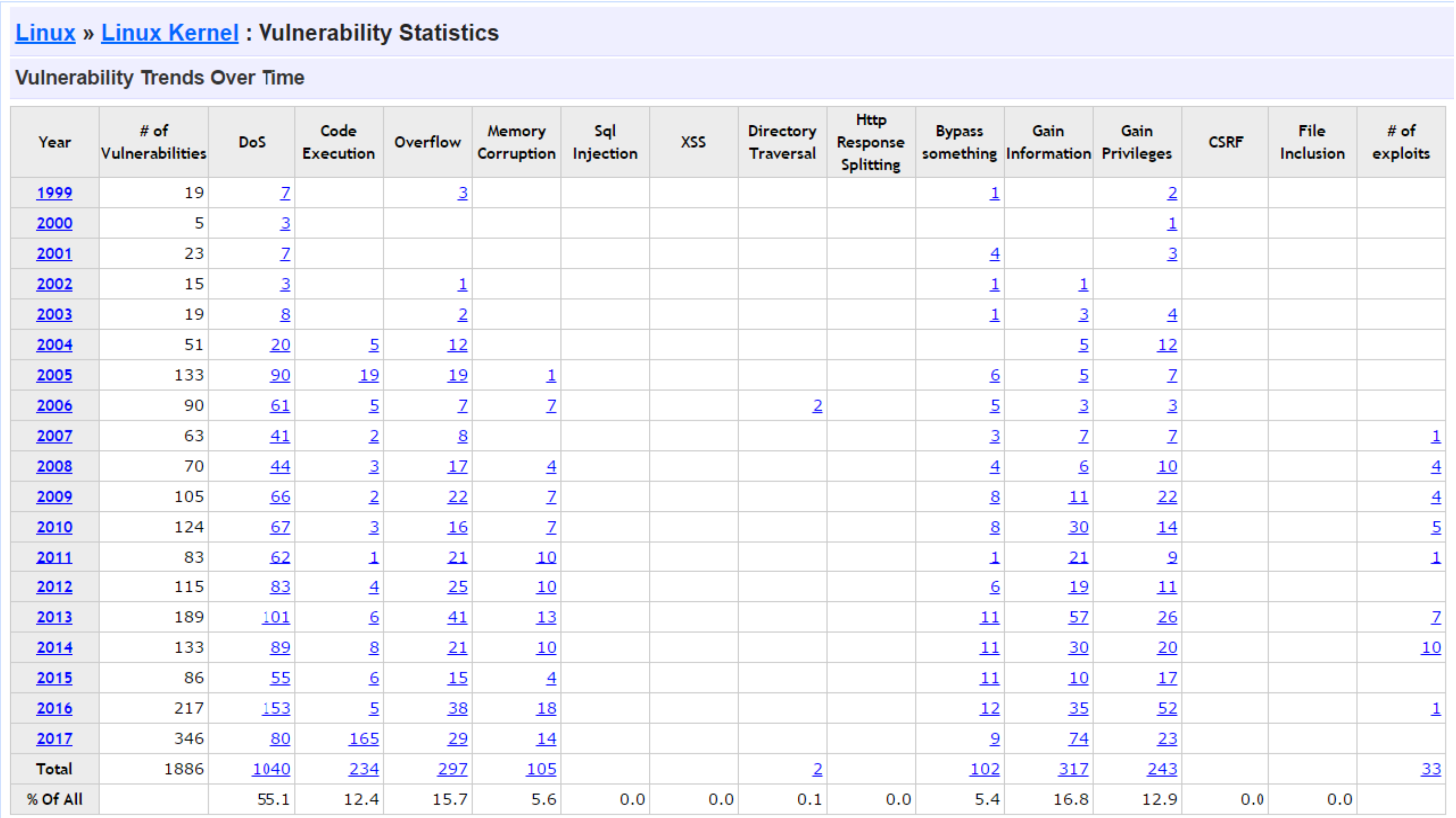
```
> On Fri, Jun 17, 2005 at 04:48:31PM +0200, J. Lievisse Adriaanse wrote:
 ...
If the Linux people actually cared about Quality, as we do, they would
not have had as many localhost kernel security holes in the last year.

How many is it... 20 so far?
```

intro

Really? Got Data?

- Somehow that statement has always been stuck in my head
- Is it true?
- Can we look at some data ?

Source: https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html



Linux » Linux Kernel : Vulnerability Statistics

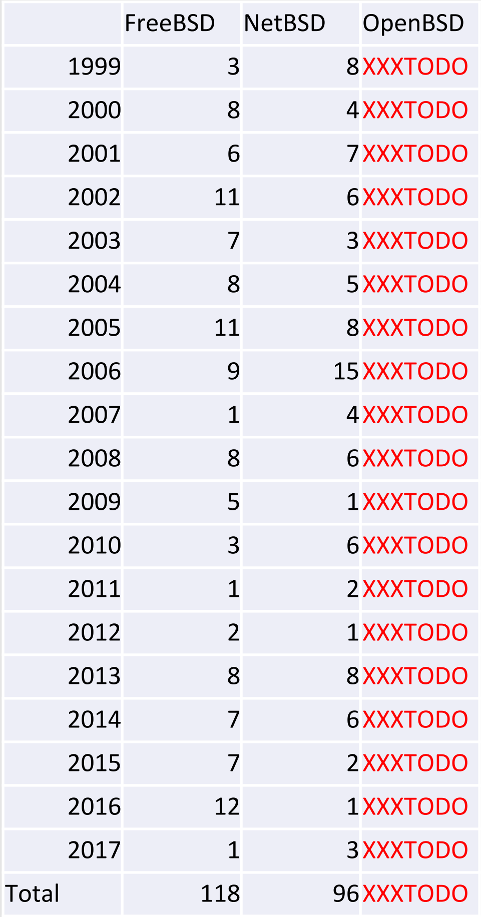
Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	19	7		3						1		2			
2000	5	3										1			
2001	23	7								4		3			
2002	15	3		1						1	1				
2003	19	8		2						1	3	4			
2004	51	20	5	12							5	12			
2005	133	90	19	19	1					6	5	7			
2006	90	61	5	7	7			2		5	3	3			
2007	63	41	2	8						3	7	7			1
2008	70	44	3	17	4					4	6	10			4
2009	105	66	2	22	7					8	11	22			4
2010	124	67	3	16	7					8	30	14			5
2011	83	62	1	21	10					1	21	9			1
2012	115	83	4	25	10					6	19	11			
2013	189	101	6	41	13					11	57	26			7
2014	133	89	8	21	10					11	30	20			10
2015	86	55	6	15	4					11	10	17			
2016	217	153	5	38	18					12	35	52			1
2017	346	80	165	29	14					9	74	23			
Total	1886	1040	234	297	105			2		102	317	243			33
% Of All		55.1	12.4	15.7	5.6	0.0	0.0	0.1	0.0	5.4	16.8	12.9	0.0	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

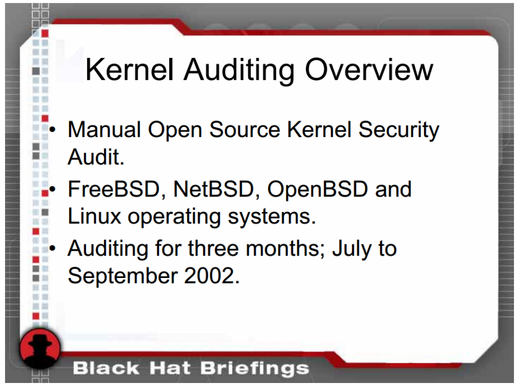


	FreeBSD	NetBSD	OpenBSD
1999	3	8	XXXTODO
2000	8	4	XXXTODO
2001	6	7	XXXTODO
2002	11	6	XXXTODO
2003	7	3	XXXTODO
2004	8	5	XXXTODO
2005	11	8	XXXTODO
2006	9	15	XXXTODO
2007	1	4	XXXTODO
2008	8	6	XXXTODO
2009	5	1	XXXTODO
2010	3	6	XXXTODO
2011	1	2	XXXTODO
2012	2	1	XXXTODO
2013	8	8	XXXTODO
2014	7	6	XXXTODO
2015	7	2	XXXTODO
2016	12	1	XXXTODO
2017	1	3	XXXTODO
Total	118	96	XXXTODO

- 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?

Test by audit!

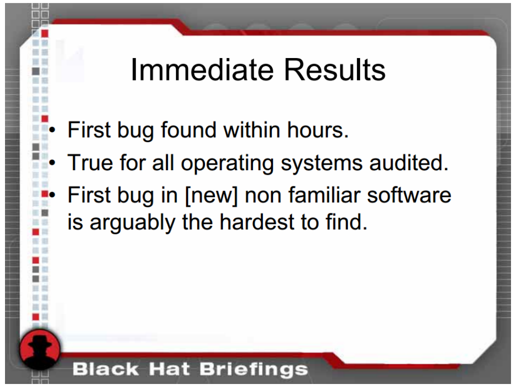
- Silvio Cesare did some interesting work in ~2002 that gives some answers
- https://www.blackhat.com/presentations/bh-usa-03/bh-us-03-cesare.pdf
- His results seem to indicate there isn't really that much of a quality difference. However:
  - that was well over a decade ago.
    - Have things changed?
  - 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 ?



Kernel Auditing Overview

- Manual Open Source Kernel Security Audit.
- FreeBSD, NetBSD, OpenBSD and Linux operating systems.
- Auditing for three months; July to September 2002.

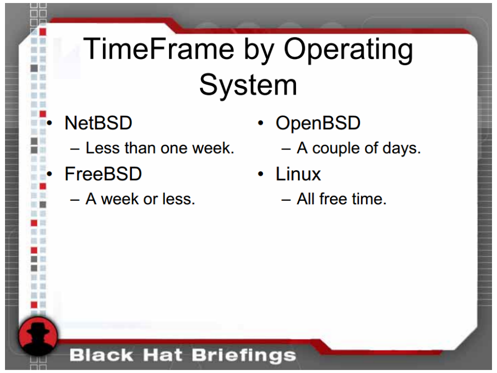
Black Hat Briefings



Immediate Results

- First bug found within hours.
- True for all operating systems audited.
- First bug in [new] non familiar software is arguably the hardest to find.

Black Hat Briefings



TimeFrame by Operating System

- NetBSD
  - Less than one week.
- FreeBSD
  - A week or less.
- OpenBSD
  - A couple of days.
- Linux
  - All free time.

Black Hat Briefings

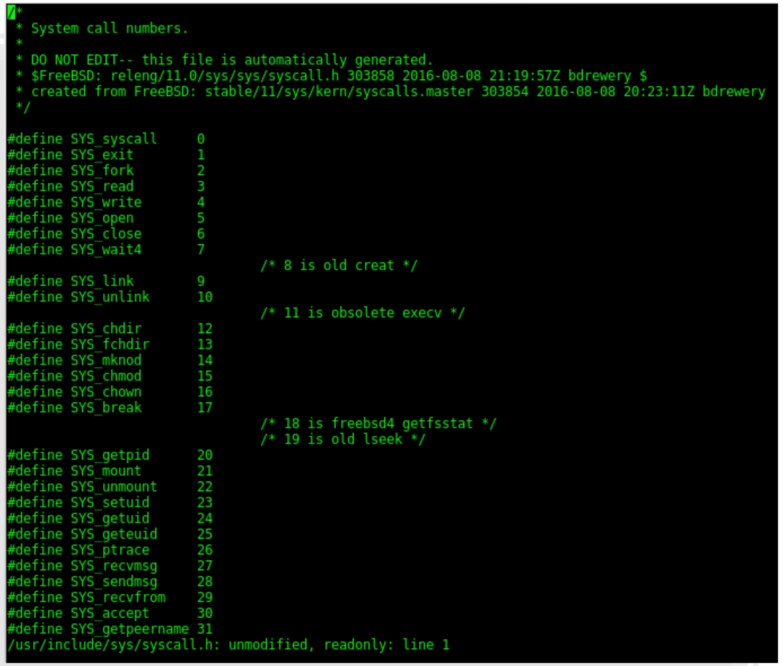
Test by Audit redux.

- 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

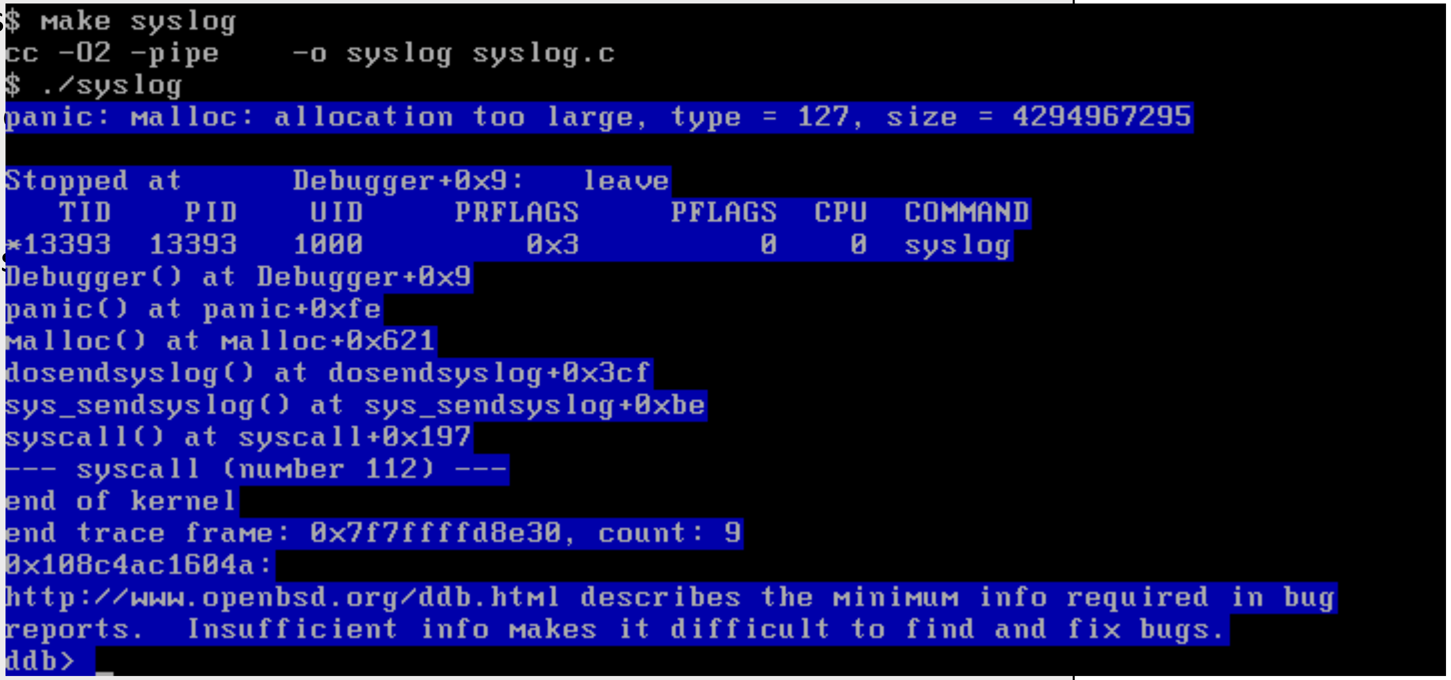
- The obvious attack surface
- Syscalls are how userland gets anything done from kernel
- Hundreds of them
  - FreeBSD: ~550
  - OpenBSD: ~330
  - NetBSD: ~480
- Assumption: given that they're obvious, and well tested, less likely to contain security bugs



```
/*
 * System call numbers.
 *
 * DO NOT EDIT-- this file is automatically generated.
 * $FreeBSD: releng/11.0/sys/sys/syscall.h 303858 2016-08-08 21:19:57Z bdrewery $
 * created from FreeBSD: stable/11/sys/kern/syscalls.master 303854 2016-08-08 20:23:11Z bdrewery
 */

#define SYS_syscall     0
#define SYS_exit        1
#define SYS_fork        2
#define SYS_read        3
#define SYS_write       4
#define SYS_open        5
#define SYS_close       6
#define SYS_wait4       7
                                /* 8 is old creat */
#define SYS_link        9
#define SYS_unlink      10
                                /* 11 is obsolete execv */
#define SYS_chdir       12
#define SYS_fchdir      13
#define SYS_mknod       14
#define SYS_chmod       15
#define SYS_chown       16
#define SYS_break       17
                                /* 18 is freebsd4 getfsstat */
                                /* 19 is old lseek */
#define SYS_getpid      20
#define SYS_mount       21
#define SYS_unmount     22
#define SYS_setuid      23
#define SYS_getuid      24
#define SYS_geteuid     25
#define SYS_ptrace      26
#define SYS_recvmsg     27
#define SYS_sendmsg     28
#define SYS_recvfrom    29
#define SYS_accept      30
#define SYS_getpeername 31
/usr/include/sys/syscall.h: unmodified, readonly: line 1
```
]

```
int
sys_sendsyslog(struct
{
        struct sys_



        } */ *uap =
        int error;
        static int d
...
        error = dos
SCARG(uap, nbyte),
          SCARG(u
...
        return (err
}
```

```
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_res
...
        len = auio.ui
        if (fp) {
...
        } else if (con
...
        } else {
...

...
        }
...
}
```



```
$ make syslog
cc -O2 -pipe    -o syslog syslog.c
$ ./syslog
panic: malloc: allocation too large, type = 127, size = 4294967295

Stopped at      Debugger+0x9:   leave
    TID     PID     UID     PRFLAGS     PFLAGS  CPU  COMMAND
*13393  13393    1000         0x3          0    0  syslog
Debugger() at Debugger+0x9
panic() at panic+0xfe
malloc() at malloc+0x621
dosendsyslog() at dosendsyslog+0x3cf
sys_sendsyslog() at sys_sendsyslog+0xbe
syscall() at syscall+0x197
--- syscall (number 112) ---
end of kernel
end trace frame: 0x7f7ffffd8e30, count: 9
0x108c4ac1604a:
http://www.openbsd.org/ddb.html describes the minimum info required in bug
reports.  Insufficient info makes it difficult to find and fix bugs.
ddb> _
```

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

- Previous assumption is not [entirely] true: bugs in syscalls do occur with some frequency
  - Especially newly added syscalls

TCP/IP stack

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

```
struct secpolicy *
key_msg2sp(

{
...

...

...
```

```
/* length check */
if (xisr->sadb_x_ipsecrequest_len < sizeof(*xisr)) {
          ipseclog((LOG_DEBUG, "key_msg2sp: "
           "invalid ipsecrequest length.\n"));
}
```

```
if (xisr->sadb_x_ipsecrequest_len > sizeof(*xisr)) {
          struct sockaddr *paddr;

          paddr = (struct sockaddr *)(xisr + 1);

          /* validity check */
          if (paddr->sa_len
                    > sizeof((*p_isr)->saidx.src)) {
                    ipseclog((LOG_DEBUG, "key_msg2sp: invalid request "
                     "address length.\n"));
                    key_freesp(newsp, KEY_SADB_UNLOCKED);
                    *error = EINVAL;
                    return NULL;
                    }
```

```
bcopy(paddr, &(*p_isr)->saidx.src,  paddr->sa_len);
```

length check is incomplete. sadb_x_ipsecrequest_len can be invalid

length check is incomplete. sadb_x_ipsecrequest_len can be invalid

length check is incomplete. paddr->sa_len can be invalid

this copy can out of bound read on paddr. Assume malicious user that controls heap chunk after paddr. could make it so paddr->sa_len is large and causes memory corruption

Sample bug

- IPSEC setsockopt()
- Out of bound read
- Can end up corrupting memory
- Affects:
  - FreeBSD 11
  - NetBSD 7.1
- 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!

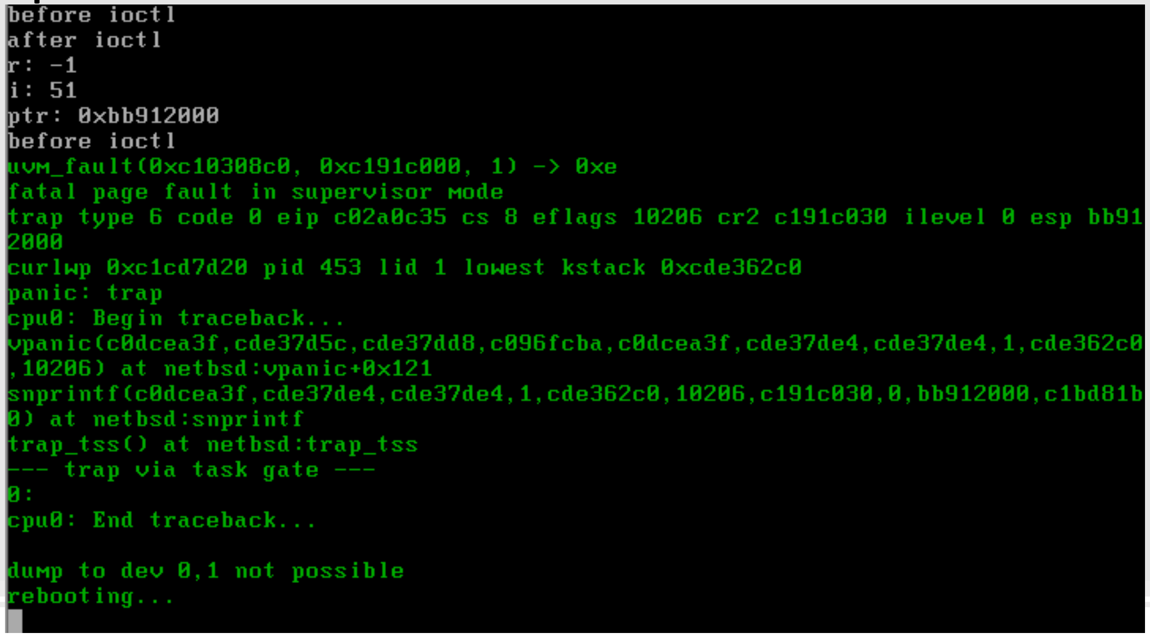
```
int
cryptof_ioctl(struct file *fp, u_long cmd, void *data)
{
...
        switch (cmd) {
...
                mutex_enter(&crypto_mtx);
                fcr->mtime = fcr->atime;
                mutex_exit(&crypto_mtx);
                mkop = (struct crypt_mkop *)data;
                knop = kmem_alloc((mkop->count * 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;
...
}
```

Integer overflow

Memory corruption due to int overflow

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



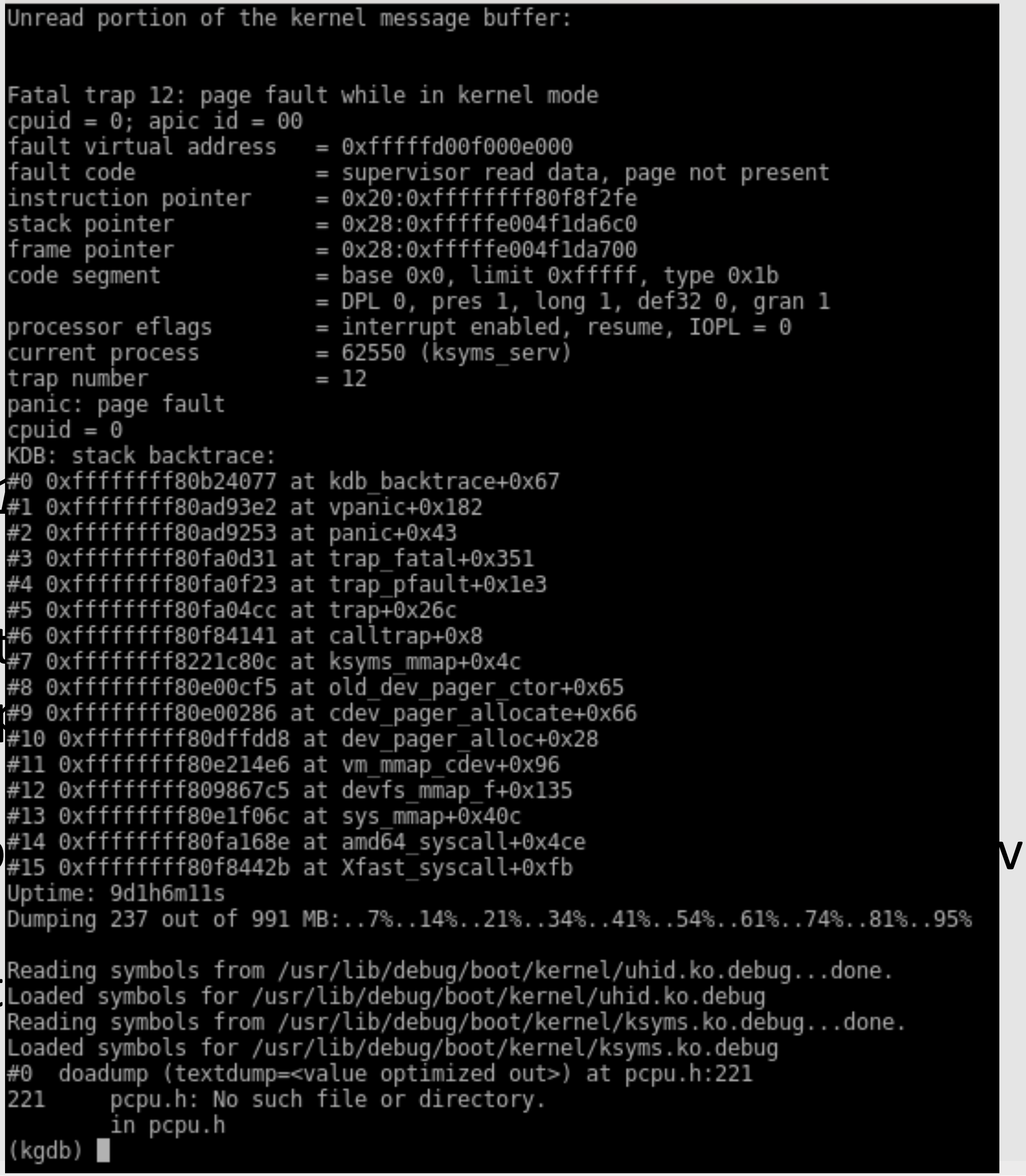
```
before ioctl
after ioctl
r: -1
i: 51
ptr: 0xbb912000
before ioctl
uvm_fault(0xc10308c0, 0xc191c000, 1) -> 0xe
fatal page fault in supervisor mode
trap type 6 code 0 eip c02a0c35 cs 8 eflags 10206 cr2 c191c030 ilevel 0 esp bb91
2000
curlwp 0xc1cd7d20 pid 453 lid 1 lowest kstack 0xcde362c0
panic: trap
cpu0: Begin traceback...
vpanic(c0dcea3f,cde37d5c,cde37dd8,c096fcba,c0dcea3f,cde37de4,cde37de4,1,cde362c0
,10206) at netbsd:vpanic+0x121
snprintf(c0dcea3f,cde37de4,cde37de4,1,cde362c0,10206,c191c030,0,bb912000,c1bd81b
0) at netbsd:snprintf
trap_tss() at netbsd:trap_tss
--- trap via task gate ---
0:
cpu0: End traceback...

dump to dev 0,1 not possible
rebooting...
```

```
static int
ksyms_open(st
{
...
        struc
...
        sc = (
           M_
...
        sc->s
        sc->s
...
        error
...
}
```

```
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*
- Expired pointer
  - open() callback saves pointer to pmap t
  - mmap() callback uses saved pointer in p
  - So how is this a problem ?
    - What if we hand fd off to another p ve)
    - And then we exit
    - If other process now does mmap, it



```
Unread portion of the kernel message buffer:


Fatal trap 12: page fault while in kernel mode
cpuid = 0; apic id = 00
fault virtual address   = 0xfffffd00f000e000
fault code              = supervisor read data, page not present
instruction pointer     = 0x20:0xffffffff80f8f2fe
stack pointer           = 0x28:0xfffffe004f1da6c0
frame pointer           = 0x28:0xfffffe004f1da700
code segment            = base 0x0, limit 0xfffff, type 0x1b
                        = DPL 0, pres 1, long 1, def32 0, gran 1
processor eflags        = interrupt enabled, resume, IOPL = 0
current process         = 62550 (ksyms_serv)
trap number             = 12
panic: page fault
cpuid = 0
KDB: stack backtrace:
#0 0xffffffff80b24077 at kdb_backtrace+0x67
#1 0xffffffff80ad93e2 at vpanic+0x182
#2 0xffffffff80ad9253 at panic+0x43
#3 0xffffffff80fa0d31 at trap_fatal+0x351
#4 0xffffffff80fa0f23 at trap_pfault+0x1e3
#5 0xffffffff80fa04cc at trap+0x26c
#6 0xffffffff80f84141 at calltrap+0x8
#7 0xffffffff8221c80c at ksyms_mmap+0x4c
#8 0xffffffff80e00cf5 at old_dev_pager_ctor+0x65
#9 0xffffffff80e00286 at cdev_pager_allocate+0x66
#10 0xffffffff80dffdd8 at dev_pager_alloc+0x28
#11 0xffffffff80e214e6 at vm_mmap_cdev+0x96
#12 0xffffffff809867c5 at devfs_mmap_f+0x135
#13 0xffffffff80e1f06c at sys_mmap+0x40c
#14 0xffffffff80fa168e at amd64_syscall+0x4ce
#15 0xffffffff80f8442b at Xfast_syscall+0xfb
Uptime: 9d1h6m11s
Dumping 237 out of 991 MB:..7%..14%..21%..34%..41%..54%..61%..74%..81%..95%

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

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
...
        if (ioc->len > sizeof(b
                return EIN

        if ((error = copyin(NI
                return err
...
        switch (st->s_family)
        case AF_INET:
...
                netaddr_t
...
        }
...
}
```

```
#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

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!

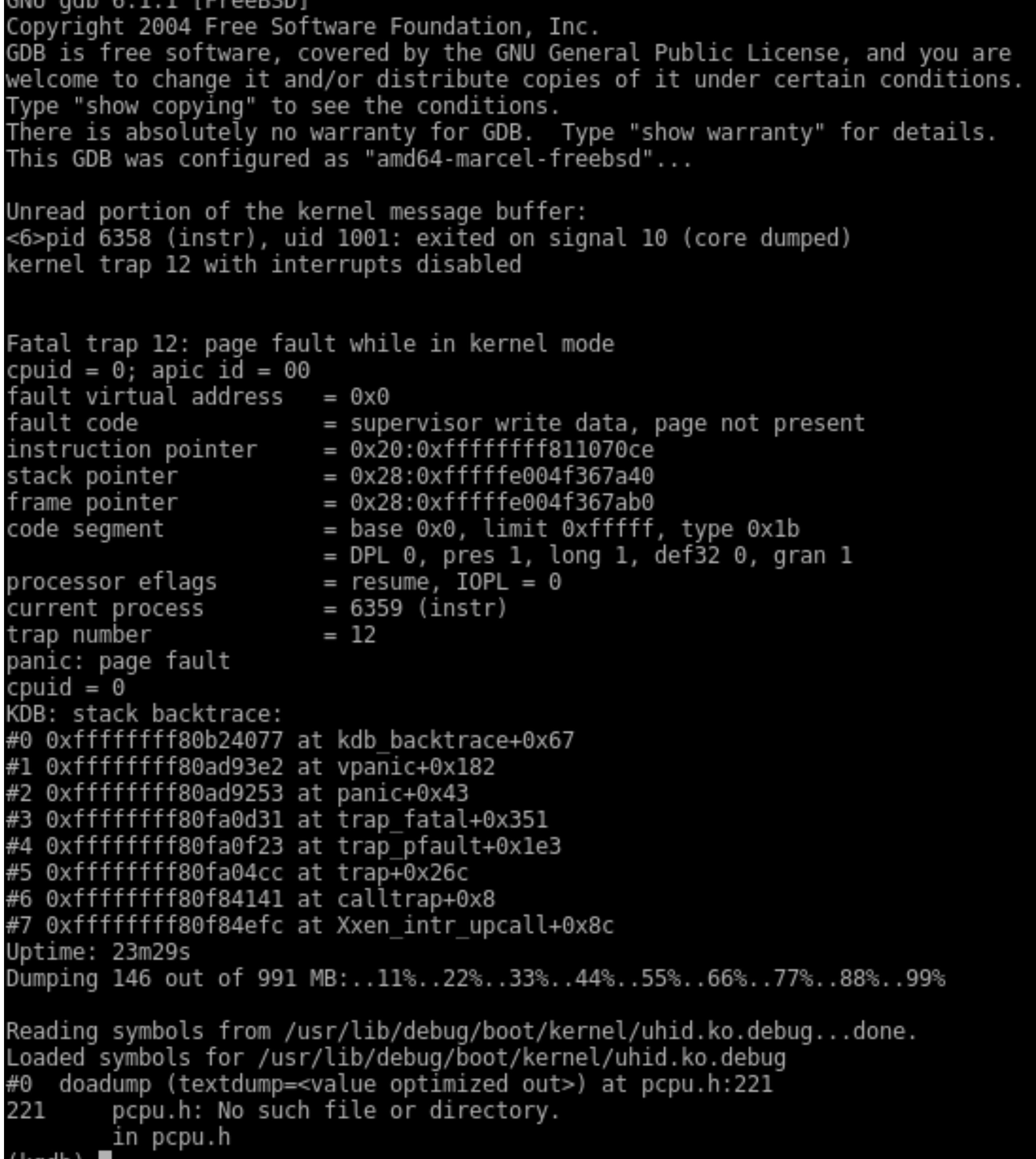
- what would happen i instructions?
- Surely a bunch of trap them

```
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();
}
```

demo!

Hit xen trap

- NULL deref



```
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "amd64-marcel-freebsd"...

Unread portion of the kernel message buffer:
<6>pid 6358 (instr), uid 1001: exited on signal 10 (core dumped)
kernel trap 12 with interrupts disabled


Fatal trap 12: page fault while in kernel mode
cpuid = 0; apic id = 00
fault virtual address   = 0x0
fault code              = supervisor write data, page not present
instruction pointer     = 0x20:0xffffffff811070ce
stack pointer           = 0x28:0xfffffe004f367a40
frame pointer           = 0x28:0xfffffe004f367ab0
code segment            = base 0x0, limit 0xfffff, type 0x1b
                        = DPL 0, pres 1, long 1, def32 0, gran 1
processor eflags        = resume, IOPL = 0
current process         = 6359 (instr)
trap number             = 12
panic: page fault
cpuid = 0
KDB: stack backtrace:
#0 0xffffffff80b24077 at kdb_backtrace+0x67
#1 0xffffffff80ad93e2 at vpanic+0x182
#2 0xffffffff80ad9253 at panic+0x43
#3 0xffffffff80fa0d31 at trap_fatal+0x351
#4 0xffffffff80fa0f23 at trap_pfault+0x1e3
#5 0xffffffff80fa04cc at trap+0x26c
#6 0xffffffff80f84141 at calltrap+0x8
#7 0xffffffff80f84efc at Xxen_intr_upcall+0x8c
Uptime: 23m29s
Dumping 146 out of 991 MB:..11%..22%..33%..44%..55%..66%..77%..88%..99%

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.
        in pcpu.h
(kgdb) 
```

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
  - …

```
int
vfs_getcwd_scandi
   char *bufp, struc
{
              int eofflag
...
              struct vat
...

...
              dirbuflen

              if (dirbufl


              dirbuf = r
...
```

```
error = VOP_READDIR(uvp, &uio, p->p_ucred, &eofflag); ← fusefs can provide arbitrary content
…
cpos = dirbuf;
…
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, NOCRED, &bp); ← read from filesystem
...
        ibp = (char *)bp->b_data;
...
        len = howmany(fs->e2fs->e2fs_ipg - ipref, NBBY);
        loc = memcchr(&ibp[start], 0xff, len);
        if (loc == NULL) {
                len = start + 1;
                start = 0;
                loc = memcchr(&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

```
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
r
{
...
...
...
...
...
...
...
}
```

```
/*
 * A
 * th
 */
void
atu_
{
...
...
}
```

memory corruption

```
void
otu
{
...
...
...
...
...
...
}
```

```
void
rsu_event_survey(struct rsu_softc *sc, uint8_t *buf, int len)
{
...
		struct ndis_wlan_bssid_ex *bss;
		struct mbuf *m;
		int pktlen;
...
		bss = (struct ndis_wlan_bssid_ex *)buf;
...
		if (__predict_false(len < sizeof(*bss) + letoh32(bss->ieslen)))  ← could int overflow
				return;
...
		/* 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

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
        - info leaks
        - out of bound read
        - NULL deref
        - 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 <u>most</u> 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.

    - FBSD is somewhere in between

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 midiioctl() 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 ?