A Bug Hunter’s Perspective on Linux Drivers

A walk through the land of I/O Control

Jeremy Brown, 2015

Agenda

I. **Introduction**

II. Device Drivers

    I. I/O Control

    II. Talking to a driver

III. Bug Hunting

    I. Discovery & Debugging

    II. Tooling

IV. Conclusion

#whoami

- Jeremy Brown
  - Independent researcher / consultant
  - Formerly of Microsoft
    - Windows/Phone/Xbox Security
    - Malware Protection Center
  - Also, Tenable
    - Nessus
    - RE patches

What I won’t teach you

- Comprehensive driver fundamentals
  - You need lots of time and a few different books
- How to become a driver security expert
  - Only a few of them around– djrbliss, spender, j00ru, etc
- Probably how to earn a living from these bugs
  - It’s OK to do research you enjoy, not just for $$$

What I hope to teach you

- What you “need to know” to get started
  - Determining attack surface
  - Finding and reviewing IOCTL handlers
  - Some primitive fuzzing techniques
- Knowledge applicable to *nix device drivers
  - Some of which translates to other OSes
- Take you from knowing nothing to something

Agenda

I. Introduction

II. Device Drivers

  I. I/O Control

  II. Talking to a driver

III. Bug Hunting

  I. Discovery & Debugging

IV. Conclusion

Device drivers

- Plug-in for the OS
  - Code that simply talks to a device
    - Commonly physical or virtual devices
  - Lives in kernel or user space
  - Natively or through abstraction layers
    - eg. wrappers or libraries

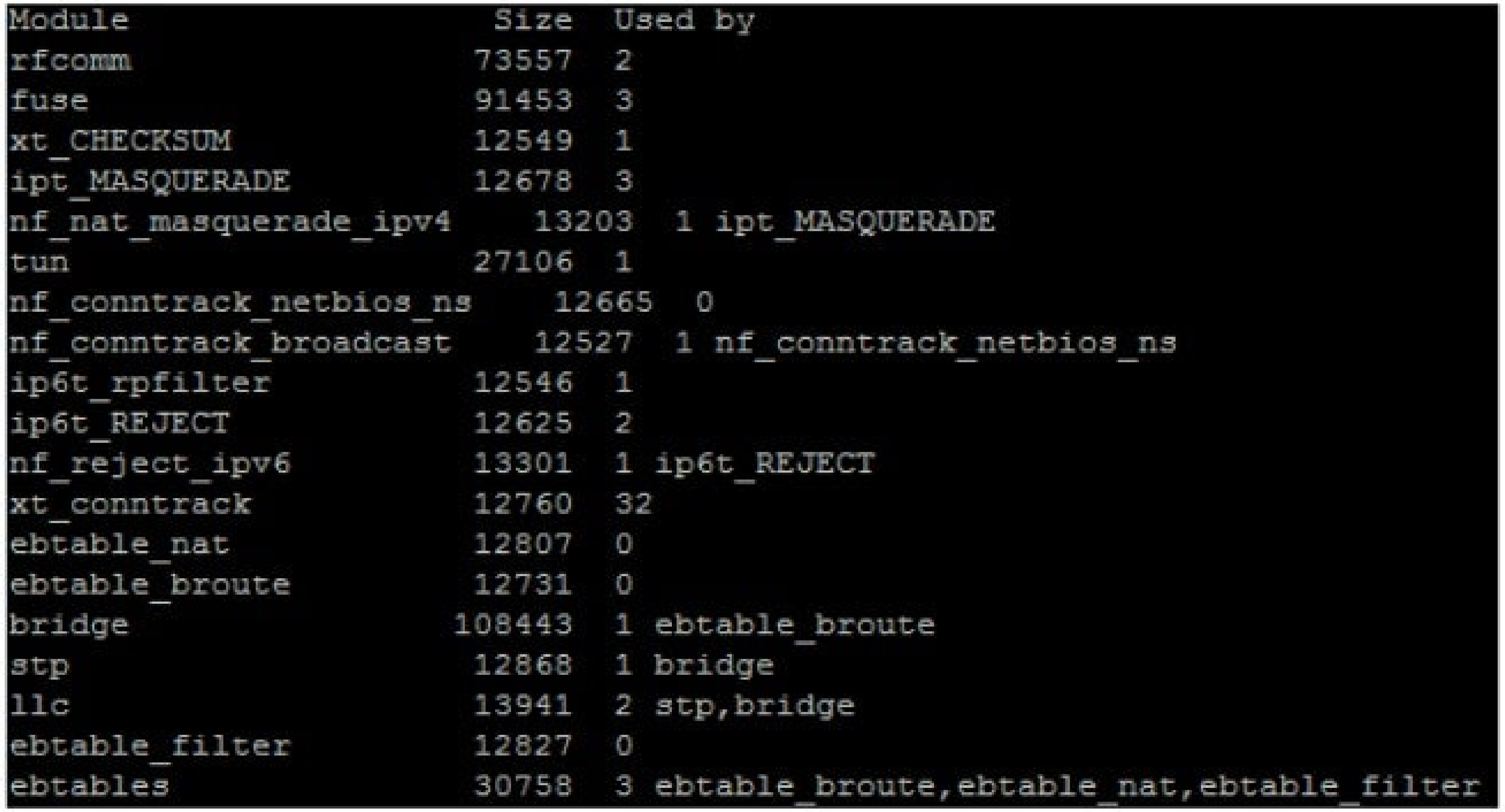
Frameworks

- Traditional Drivers
  - – Live in kernel space, eg. HID.ko
  - – Cons are kernel panics, debugging & overhead
- Userspace Drivers
  - – Live in user space, eg. native frameworks like UIO
  - – Interesting con – it can't register interrupt handler
    - Must either poll or use some other driver (kernel)
  - – Pro is that network libraries are mature

Reference: http://www.enea.com/Documents/Resources/Whitepapers/Enea-User-Space-Drivers-in-Linux_Whitepaper_2013.pdf

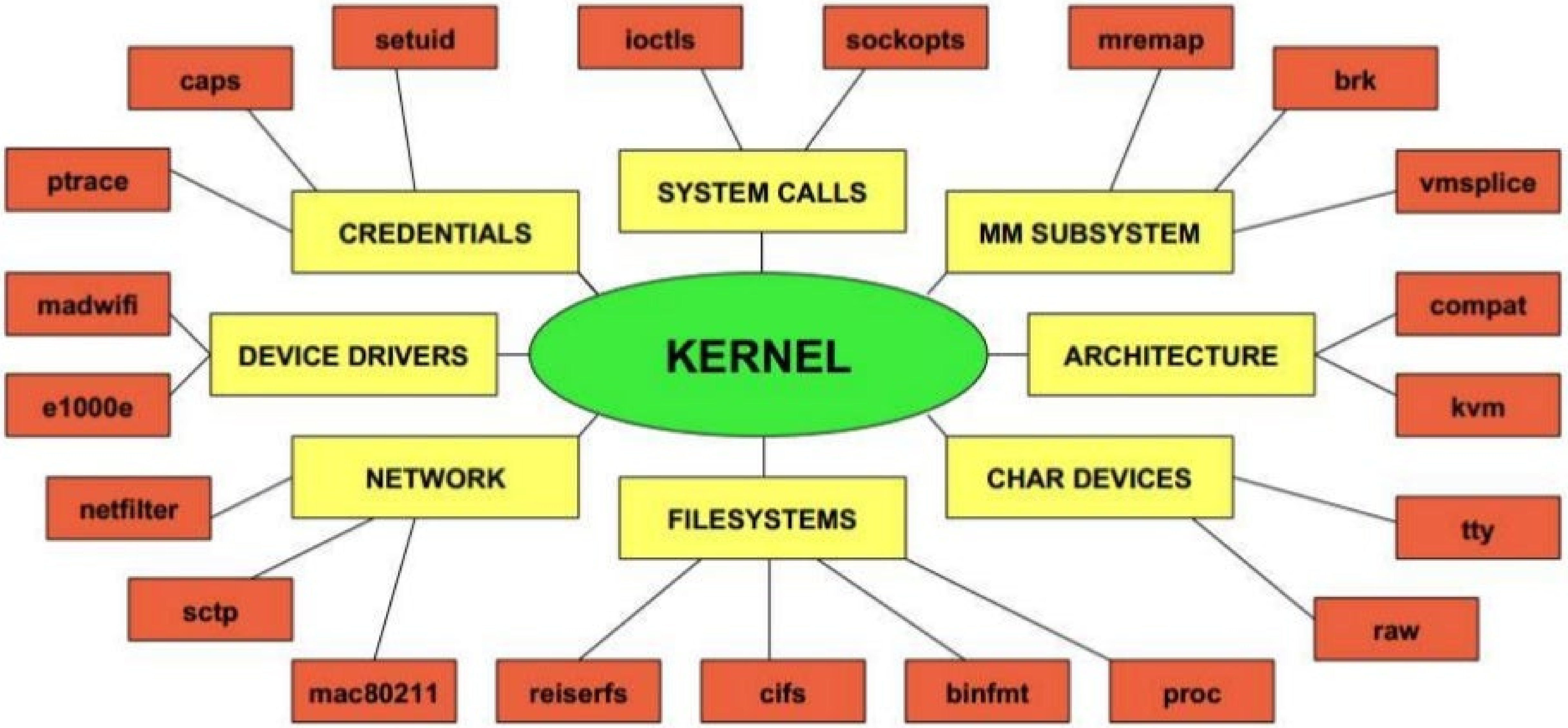
Where are these drivers?

- **lsmod** shows you which ones are loaded
  - – Parses /proc/modules



```
Module                  Size  Used by
rfcomm                 73557  2
fuse                   91453  3
xt_CHECKSUM            12549  1
ipt_MASQUERADE         12678  3
nf_nat_masquerade_ipv4    13203  1 ipt_MASQUERADE
tun                    27106  1
nf_conntrack_netbios_ns    12665  0
nf_conntrack_broadcast    12527  1 nf_conntrack_netbios_ns
ip6t_rpfilter          12546  1
ip6t_REJECT            12625  2
nf_reject_ipv6         13301  1 ip6t_REJECT
xt_conntrack           12760  32
ebtable_nat            12807  0
ebtable_broute         12731  0
bridge                108443  1 ebtable_broute
stp                    12868  1 bridge
llc                    13941  2 stp,bridge
ebtable_filter         12827  0
ebtables               30758  3 ebtable_broute,ebtable_nat,ebtable_filter
```

Linux Kernel Attack Surface



Credit: Jon Oberheide, SOURCE Boston 2010, "Linux Kernel Exploitation"
Reference: https://jon.oberheide.org/files/source10-linuxkernel-jonoberheide.pdf

I/O Control

- Common way to operate a device other than simply read/write
  - “We need a way to format floppies and war dial!”
- Implemented by a *special* type of system call
  - Most *nix use **ioctl()**
  - On Windows it’s **DeviceIoControl()**

I/O Control

- What do kernel devs say about I/O Control?
  - “The ioctl() system call has long been out of favor among the kernel developers, who see it as a **completely uncontrolled entry point into the kernel**[…]”

Reference: http://lwn.net/Articles/119652/

I/O Control

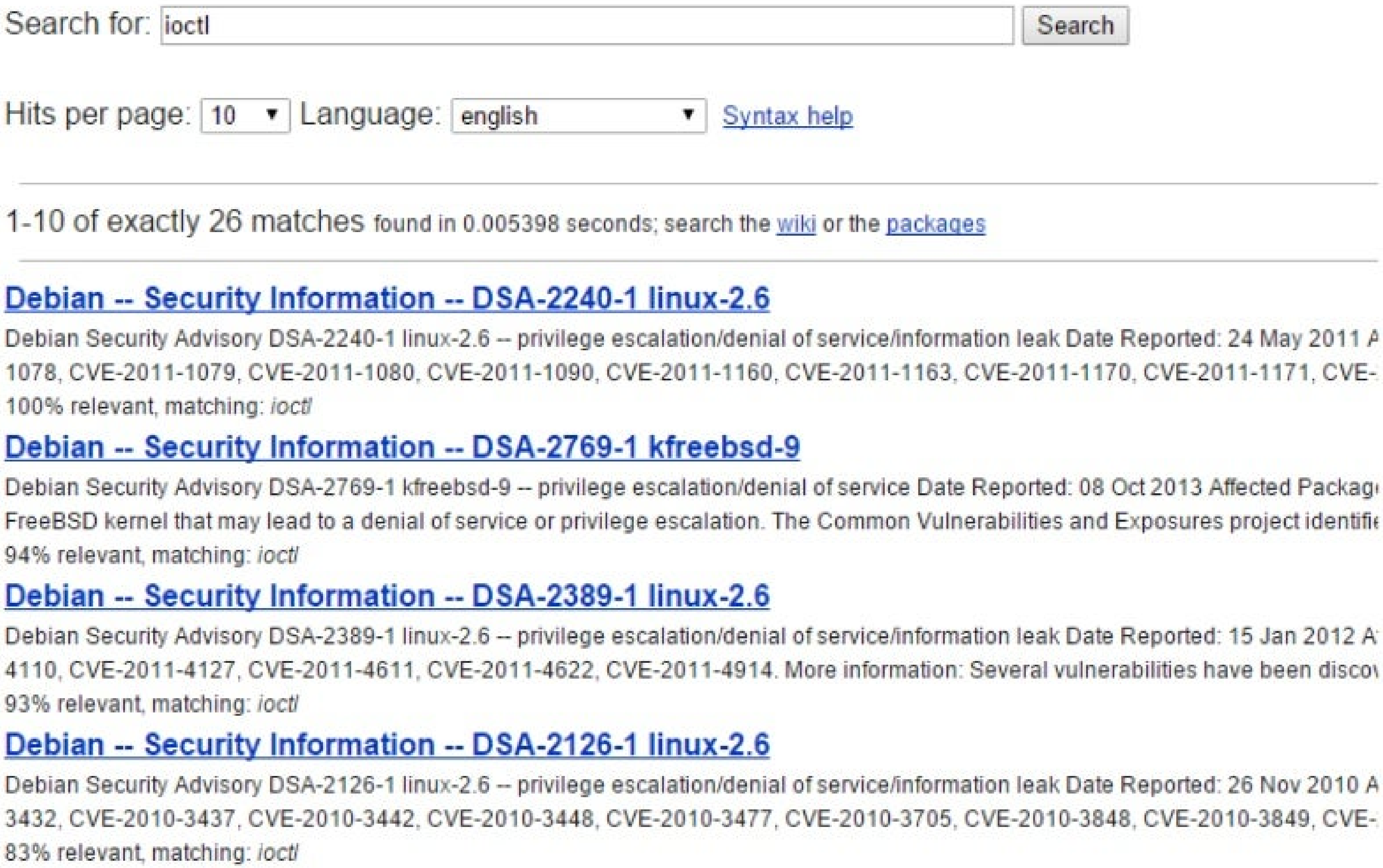
- Continued
  - "Given the vast number of applications which expect ioctl() to be present, however, **it will not go away anytime soon**"

Image: http://www.dailygalaxy.com/.a/6a00d8341bf7f753ef0133ec9e7169970b-pi

A worthy target

- Drivers provide kernel entry points
  - Find bugs in drivers, execute codes in kernel mode
- Various Impacts
  - Privilege escalation
  - Info Leaks
  - Sandbox escapes

Previous Work



Search for: ioctl

Hits per page: 10 ▾ Language: english ▾ Syntax help

1-10 of exactly 26 matches found in 0.005398 seconds; search the wiki or the packages

Debian -- Security Information -- DSA-2240-1 linux-2.6

Debian Security Advisory DSA-2240-1 linux-2.6 -- privilege escalation/denial of service/information leak Date Reported: 24 May 2011 A
1078, CVE-2011-1079, CVE-2011-1080, CVE-2011-1090, CVE-2011-1160, CVE-2011-1163, CVE-2011-1170, CVE-2011-1171, CVE-
100% relevant, matching: *ioctl*

Debian -- Security Information -- DSA-2769-1 kfreebsd-9

Debian Security Advisory DSA-2769-1 kfreebsd-9 -- privilege escalation/denial of service Date Reported: 08 Oct 2013 Affected Packag
FreeBSD kernel that may lead to a denial of service or privilege escalation. The Common Vulnerabilities and Exposures project identifi
94% relevant, matching: *ioctl*

Debian -- Security Information -- DSA-2389-1 linux-2.6

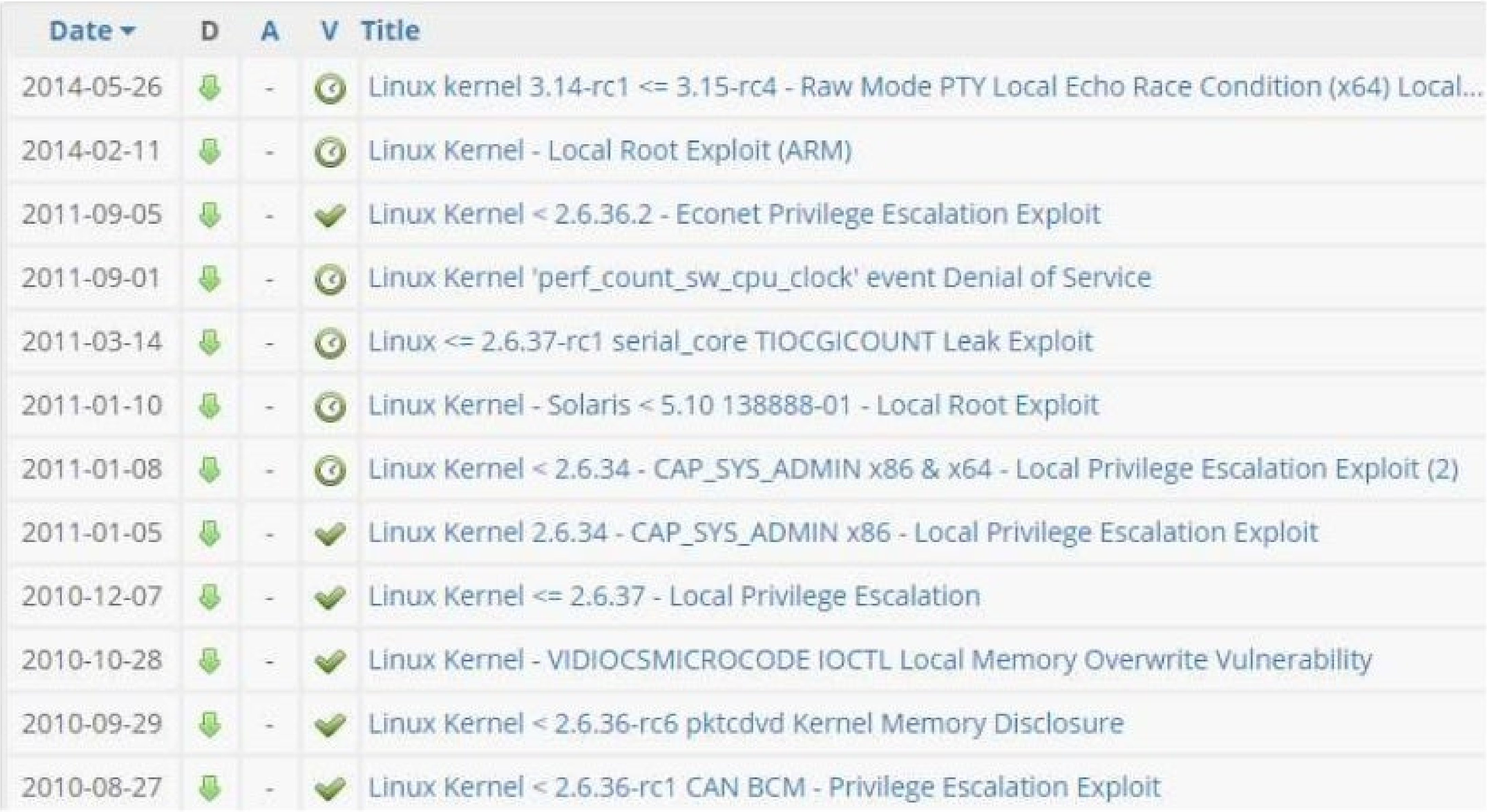
Debian Security Advisory DSA-2389-1 linux-2.6 -- privilege escalation/denial of service/information leak Date Reported: 15 Jan 2012 A
4110, CVE-2011-4127, CVE-2011-4611, CVE-2011-4622, CVE-2011-4914. More information: Several vulnerabilities have been disco
93% relevant, matching: *ioctl*

Debian -- Security Information -- DSA-2126-1 linux-2.6

Debian Security Advisory DSA-2126-1 linux-2.6 -- privilege escalation/denial of service/information leak Date Reported: 26 Nov 2010 A
3432, CVE-2010-3437, CVE-2010-3442, CVE-2010-3448, CVE-2010-3477, CVE-2010-3705, CVE-2010-3848, CVE-2010-3849, CVE-
83% relevant, matching: *ioctl*

Reference: http://search.debian.org/cgi-bin/omega?DB=en&P=ioctl

Previous Work

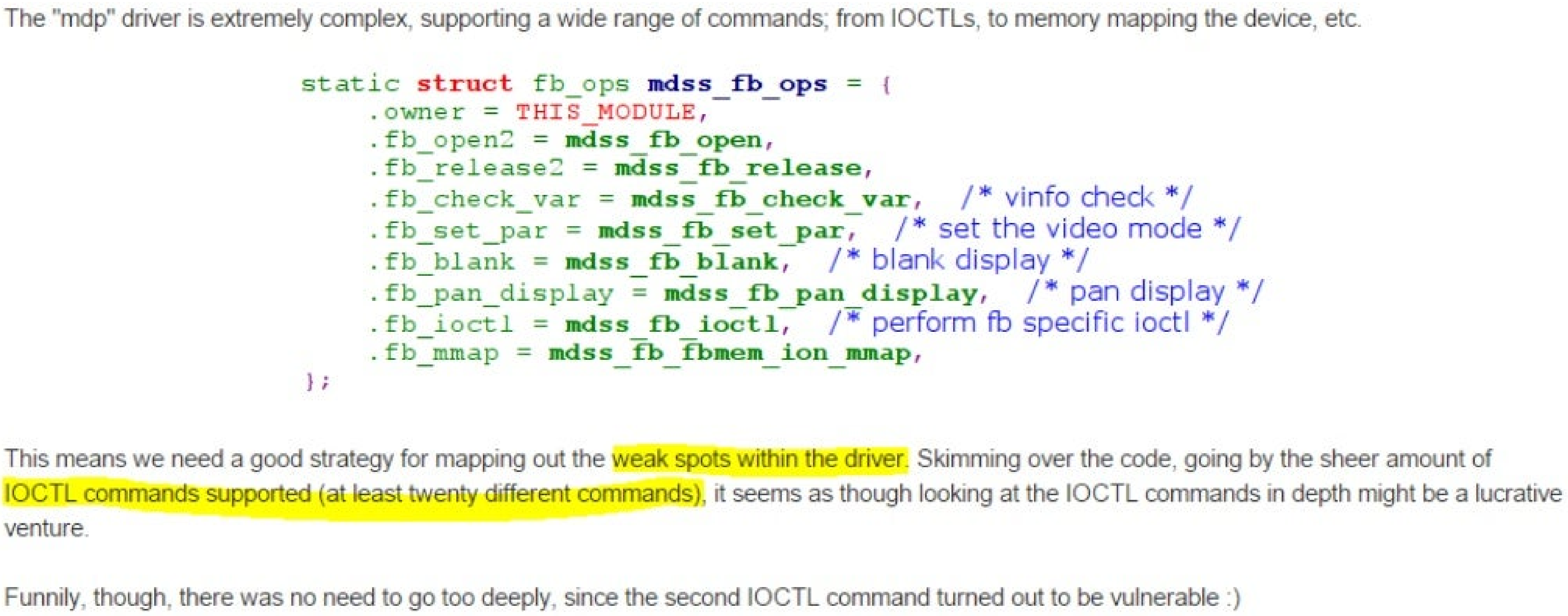


Date ▾	D	A	V	Title
2014-05-26	↓	-	◷	Linux kernel 3.14-rc1 <= 3.15-rc4 - Raw Mode PTY Local Echo Race Condition (x64) Local...
2014-02-11	↓	-	◷	Linux Kernel - Local Root Exploit (ARM)
2011-09-05	↓	-	✓	Linux Kernel < 2.6.36.2 - Econet Privilege Escalation Exploit
2011-09-01	↓	-	◷	Linux Kernel 'perf_count_sw_cpu_clock' event Denial of Service
2011-03-14	↓	-	◷	Linux <= 2.6.37-rc1 serial_core TIOCGICOUNT Leak Exploit
2011-01-10	↓	-	◷	Linux Kernel - Solaris < 5.10 138888-01 - Local Root Exploit
2011-01-08	↓	-	◷	Linux Kernel < 2.6.34 - CAP_SYS_ADMIN x86 & x64 - Local Privilege Escalation Exploit (2)
2011-01-05	↓	-	✓	Linux Kernel 2.6.34 - CAP_SYS_ADMIN x86 - Local Privilege Escalation Exploit
2010-12-07	↓	-	✓	Linux Kernel <= 2.6.37 - Local Privilege Escalation
2010-10-28	↓	-	✓	Linux Kernel - VIDIOCSMICROCODE IOCTL Local Memory Overwrite Vulnerability
2010-09-29	↓	-	✓	Linux Kernel < 2.6.36-rc6 pktcdvd Kernel Memory Disclosure
2010-08-27	↓	-	✓	Linux Kernel < 2.6.36-rc1 CAN BCM - Privilege Escalation Exploit

Reference: https://www.exploit-db.com/search/?action=search&description=linux&text=ioctl

Android Privilege Escalation

- Recent blog detailing CVE-2014-4323



The "mdp" driver is extremely complex, supporting a wide range of commands; from IOCTLs, to memory mapping the device, etc.

```
static struct fb_ops mdss_fb_ops = {
    .owner = THIS_MODULE,
    .fb_open2 = mdss_fb_open,
    .fb_release2 = mdss_fb_release,
    .fb_check_var = mdss_fb_check_var,  /* vinfo check */
    .fb_set_par = mdss_fb_set_par,  /* set the video mode */
    .fb_blank = mdss_fb_blank,  /* blank display */
    .fb_pan_display = mdss_fb_pan_display,  /* pan display */
    .fb_ioctl = mdss_fb_ioctl,  /* perform fb specific ioctl */
    .fb_mmap = mdss_fb_fbmem_ion_mmap,
};
```

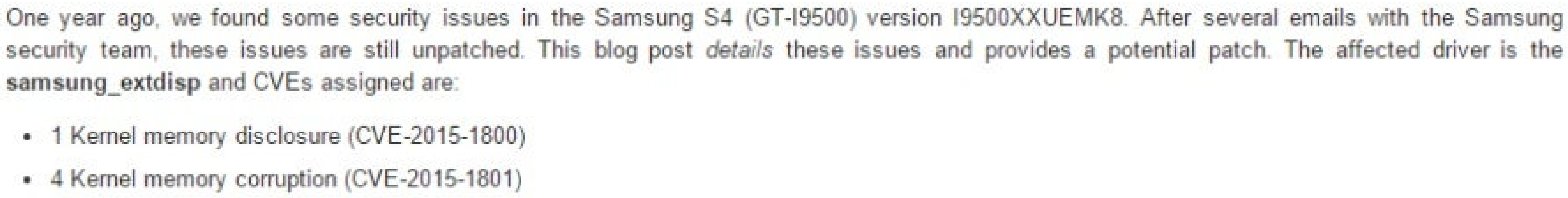
This means we need a good strategy for mapping out the weak spots within the driver. Skimming over the code, going by the sheer amount of IOCTL commands supported (at least twenty different commands), it seems as though looking at the IOCTL commands in depth might be a lucrative venture.

Funnily, though, there was no need to go too deeply, since the second IOCTL command turned out to be vulnerable :)

Reference: http://bits-please.blogspot.com/2015/08/android-linux-kernel-privilege_26.html

Android Privilege Escalation

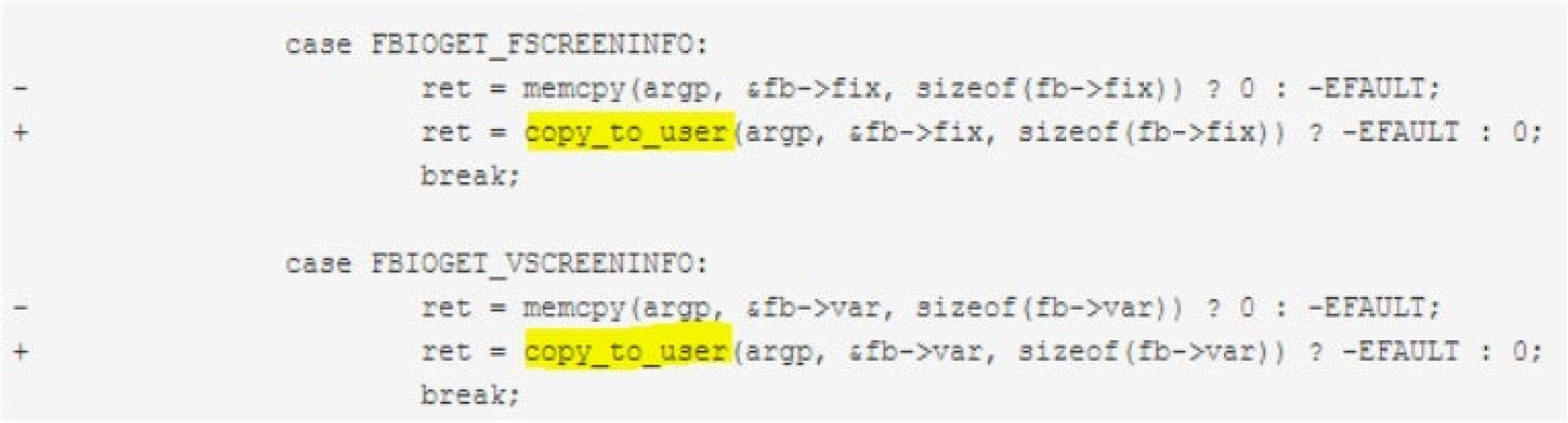
- Vulnerabilities in the Samsung S4



One year ago, we found some security issues in the Samsung S4 (GT-I9500) version I9500XXUEMK8. After several emails with the Samsung security team, these issues are still unpatched. This blog post *details* these issues and provides a potential patch. The affected driver is the **samsung_extdisp** and CVEs assigned are:

- 1 Kernel memory disclosure (CVE-2015-1800)
- 4 Kernel memory corruption (CVE-2015-1801)

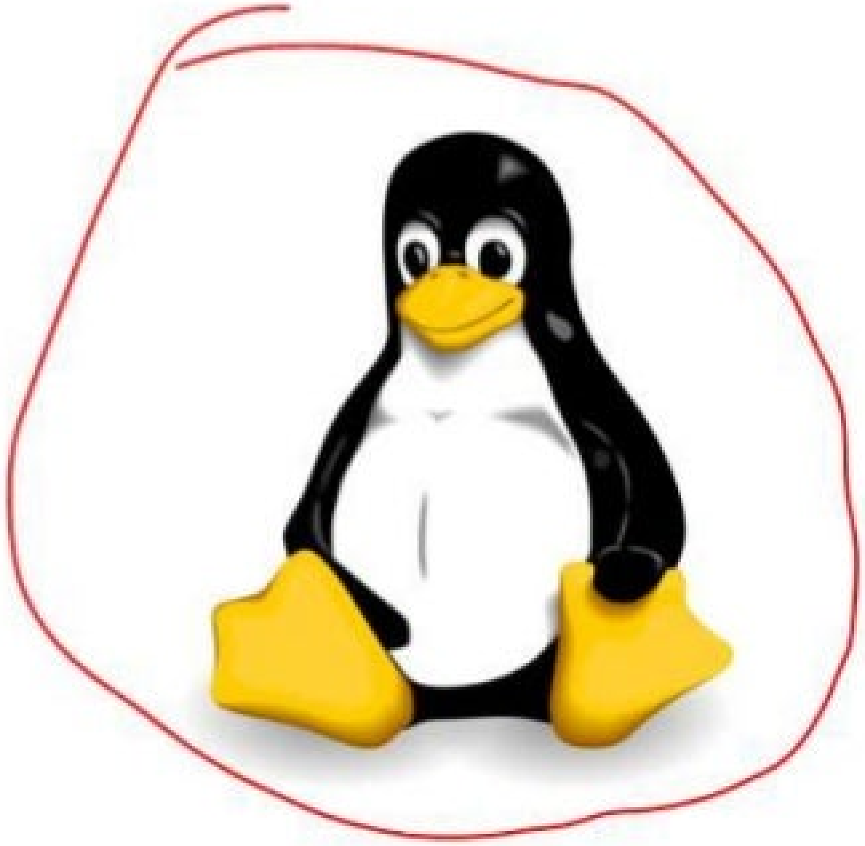
- **memcpy()** doesn’t check user/kernel access!



```
            case FBIOGET_FSCREENINFO:
-                   ret = memcpy(argp, &fb->fix, sizeof(fb->fix)) ? 0 : -EFAULT;
+                   ret = copy_to_user(argp, &fb->fix, sizeof(fb->fix)) ? -EFAULT : 0;
                    break;

            case FBIOGET_VSCREENINFO:
-                   ret = memcpy(argp, &fb->var, sizeof(fb->var)) ? 0 : -EFAULT;
+                   ret = copy_to_user(argp, &fb->var, sizeof(fb->var)) ? -EFAULT : 0;
                    break;
```

Reference: http://blog.quarkslab.com/kernel-vulnerabilities-in-the-samsung-s4.html

Time to Learn



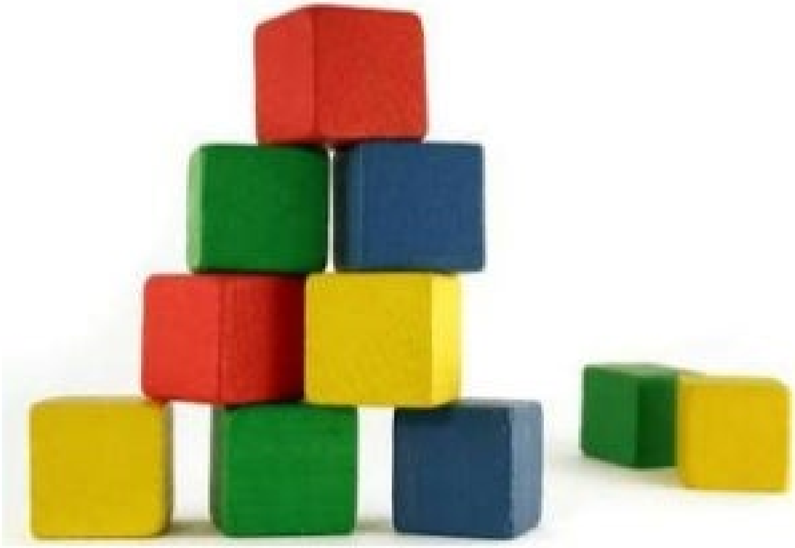
Device types

- Char
  - Character Device
  - Think of it as a “stream of bytes”
  - Mapped to file system nodes, eg. /dev/XXXXX
- Char device vs regular file
  - Sequential and arbitrary access, respectively

Corbet, Jonathan, and Alessandro Rubini. *Linux Device Drivers*. 3rd ed. : O'Reilly, 2005. Print.

Device types

- Block
  - Usually hosts a filesystem
  - Also accessible by filesystem nodes
    - But through a different interface than char devices



Corbet, Jonathan, and Alessandro Rubini. *Linux Device Drivers*. 3rd ed. : O'Reilly, 2005. Print.

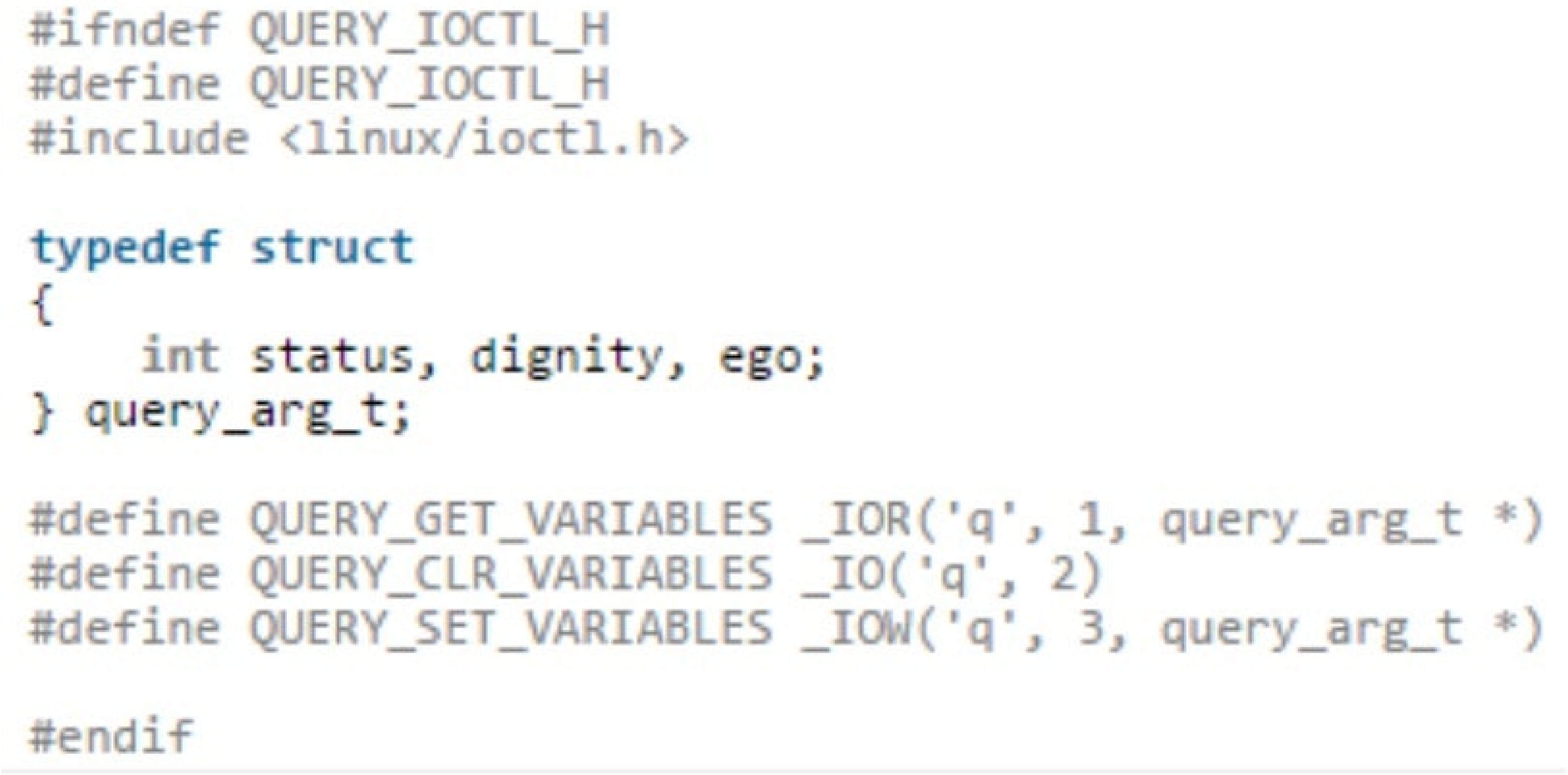
Device types

- Network
  - – Device that exchanges data with hosts
  - – Eg. Loopback, eth0
  - – No traditional read/write
    - Special packet transmission functions are called

Corbet, Jonathan, and Alessandro Rubini. *Linux Device Drivers*. 3rd ed. : O'Reilly, 2005. Print.

Talking to a driver

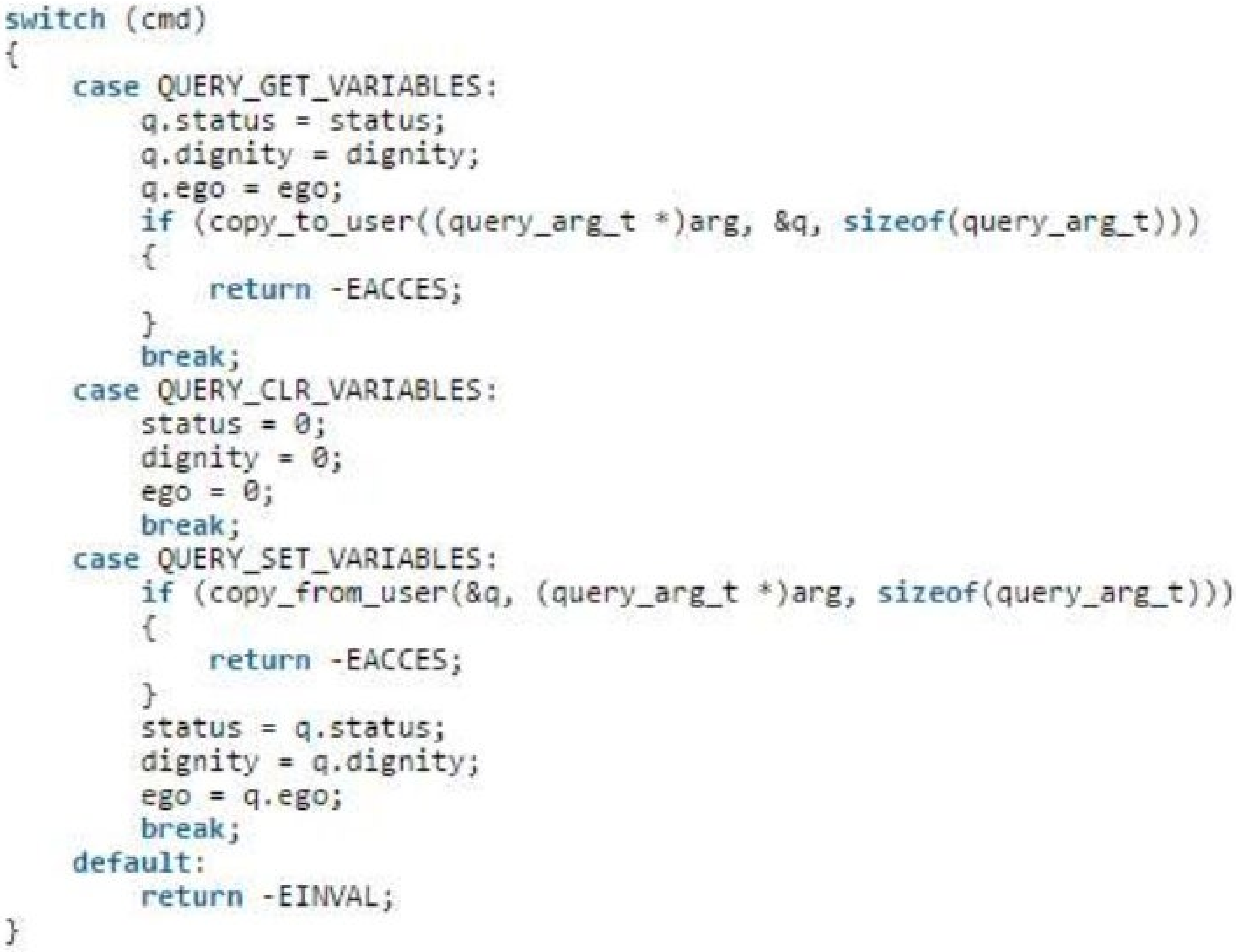
Example driver.c



```
#ifndef QUERY_IOCTL_H
#define QUERY_IOCTL_H
#include <linux/ioctl.h>

typedef struct
{
    int status, dignity, ego;
} query_arg_t;

#define QUERY_GET_VARIABLES _IOR('q', 1, query_arg_t *)
#define QUERY_CLR_VARIABLES _IO('q', 2)
#define QUERY_SET_VARIABLES _IOW('q', 3, query_arg_t *)

#endif
```

Reference: http://www.opensourceforu.com/2011/08/io-control-in-linux/

Example driver.c

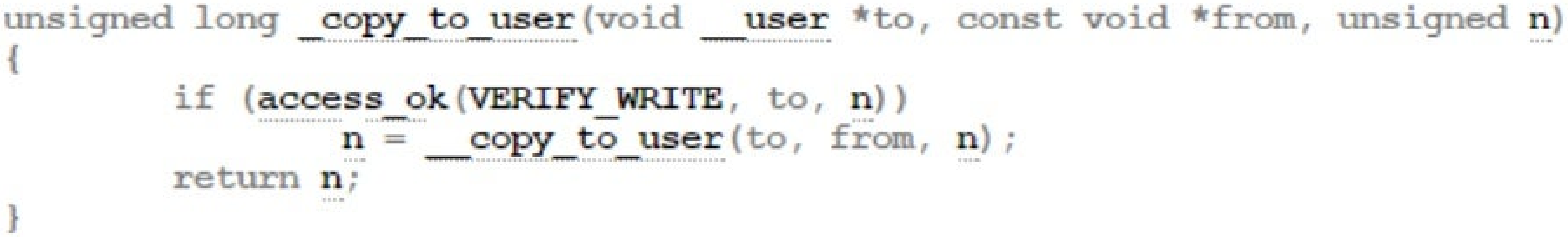


```
switch (cmd)
{
    case QUERY_GET_VARIABLES:
        q.status = status;
        q.dignity = dignity;
        q.ego = ego;
        if (copy_to_user((query_arg_t *)arg, &q, sizeof(query_arg_t)))
        {
            return -EACCES;
        }
        break;
    case QUERY_CLR_VARIABLES:
        status = 0;
        dignity = 0;
        ego = 0;
        break;
    case QUERY_SET_VARIABLES:
        if (copy_from_user(&q, (query_arg_t *)arg, sizeof(query_arg_t)))
        {
            return -EACCES;
        }
        status = q.status;
        dignity = q.dignity;
        ego = q.ego;
        break;
    default:
        return -EINVAL;
}
```

Reference: http://www.opensourceforu.com/2011/08/io-control-in-linux/

copy_to_user()

- Does some overflow checks and then calls
  - **_copy_to_user()**



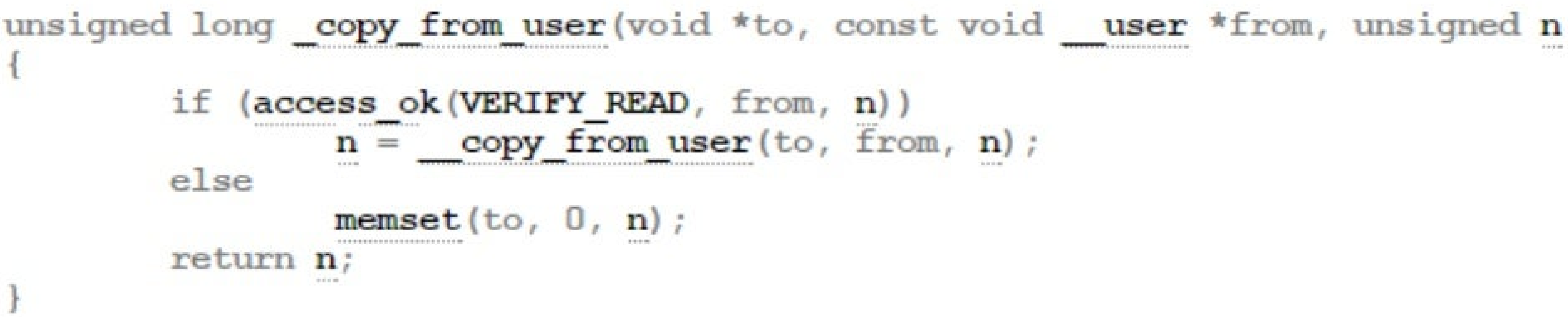
```
unsigned long _copy_to_user(void __user *to, const void *from, unsigned n)
{
        if (access_ok(VERIFY_WRITE, to, n))
                n = __copy_to_user(to, from, n);
        return n;
}
```

- **access_ok** ensures userspace pointer is valid

Reference: http://lxr.free-electrons.com/source/arch/x86/lib/usercopy_32.c#L657

copy_from_user()

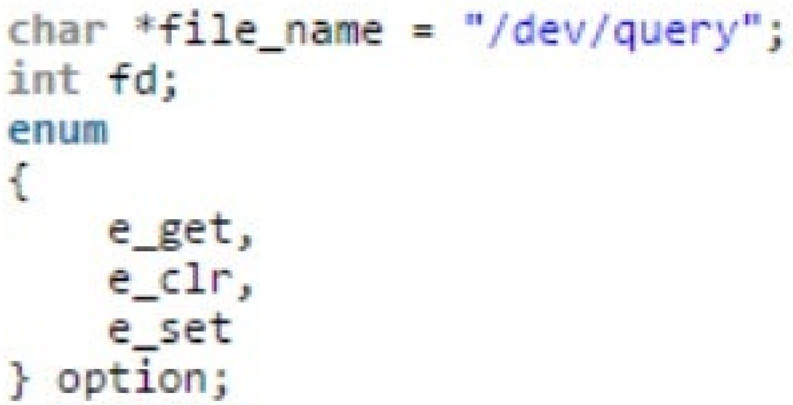
- Calls **_copy_from_user()**



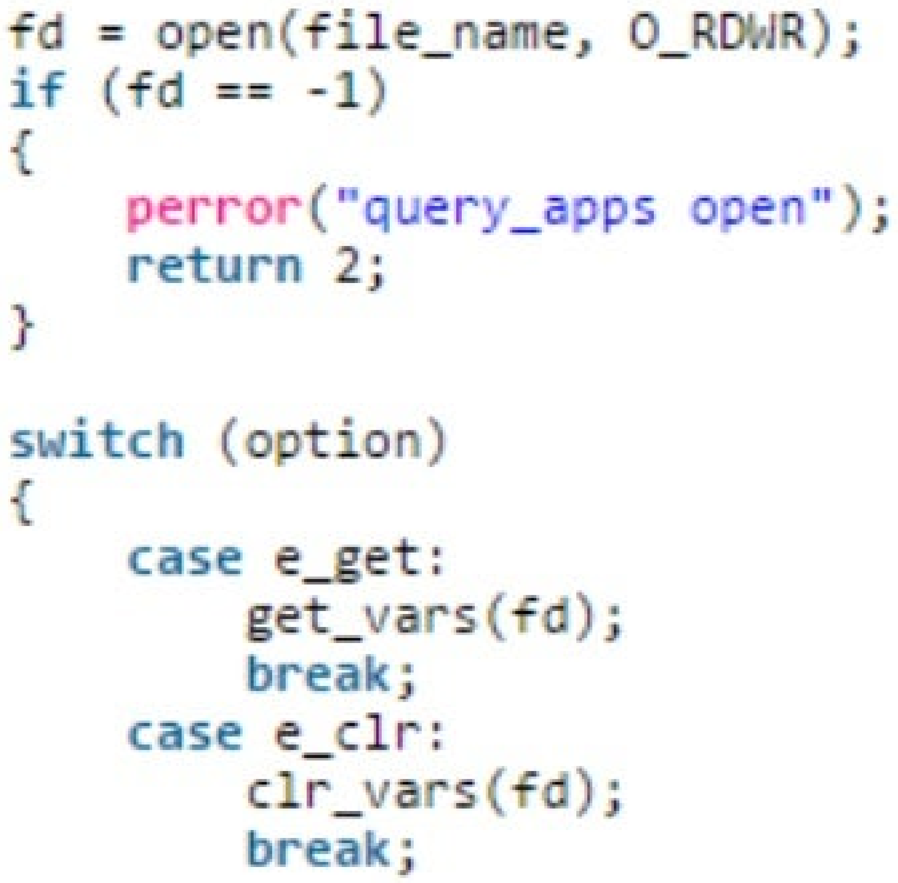
```
unsigned long _copy_from_user(void *to, const void __user *from, unsigned n
{
        if (access_ok(VERIFY_READ, from, n))
                n = __copy_from_user(to, from, n);
        else
                memset(to, 0, n);
        return n;
}
```

- Same idea, but checks what the user specified for **from** instead of **to**
- Also **VERIFY_READ** instead of **VERIFY_WRITE**

Example client.c



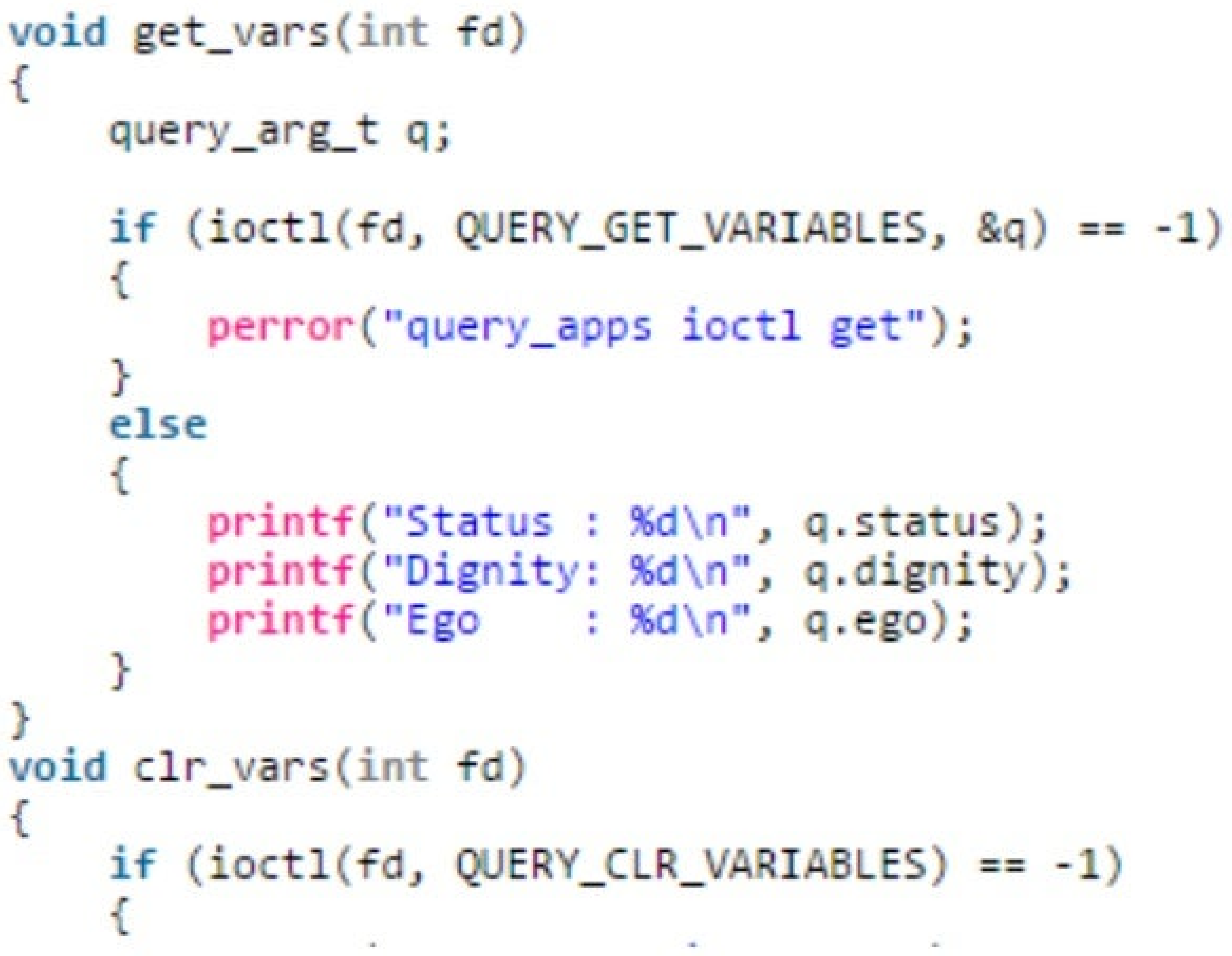
```
char *file_name = "/dev/query";
int fd;
enum
{
    e_get,
    e_clr,
    e_set
} option;
```



```
fd = open(file_name, O_RDWR);
if (fd == -1)
{
    perror("query_apps open");
    return 2;
}

switch (option)
{
    case e_get:
        get_vars(fd);
        break;
    case e_clr:
        clr_vars(fd);
        break;
```

Reference: http://www.opensourceforu.com/2011/08/io-control-in-linux/

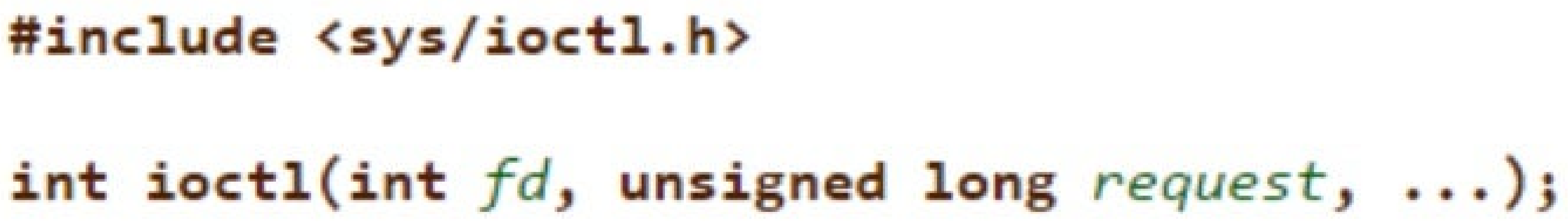
Example client.c



```
void get_vars(int fd)
{
    query_arg_t q;

    if (ioctl(fd, QUERY_GET_VARIABLES, &q) == -1)
    {
        perror("query_apps ioctl get");
    }
    else
    {
        printf("Status : %d\n", q.status);
        printf("Dignity: %d\n", q.dignity);
        printf("Ego    : %d\n", q.ego);
    }
}
void clr_vars(int fd)
{
    if (ioctl(fd, QUERY_CLR_VARIABLES) == -1)
    {
```

Reference: http://www.opensourceforu.com/2011/08/io-control-in-linux/

ioctl()



```
#include <sys/ioctl.h>

int ioctl(int fd, unsigned long request, ...);
```

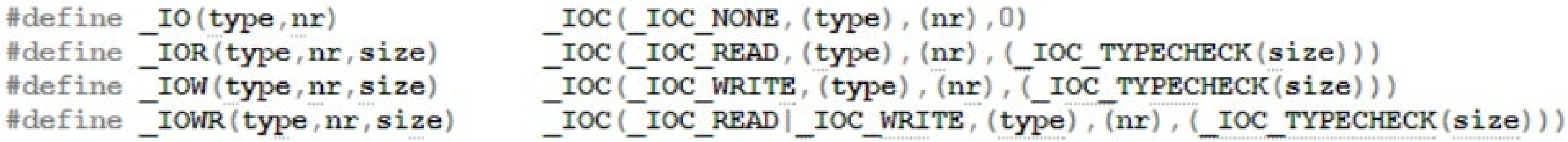
- 1st arg
  - an opened file descriptor for the device
- 2nd arg
  - control code, specific to each operation
- 3rd arg (optional)
  - Buffer to send or receive data

IOCTL Macros

- IO
  - No data transfer
- IOR
  - Read by kernel, write to user
- IOW
  - Read from user, write by kernel
- IORW
  - Bi-directional data transfer

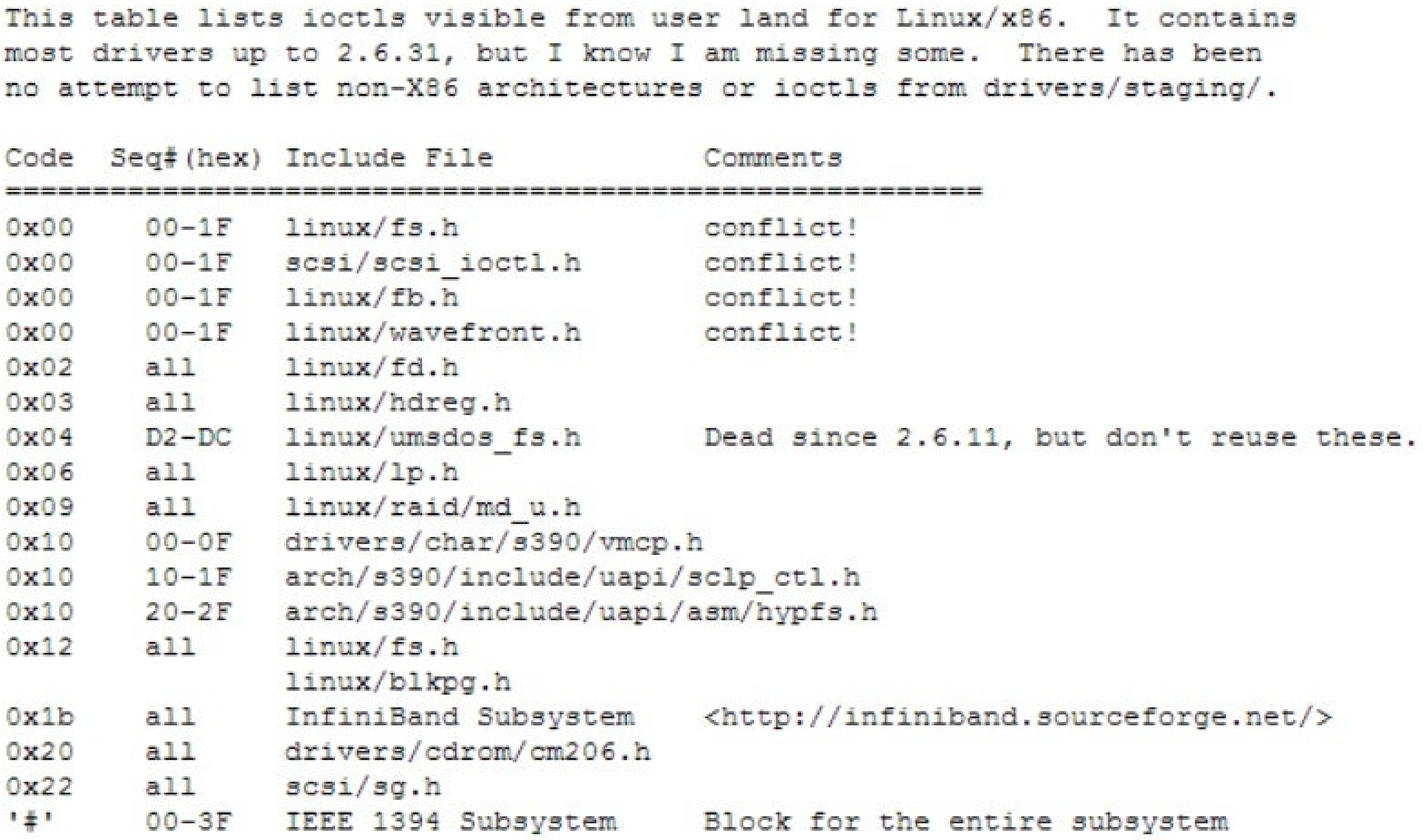
IOCTL Macros

- Four bitfields
  - Type (magic number)
  - Number
  - Direction
  - Size



```
#define _IO(type,nr)            _IOC(_IOC_NONE,(type),(nr),0)
#define _IOR(type,nr,size)      _IOC(_IOC_READ,(type),(nr),(_IOC_TYPECHECK(size)))
#define _IOW(type,nr,size)      _IOC(_IOC_WRITE,(type),(nr),(_IOC_TYPECHECK(size)))
#define _IOWR(type,nr,size)     _IOC(_IOC_READ|_IOC_WRITE,(type),(nr),(_IOC_TYPECHECK(size)))
```

I/O Control



```
This table lists ioctls visible from user land for Linux/x86.  It contains
most drivers up to 2.6.31, but I know I am missing some.  There has been
no attempt to list non-X86 architectures or ioctls from drivers/staging/.

Code  Seq#(hex) Include File            Comments
========================================================
0x00    00-1F   linux/fs.h              conflict!
0x00    00-1F   scsi/scsi_ioctl.h       conflict!
0x00    00-1F   linux/fb.h              conflict!
0x00    00-1F   linux/wavefront.h       conflict!
0x02    all     linux/fd.h
0x03    all     linux/hdreg.h
0x04    D2-DC   linux/umsdos_fs.h       Dead since 2.6.11, but don't reuse these.
0x06    all     linux/lp.h
0x09    all     linux/raid/md_u.h
0x10    00-0F   drivers/char/s390/vmcp.h
0x10    10-1F   arch/s390/include/uapi/sclp_ctl.h
0x10    20-2F   arch/s390/include/uapi/asm/hypfs.h
0x12    all     linux/fs.h
                linux/blkpg.h
0x1b    all     InfiniBand Subsystem    <http://infiniband.sourceforge.net/>
0x20    all     drivers/cdrom/cm206.h
0x22    all     scsi/sg.h
'#'     00-3F   IEEE 1394 Subsystem     Block for the entire subsystem
```

Reference: https://www.kernel.org/doc/Documentation/ioctl/ioctl-number.txt

Agenda

I. Introduction

II. Device Drivers

    I. I/O Control

    II. Talking to a driver

III. Bug Hunting

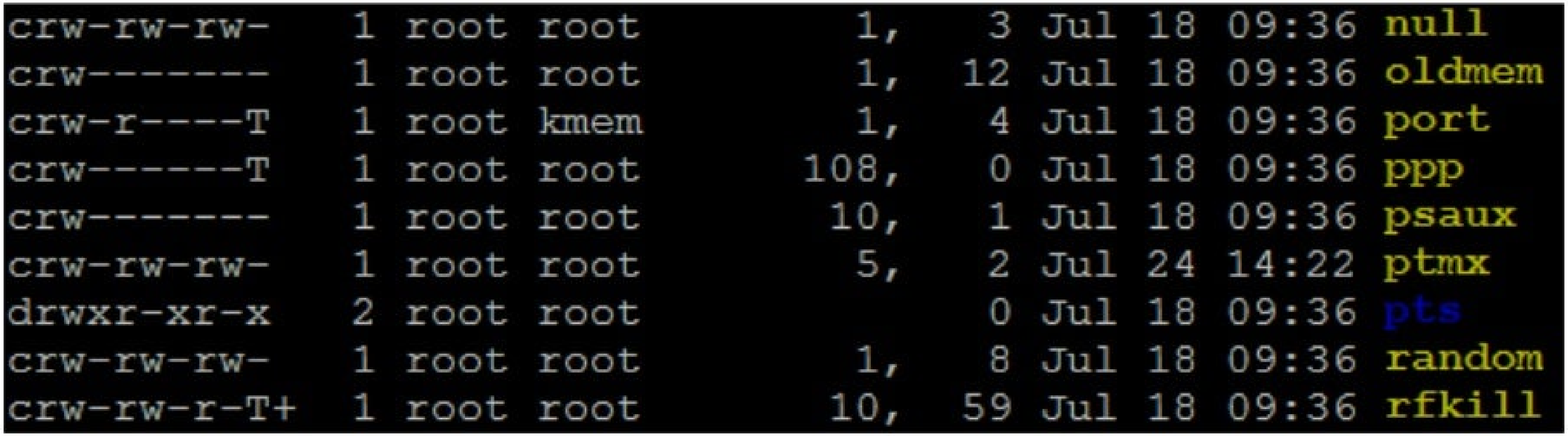
    I. Discovery & Debugging

    II. Tooling

IV. Conclusion

Determining Access

- **/dev/random** is open to everyone
- **ppp** though, not so much



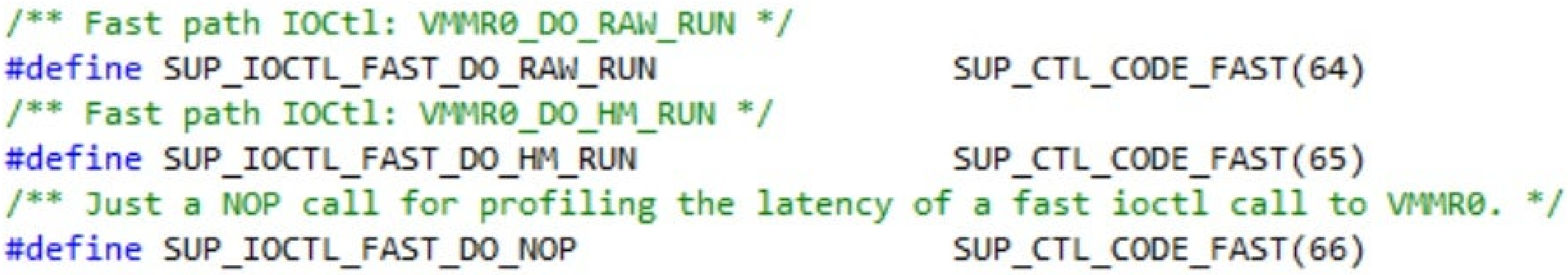
```
crw-rw-rw-   1 root root      1,   3 Jul 18 09:36 null
crw-------   1 root root      1,  12 Jul 18 09:36 oldmem
crw-r----T   1 root kmem      1,   4 Jul 18 09:36 port
crw------T   1 root root    108,   0 Jul 18 09:36 ppp
crw-------   1 root root     10,   1 Jul 18 09:36 psaux
crw-rw-rw-   1 root root      5,   2 Jul 24 14:22 ptmx
drwxr-xr-x   2 root root           0 Jul 18 09:36 pts
crw-rw-rw-   1 root root      1,   8 Jul 18 09:36 random
crw-rw-r-T+  1 root root     10,  59 Jul 18 09:36 rfkill
```

Capabilities

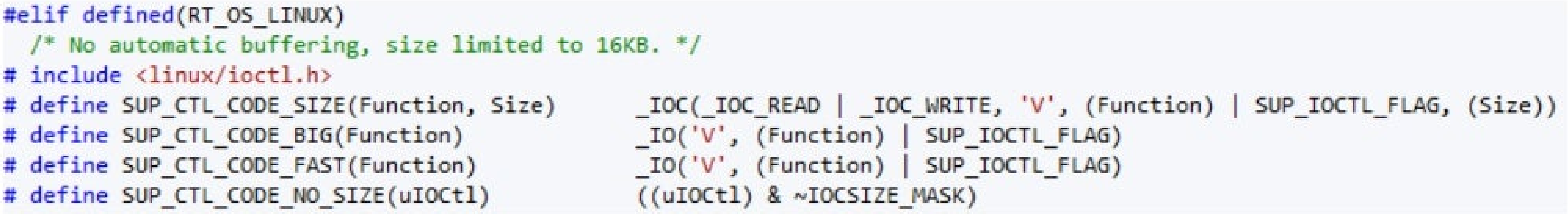
- Intended to split root into different privileges
  - No more all-or-nothing
- ~40 different caps can be applied to binaries
  - Eg. Want to sniff packets with Wireshark?
    - **setcap CAP_NET_RAW+ep /usr/bin/dumpcap**

Finding IOCTLs

- VirtualBox’s SUPDrvIOC.h



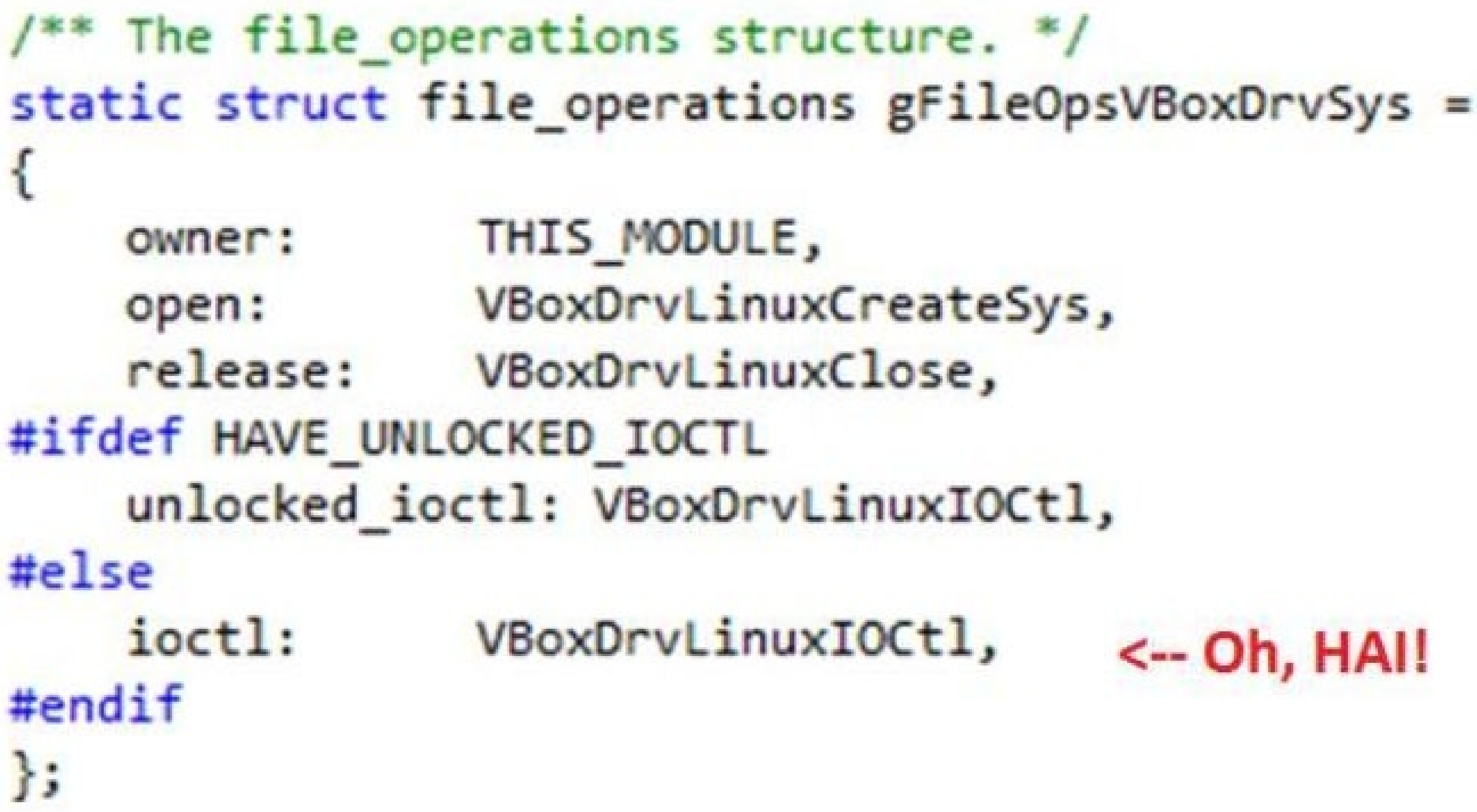
```
/** Fast path IOCtl: VMMR0_DO_RAW_RUN */
#define SUP_IOCTL_FAST_DO_RAW_RUN                       SUP_CTL_CODE_FAST(64)
/** Fast path IOCtl: VMMR0_DO_HM_RUN */
#define SUP_IOCTL_FAST_DO_HM_RUN                        SUP_CTL_CODE_FAST(65)
/** Just a NOP call for profiling the latency of a fast ioctl call to VMMR0. */
#define SUP_IOCTL_FAST_DO_NOP                           SUP_CTL_CODE_FAST(66)
```



```
#elif defined(RT_OS_LINUX)
  /* No automatic buffering, size limited to 16KB. */
# include <linux/ioctl.h>
# define SUP_CTL_CODE_SIZE(Function, Size)      _IOC(_IOC_READ | _IOC_WRITE, 'V', (Function) | SUP_IOCTL_FLAG, (Size))
# define SUP_CTL_CODE_BIG(Function)             _IO('V', (Function) | SUP_IOCTL_FLAG)
# define SUP_CTL_CODE_FAST(Function)            _IO('V', (Function) | SUP_IOCTL_FLAG)
# define SUP_CTL_CODE_NO_SIZE(uIOCtl)           ((uIOCtl) & ~IOCSIZE_MASK)
```

Finding the IOCTL Handler

- VirtualBox's SUPDrv-linux.c

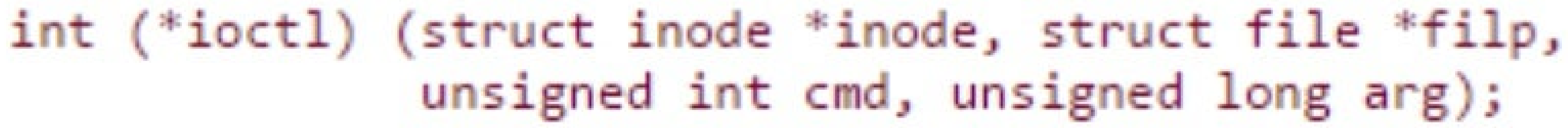


```
/** The file_operations structure. */
static struct file_operations gFileOpsVBoxDrvSys =
{
    owner:      THIS_MODULE,
    open:       VBoxDrvLinuxCreateSys,
    release:    VBoxDrvLinuxClose,
#ifdef HAVE_UNLOCKED_IOCTL
    unlocked_ioctl: VBoxDrvLinuxIOCtl,
#else
    ioctl:      VBoxDrvLinuxIOCtl,
#endif
};
```

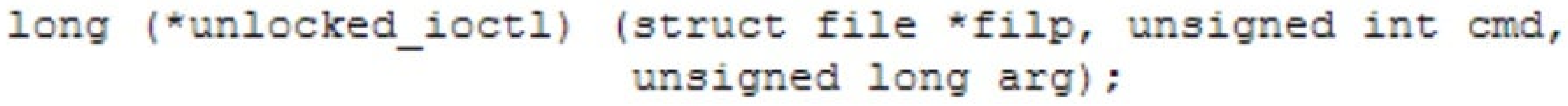
<-- Oh, HAI!

ioctl() vs unlocked_ioctl()

- unlocked_ioctl() was added to kernel 2.6.11
  - IOCTLs no longer use the old BKL (Big Kernel **Lock**)
  - One less parameter (inode)
  - Not important for bug hunting, but now you know



```
int (*ioctl) (struct inode *inode, struct file *filp,
              unsigned int cmd, unsigned long arg);
```

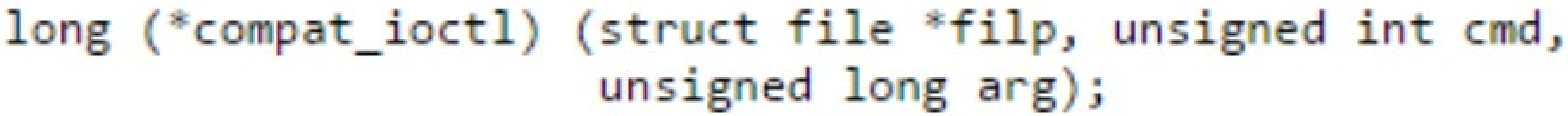


```
long (*unlocked_ioctl) (struct file *filp, unsigned int cmd,
                        unsigned long arg);
```

References: http://lwn.net/Articles/119652/
http://www.makelinux.net/ldd3/chp-6-sect-1

compat_ioctl()

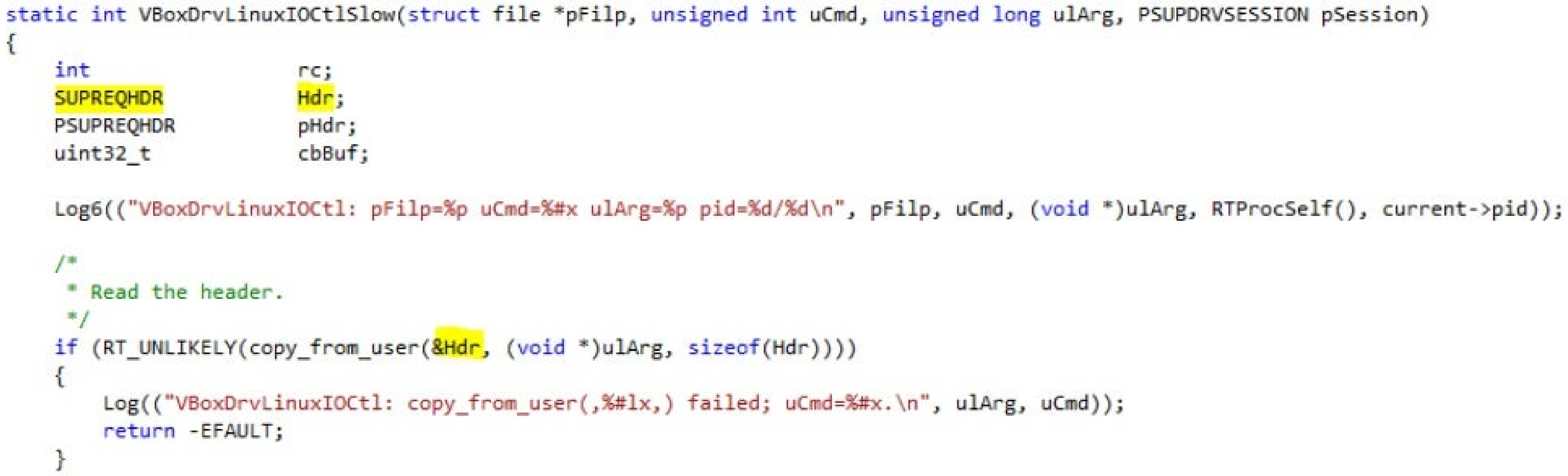
- Handles 32-bit processes calling ioctl() on 64-bit platforms
- Added in same kernel as unlocked_ioctl()



```
long (*compat_ioctl) (struct file *filp, unsigned int cmd,
                      unsigned long arg);
```

Identify buffer types

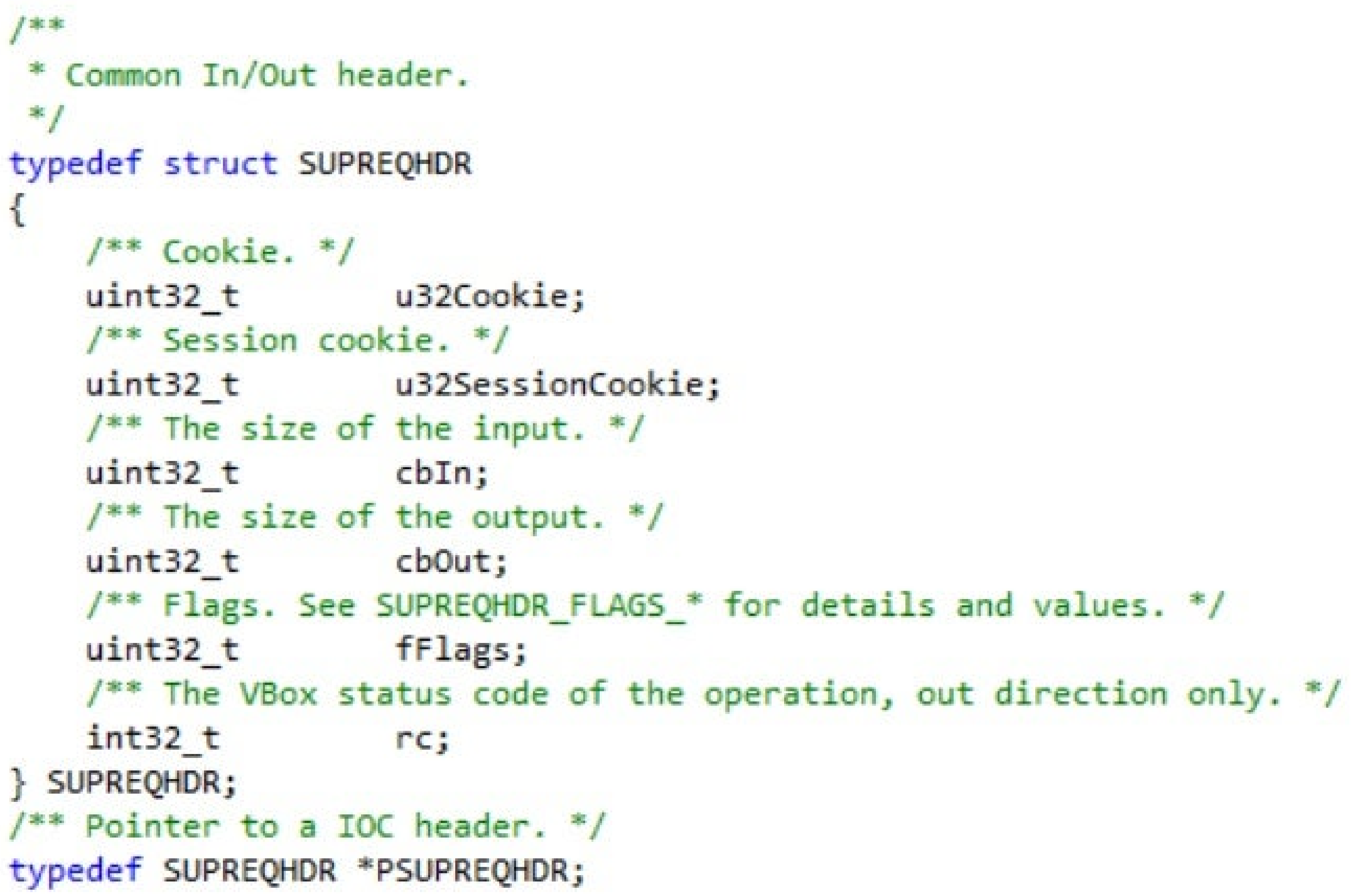
- VirtualBox's SUPDrv-linux.c



```
static int VBoxDrvLinuxIOCtlSlow(struct file *pFilp, unsigned int uCmd, unsigned long ulArg, PSUPDRVSESSION pSession)
{
    int                     rc;
    SUPREQHDR               Hdr;
    PSUPREQHDR              pHdr;
    uint32_t                cbBuf;

    Log6(("VBoxDrvLinuxIOCtl: pFilp=%p uCmd=%#x ulArg=%p pid=%d/%d\n", pFilp, uCmd, (void *)ulArg, RTProcSelf(), current->pid));

    /*
     * Read the header.
     */
    if (RT_UNLIKELY(copy_from_user(&Hdr, (void *)ulArg, sizeof(Hdr))))
    {
        Log(("VBoxDrvLinuxIOCtl: copy_from_user(,%#lx,) failed; uCmd=%#x.\n", ulArg, uCmd));
        return -EFAULT;
    }
```

Identify buffer types

- Vbox’s SUPDrvIOC.h



```
/**
 * Common In/Out header.
 */
typedef struct SUPREQHDR
{
    /** Cookie. */
    uint32_t        u32Cookie;
    /** Session cookie. */
    uint32_t        u32SessionCookie;
    /** The size of the input. */
    uint32_t        cbIn;
    /** The size of the output. */
    uint32_t        cbOut;
    /** Flags. See SUPREQHDR_FLAGS_* for details and values. */
    uint32_t        fFlags;
    /** The VBox status code of the operation, out direction only. */
    int32_t         rc;
} SUPREQHDR;
/** Pointer to a IOC header. */
typedef SUPREQHDR *PSUPREQHDR;
```

CHINA

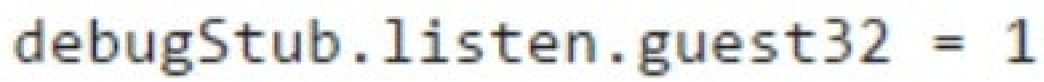
Kernel Debugging

- Host
  - Mac OS X (Yosemite)
  - VMware Fusion
- Guest
  - Ubuntu 14.04 (x86)

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

Kernel Debugging

- Configure the host
  – Copy the ISO, create a new VM, install guest OS
  – Edit the VMX config
    - (right-click and show package contents on Mac)
  – Add this line to the end of the file



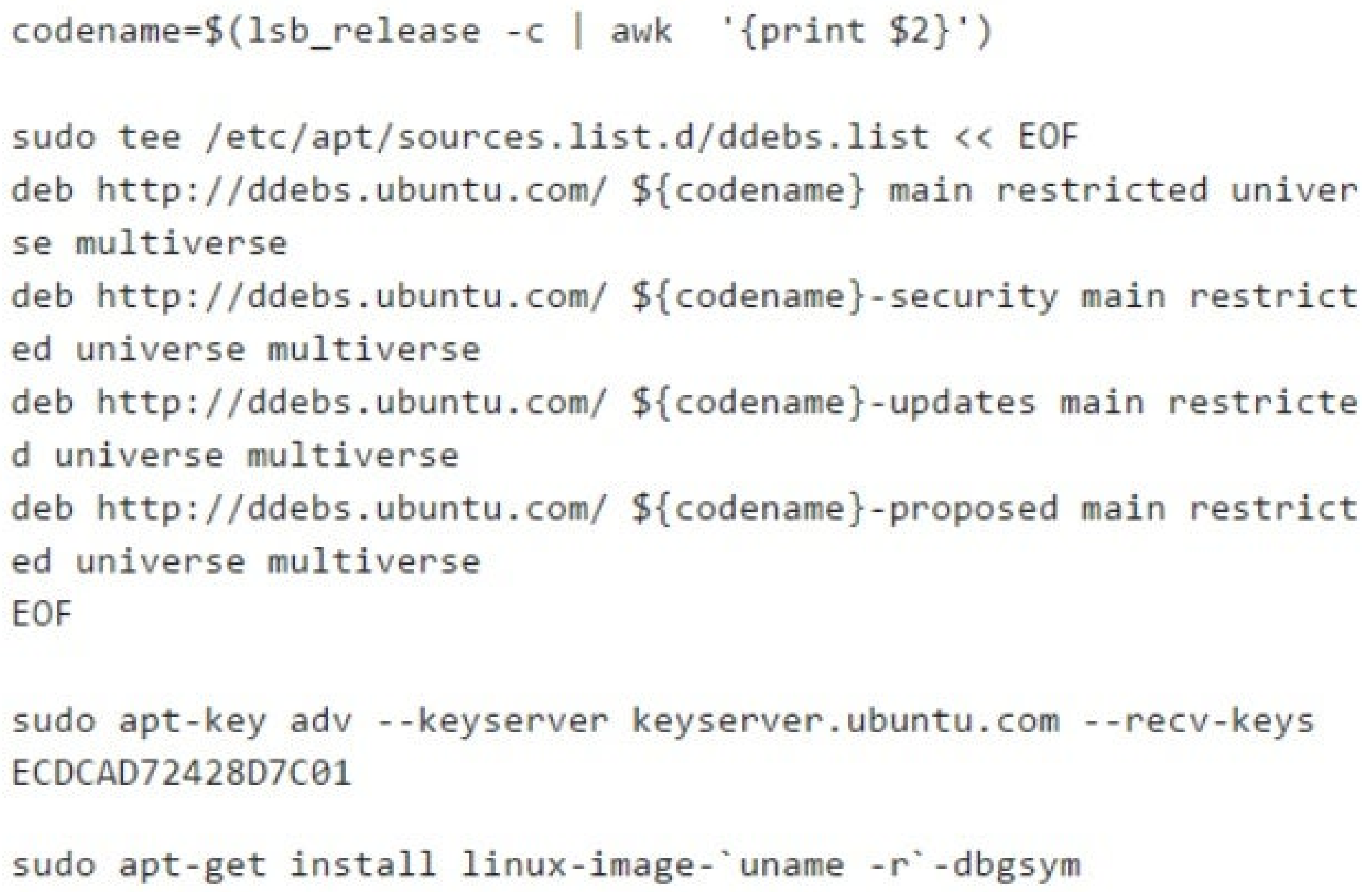
```
debugStub.listen.guest32 = 1
```

  – Boot up guest VM

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

Kernel Debugging

- Configure the guest



```
codename=$(lsb_release -c | awk  '{print $2}')

sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
deb http://ddebs.ubuntu.com/ ${codename} main restricted univer
se multiverse
deb http://ddebs.ubuntu.com/ ${codename}-security main restrict
ed universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-updates main restricte
d universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-proposed main restrict
ed universe multiverse
EOF

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
ECDCAD72428D7C01

sudo apt-get install linux-image-`uname -r`-dbgsym
```

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

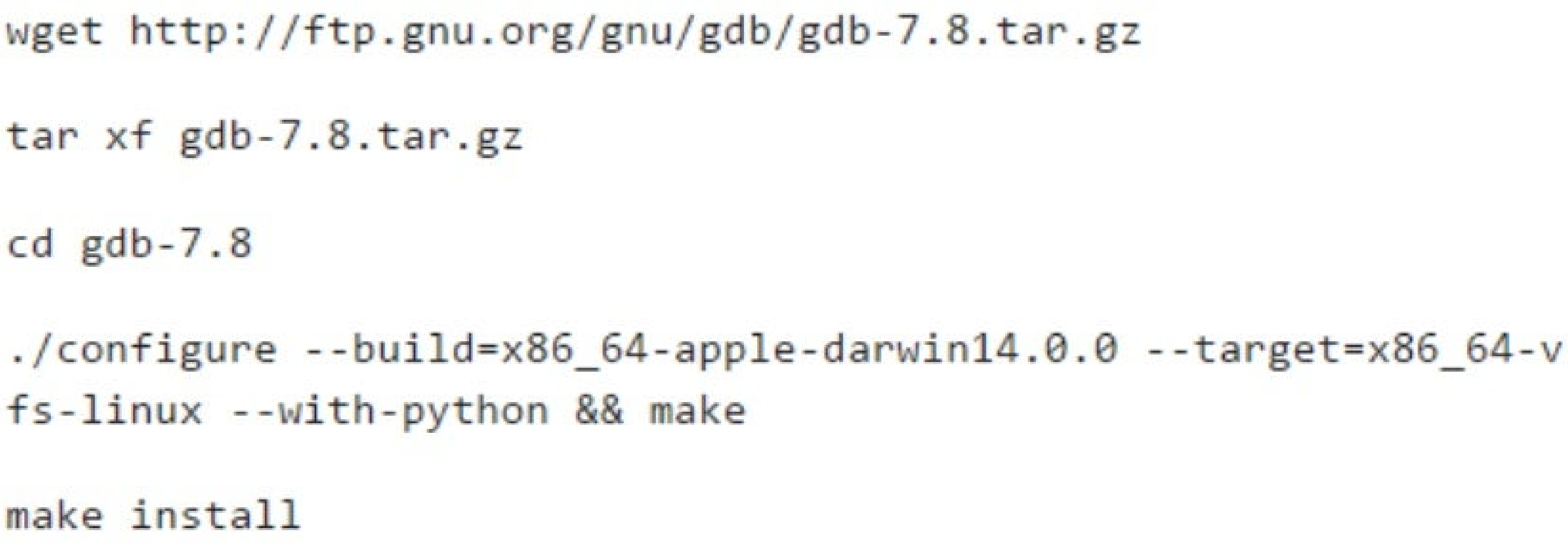
Kernel Debugging

- Configure the guest
  - Copy the debug kernel to the host
    - /usr/lib/debug/boot/**vmlinux-<kernel version>-generic**

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

Kernel Debugging

- GDB

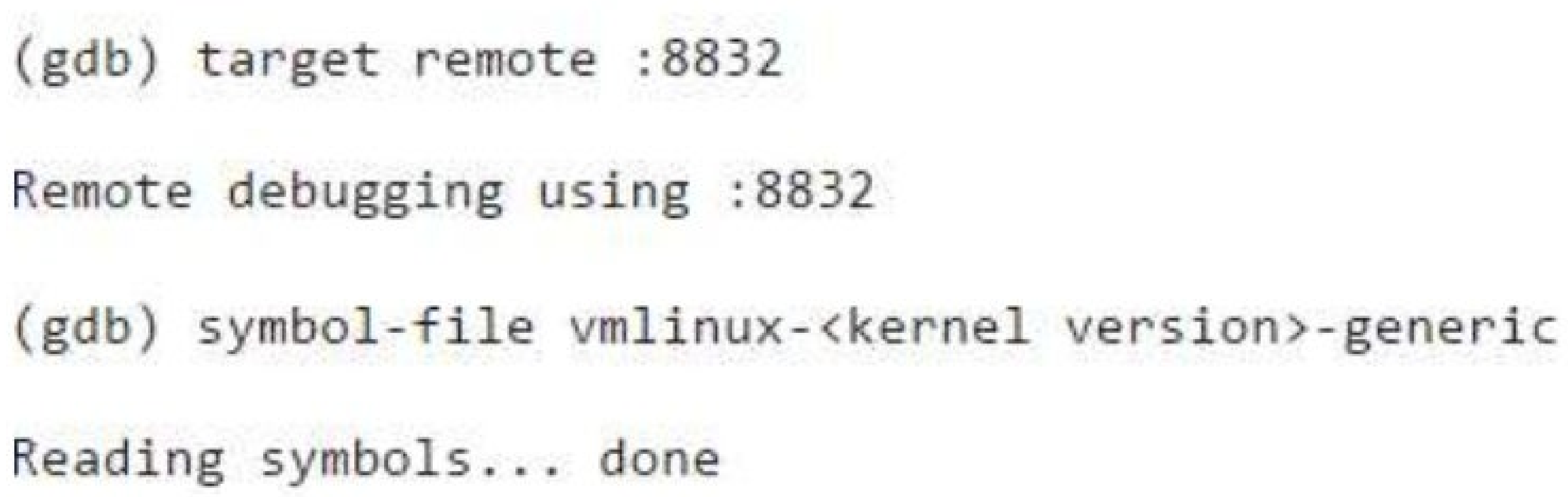


```
wget http://ftp.gnu.org/gnu/gdb/gdb-7.8.tar.gz

tar xf gdb-7.8.tar.gz

cd gdb-7.8

./configure --build=x86_64-apple-darwin14.0.0 --target=x86_64-v
fs-linux --with-python && make

make install
```

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

Kernel Debugging

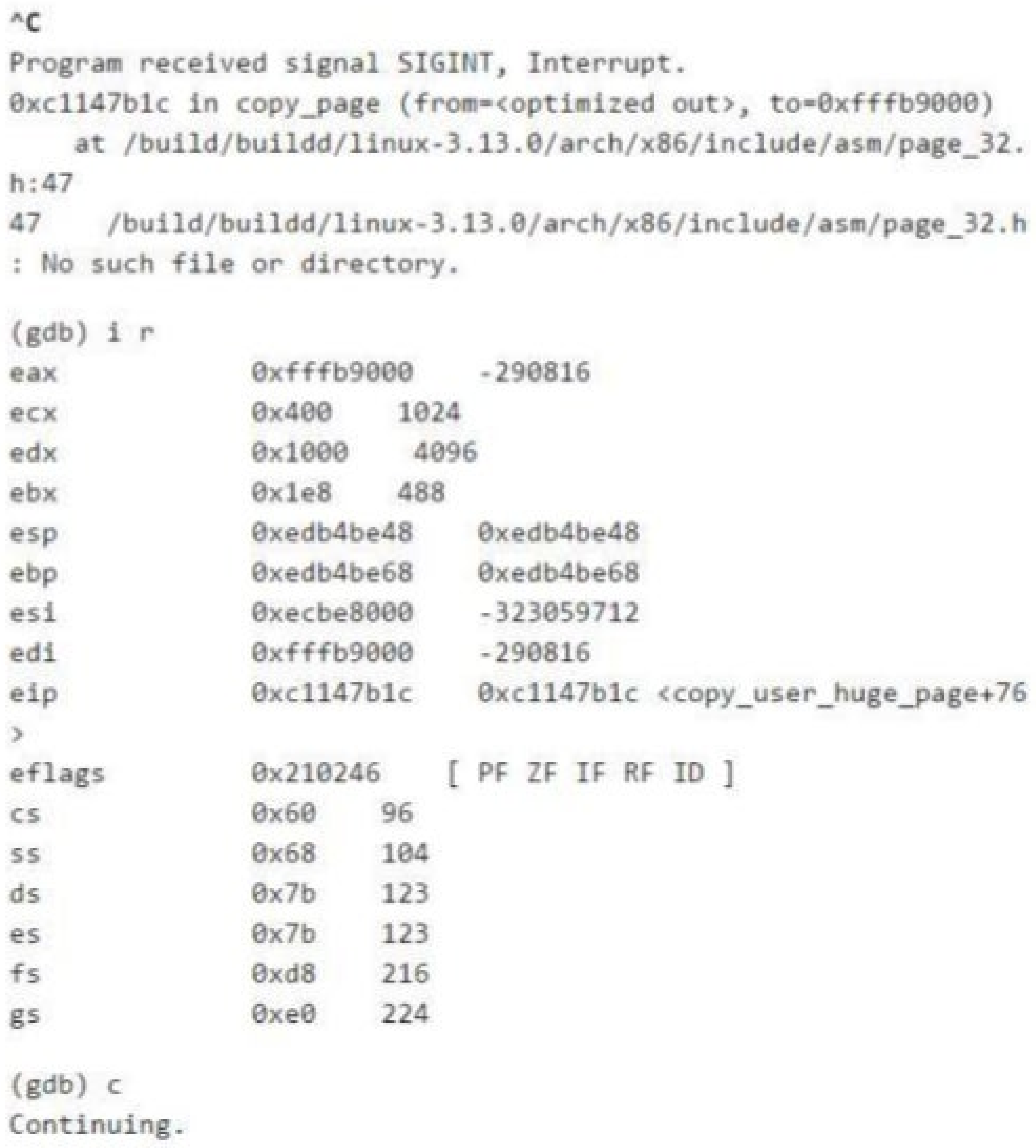
- GDB



```
(gdb) target remote :8832

Remote debugging using :8832

(gdb) symbol-file vmlinux-<kernel version>-generic

Reading symbols... done
```

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

Kernel Debugging



```
^C
Program received signal SIGINT, Interrupt.
0xc1147b1c in copy_page (from=<optimized out>, to=0xfffb9000)
    at /build/buildd/linux-3.13.0/arch/x86/include/asm/page_32.
h:47
47      /build/buildd/linux-3.13.0/arch/x86/include/asm/page_32.h
: No such file or directory.

(gdb) i r
eax            0xfffb9000    -290816
ecx            0x400    1024
edx            0x1000    4096
ebx            0x1e8    488
esp            0xedb4be48    0xedb4be48
ebp            0xedb4be68    0xedb4be68
esi            0xecbe8000    -323059712
edi            0xfffb9000    -290816
eip            0xc1147b1c    0xc1147b1c <copy_user_huge_page+76
>
eflags         0x210246    [ PF ZF IF RF ID ]
cs             0x60    96
ss             0x68    104
ds             0x7b    123
es             0x7b    123
fs             0xd8    216
gs             0xe0    224

(gdb) c
Continuing.
```

More details: http://jidanhunter.blogspot.com/2015/01/linux-kernel-debugging-with-vmware-and.html

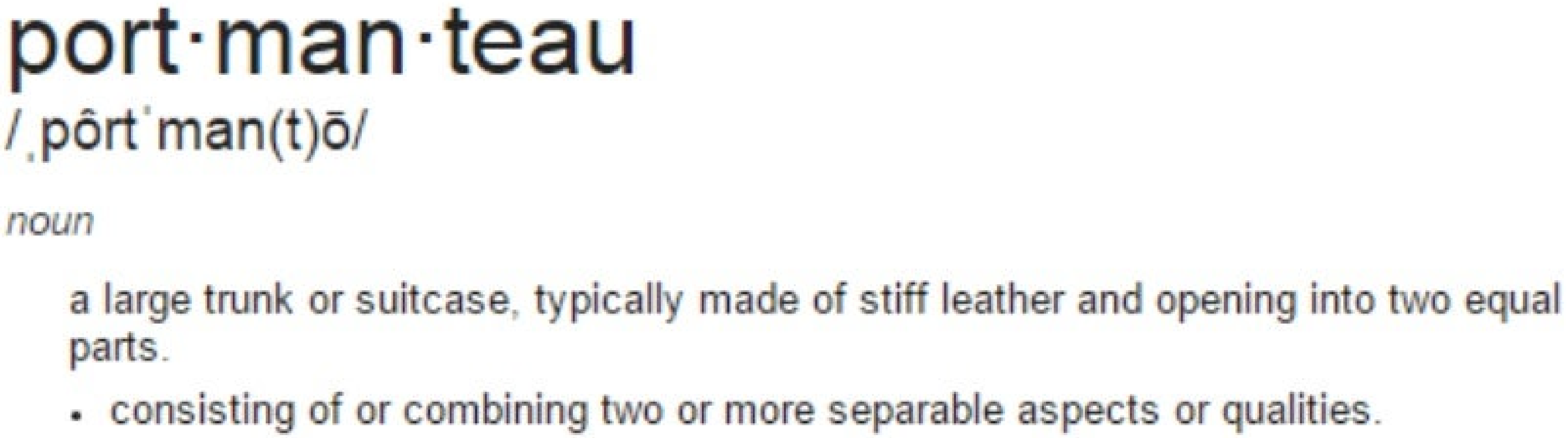
Kernel Debugging

- Intentionally trigger a kernel panic
  - **echo c > /proc/sysrq-trigger**

```
usbcore: registered new driver hiddev
usbcore: registered new driver usbhid
drivers/usb/input/hid-core.c: v2.6:USB HID     re driver
PNP: PS/2 Controller [PNP0303:KBC,PNP0F13:M     ] at 0x60,0x64      1,12
serio: i8042 AUX port at 0x60,0x64 irq 12
serio: i8042 KBD port at 0x60,0x64 irq 1
mice: PS/2 mouse device common for all mice
md: md driver 0.90.3 MAX_MD_DEVS=256, MD_SB_DISKS=27
md: bitmap version 4.39
TCP bic registered
Initializing IPsec netlink socket
NET: Registered protocol family 1
NET: Registered protocol family 17
Using IPI No-Shortcut mode
Time: tsc clocksource has been installed.
ACPI: (supports S0 S1 S5)
md: Autodetecting RAID arrays.
md: autorun ...
md: ... autorun DONE.
RAMDISK: Compressed image found at block 0
invalid compressed format (err=2)
VFS: Cannot open root device "<NULL>" or unkn
Please append a correct "root=" boot option
Kernel panic - not syncing: VFS: Unable to mount
```

Image: http://kernelpanicoggcast.net/images/KernelPanic.png

Tooling



port·man·teau
/ˌpôrtˈman(t)ō/

noun

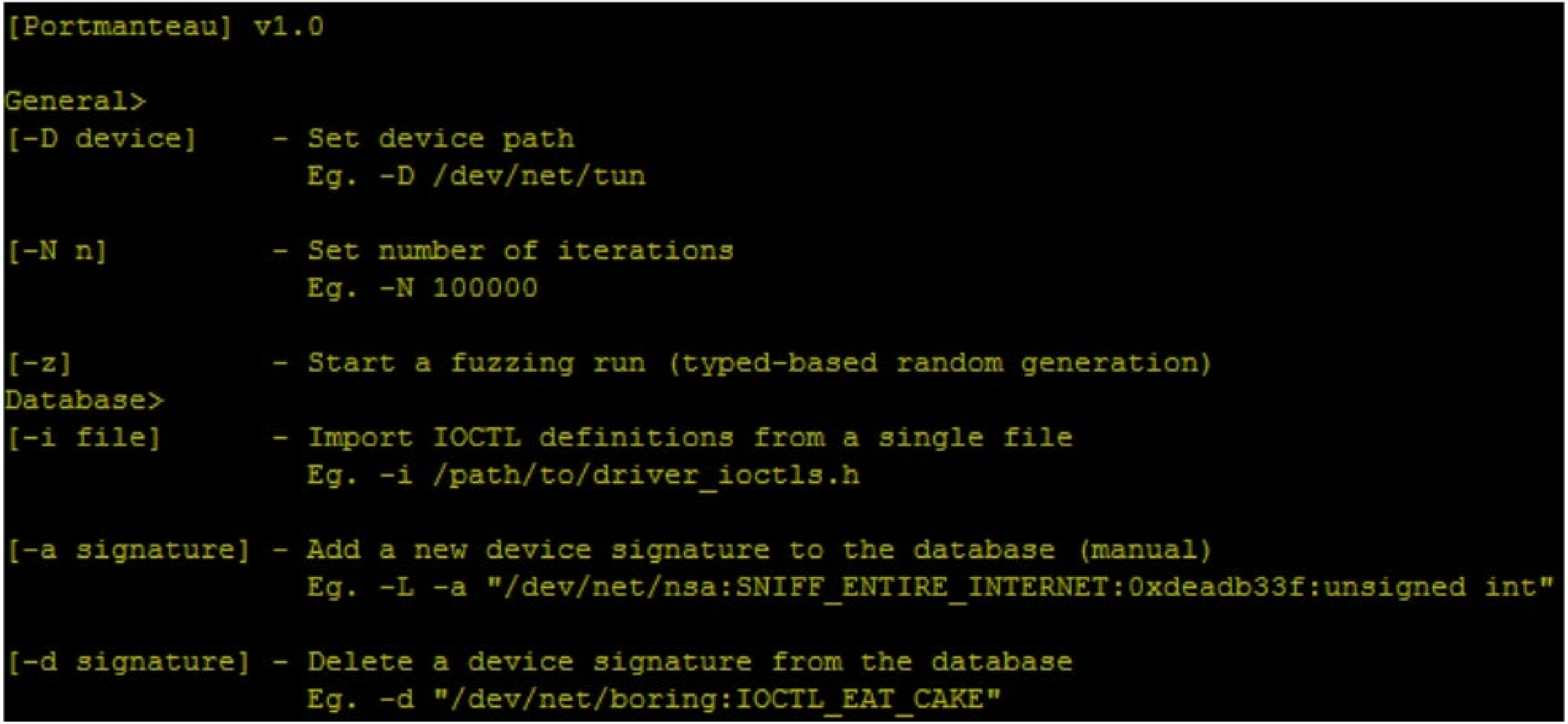
a large trunk or suitcase, typically made of stiff leather and opening into two equal parts.

- consisting of or combining two or more separable aspects or qualities.

Tooling

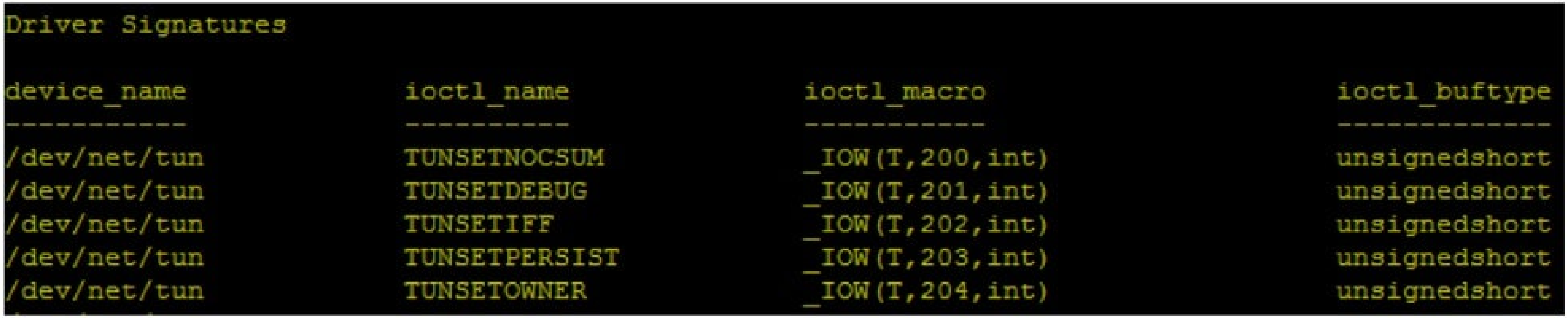
- Experimental *nix tooling
  - Runs on Linux, but easily ported to other unix
- Highlights
  - Static code-based discovery and import of IOCTLs
  - SQLite database support
  - Generational fuzzing
  - Auto-generates PoCs

Tooling



```
[Portmanteau] v1.0

General>
[-D device]     - Set device path
                  Eg. -D /dev/net/tun

[-N n]          - Set number of iterations
                  Eg. -N 100000

[-z]            - Start a fuzzing run (typed-based random generation)
Database>
[-i file]       - Import IOCTL definitions from a single file
                  Eg. -i /path/to/driver_ioctls.h

[-a signature] - Add a new device signature to the database (manual)
                  Eg. -L -a "/dev/net/nsa:SNIFF_ENTIRE_INTERNET:0xdeadb33f:unsigned int"

[-d signature] - Delete a device signature from the database
                  Eg. -d "/dev/net/boring:IOCTL_EAT_CAKE"
```

Tooling



```
Driver Signatures

device_name              ioctl_name               ioctl_macro                      ioctl_buftype
-----------              ----------               -----------                      -------------
/dev/net/tun             TUNSETNOCSUM             _IOW(T,200,int)                  unsignedshort
/dev/net/tun             TUNSETDEBUG              _IOW(T,201,int)                  unsignedshort
/dev/net/tun             TUNSETIFF                _IOW(T,202,int)                  unsignedshort
/dev/net/tun             TUNSETPERSIST            _IOW(T,203,int)                  unsignedshort
/dev/net/tun             TUNSETOWNER              _IOW(T,204,int)                  unsignedshort
```

Tooling

- Will make it available after the talk
  - Ping me if you’d like it sooner

Agenda

I. Introduction

II. Device drivers

  I. I/O Control

  II. Talking to a driver

III. Bug Hunting

  I. Discovery & Debugging

  II. Tooling

IV. Conclusion

Conclusion

- Drivers are a growing area for vuln research
  - Kernel code is becoming a more attractive target
  - Impact grows larger per app dependencies
- Fundamentals and tooling can help
- There’s going to be bugs here for a long time
  - Think “forgotten syscalls”

Future Work

- Targeting abstraction layers
  - Eg. LibUSB
- Binary Analysis
  - IDA scripts that find & document IOCTL calls
- Other platforms
  - Diversity of bugs can vary based on how the kernel works

Thank you!

Questions?