Kernel Tales

Testing security of Kernels for Android or embedded devices by fuzzing drivers (on aarch64)

Vito Rallo
@vitorallo

vito.rallo@gmail.com

I work for:

pwc

Agenda

- Introduction to vulnerability research in the kernel; why?
- Analysis of Android embedded devices (supply chain);
- Security considerations for Android and IoT devices in general;
- Threat modelling;
- Methodology;
- Trinity, or better to say: Dronity!
- Practical details;
- Setup your toolchain;
- Navigate and understand the source code;
- Little hacks, cross-compile, deploy, enjoy!
- Compile and run dronity... fuzz fuzz fuzz
- Fuzzing on the ODROID BOARD
- Fuzzing on the Nexus 9
- The aarch64 emulator (QEMU/ranchu)
- Fuzzing ranchu;
- Customize Dronity, develop your own IOCTL module;
- (time permitting) Debugging the kernel on aarch64 emulator: attach gdb, use unstripped symbols kernel, breakpoints and kernel panic analysis

THEORY

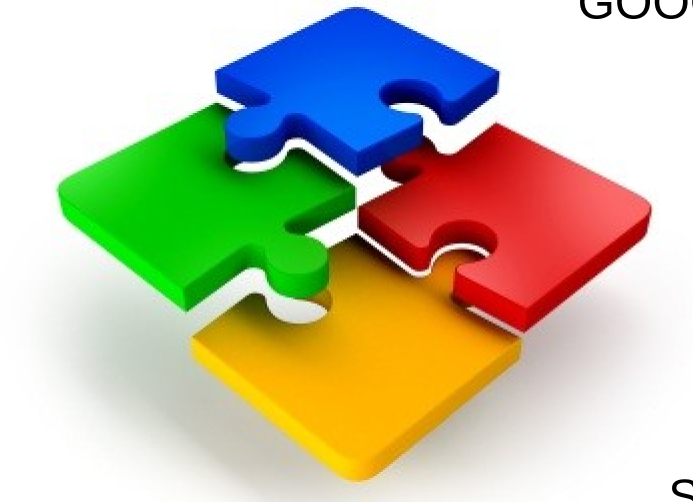
HANDS-ON

Few more bits

- This workshop is not:
  - Kernel/bin reverse engineering
  - Exploitation!!!
- I am not:
  - A serious kernel developer
  - Full time vulnerability researcher
  - Dan Rosenberg, Geohot.. etc :)

Puzzle Code

- A puzzle of different source, providers, vendors, where proper code review and control is hard!



LINUX

GOOGLE

ARM

MALI/ARM

SAMSUNG/TI/QUALCOMM

BROADCOM

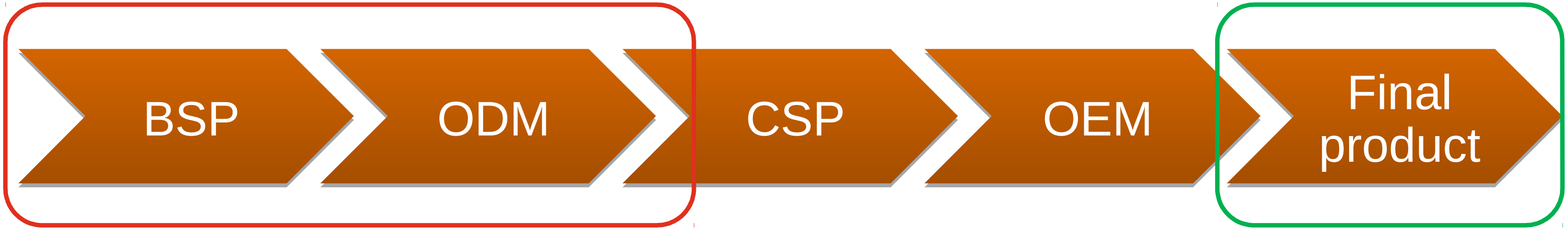
Security in the embedded devices supply chain

Security must be guaranteed and built in from the bottom of the embedded device **supply chain**, to create and deliver good, safe and high quality devices.

Two important factors:

•the inner security and the features built into the product;

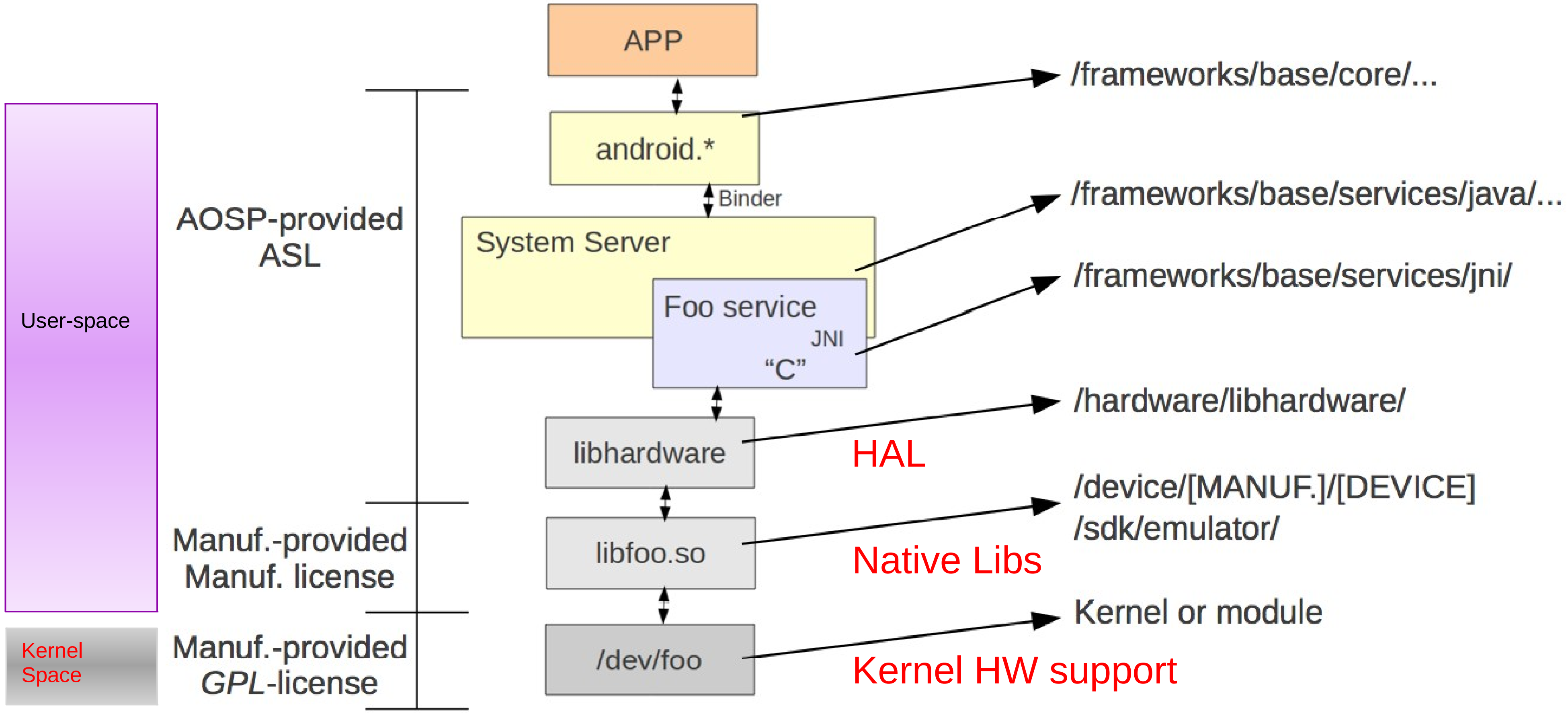
•the way developers use them.



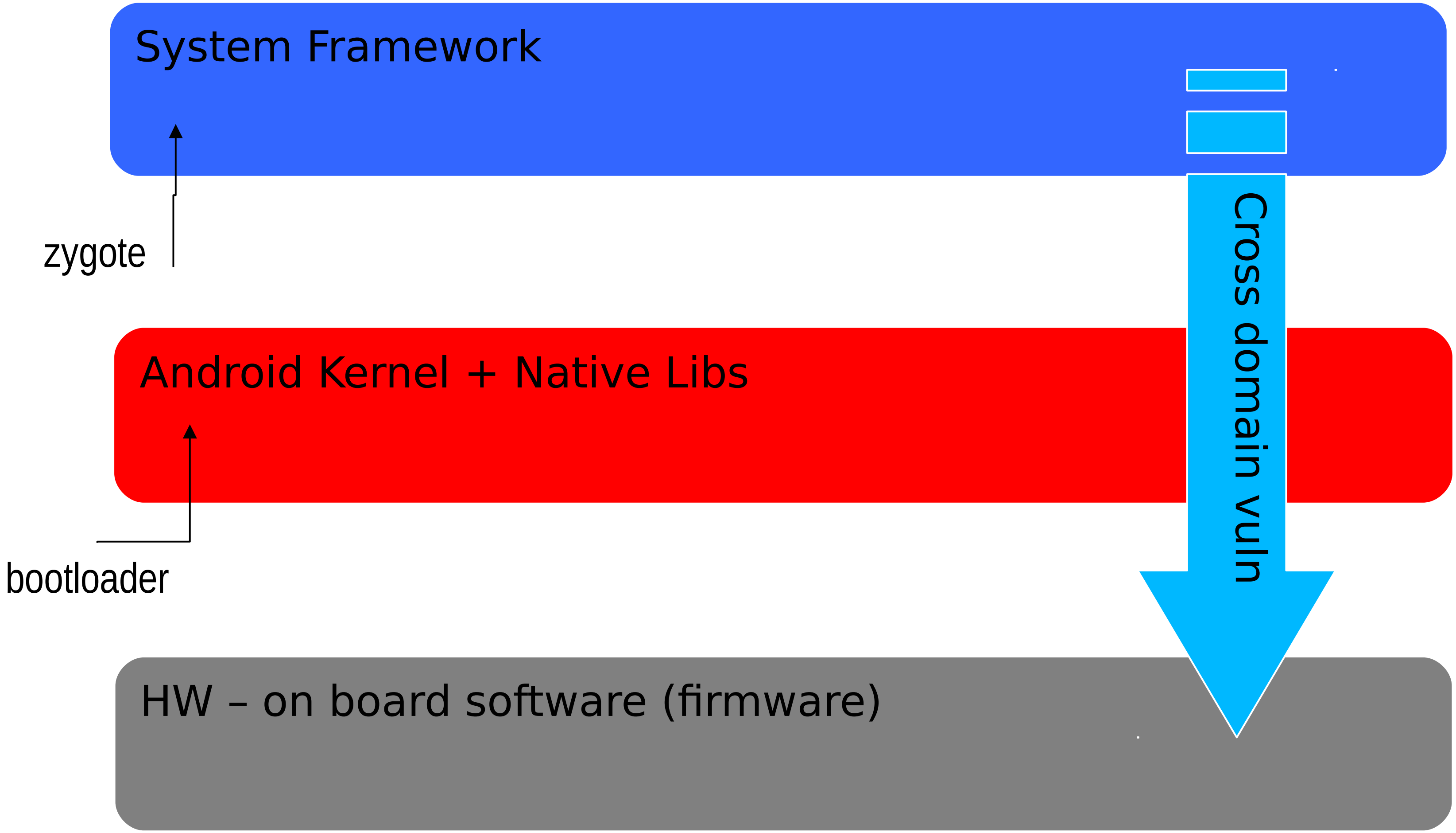
Impact of ROOTed Android

- Customize the ROM (install/remove)
- Break DRM, use unhallowed media sources
- Connect external devices
- Get malicious software, botnets and others
- Dump proprietary code and reverse
- Crack and use software decoding tools for smartcards and encrypted channels

Where are we?


APP
android.*
Binder
System Server
Foo service
JNI
"C"
libhardware
libfoo.so
/dev/foo
User-space
Kernel Space
AOSP-provided ASL
Manuf.-provided Manuf. license
Manuf.-provided GPL-license
/frameworks/base/core/...
/frameworks/base/services/java/...
/frameworks/base/services/jni/
/hardware/libhardware/
HAL
/device/[MANUF.]/[DEVICE]
/sdk/emulator/
Native Libs
Kernel or module
Kernel HW support

Embedded Device Security Domains


System Framework
zygote
Android Kernel + Native Libs
bootloader
HW – on board software (firmware)
Cross domain vuln

Rooting trend on Android

- Userspace processes exploitation
- Wrong hardening and service configuration or init.script and symlinks
- Userspace libs (e.g. libstagefright)
- Recovery and update scripts
- Dalvik privilege elevation, up to system (master key family etc)

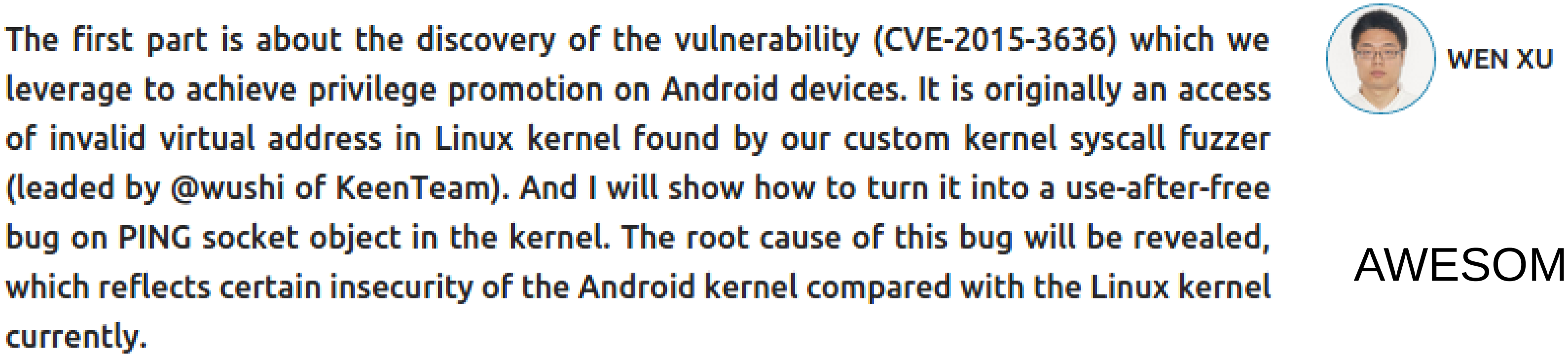
- Daemon rooting, setuid, memory corruption (gingerbreak, zergrush) and other old techniques
- **Kernel root exploitation**: framaroot (dev/exynosmem), towelroot and finally the all new pingpong root CVE-2015-3636

Kernel improvements, SELinux, Samsung root mitigation, write protection (PXN), SDLC and may be one day KASLR..... **is the same attention reserved for embedded devices?**

.. and today?

Latest bugs:
ZipBug9950697
Zip Bug 8219321 / Master keys
Zip Bug 9695860
Jar Bug 13678484 / Android FakeID
CVE 2013-6282 / put/get_user
CVE_2011_1149 / PSNueter / Ashmem Exploit
CVE_2014_3153 / Futex bug / Towelroot
CVE 2014-3847 / WeakSauce
StumpRoot
Stagefright bugs
x509 Serialization bug
PingPong root – CVE-2015-3636

rooting tool with several exploits: https://github.com/android-rooting-tools/android_run_root_shell

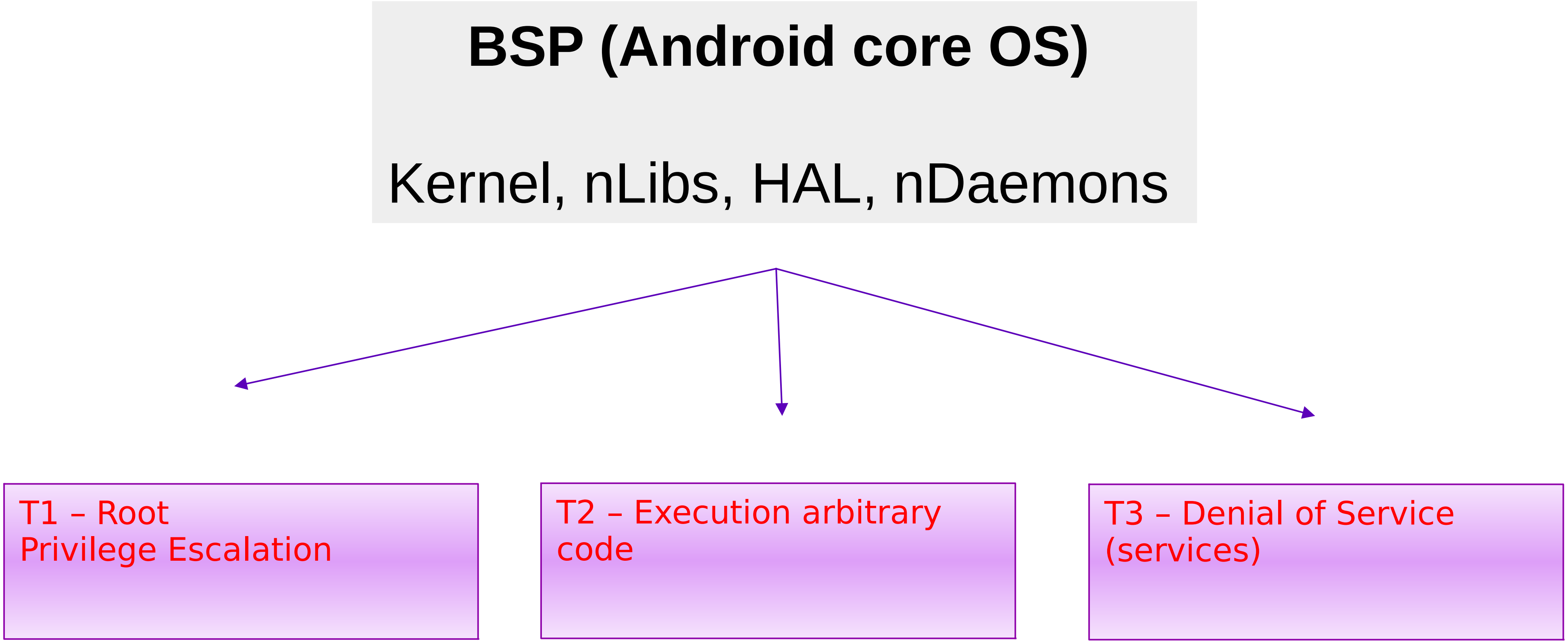


The first part is about the discovery of the vulnerability (CVE-2015-3636) which we leverage to achieve privilege promotion on Android devices. It is originally an access of invalid virtual address in Linux kernel found by our custom kernel syscall fuzzer (leaded by @wushi of KeenTeam). And I will show how to turn it into a use-after-free bug on PING socket object in the kernel. The root cause of this bug will be revealed, which reflects certain insecurity of the Android kernel compared with the Linux kernel currently.

WEN XU

AWESOME!

Threat Model



BSP (Android core OS)

Kernel, nLibs, HAL, nDaemons

T1 – Root Privilege Escalation

T2 – Execution arbitrary code

T3 – Denial of Service (services)

is it really bigger?

- Integer overflows are mitigated!
- Heap spray becomes a serious issue!
- New addressing @64bit makes more tricky writing realible exploits
- No Thum anymore?! Less ROP fun...
- New instruction and compiler optimizations
- At the moment, less HW availability!

Attack surface for rooting

Nothing changed… what are our fuzzing targets?

- **Syscalls**.
- **Drivers, IOCTLs**
- All the **communications between Userspace and Kernel Space**: i.e. Motochopper is a mmap problem on the /graphics/fb0

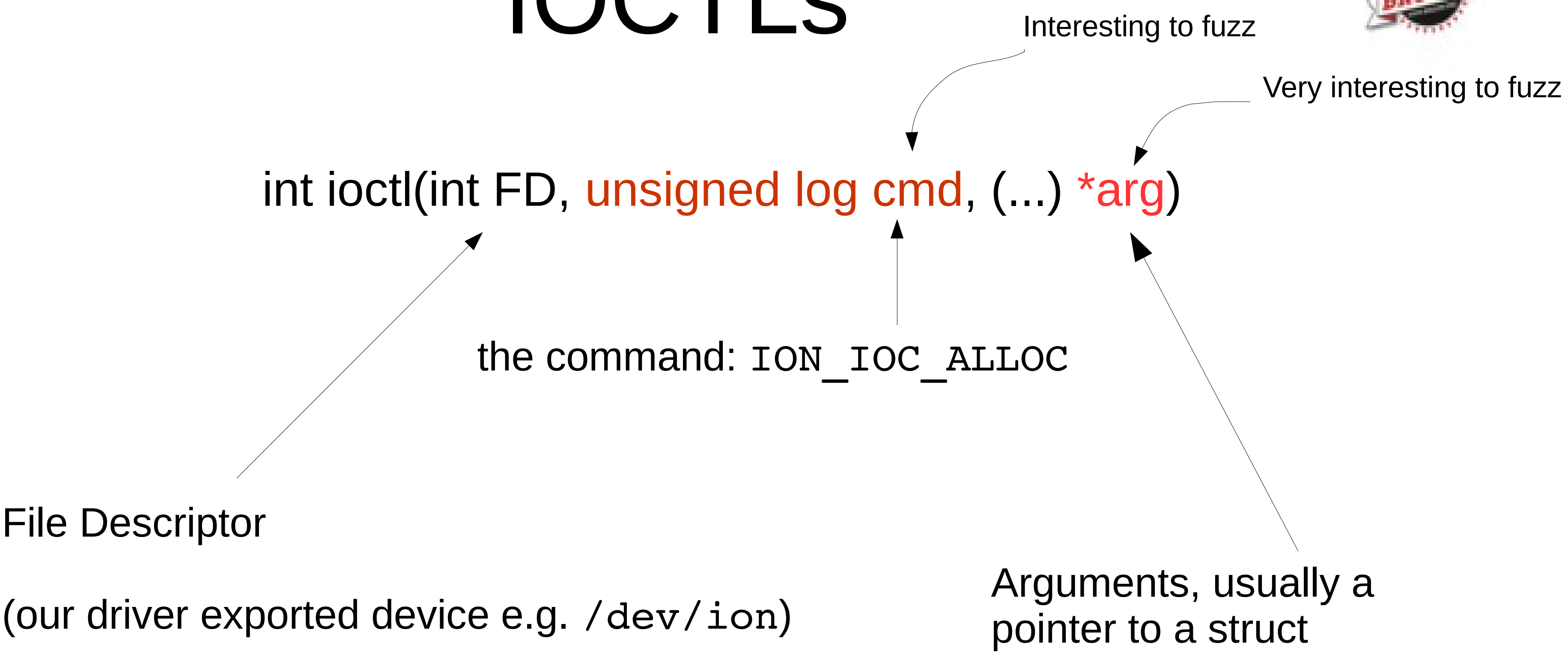
Surface fuzzing

Vulns are deep in the code!

Source Code analysis

access_ok — Checks if a user space pointer is valid
get_user — Get a simple variable from user space.
put_user — Write a simple value into user space.
__get_user — Get a simple variable from user space, with less checking.
__put_user — Write a simple value into user space, with less checking.
__copy_to_user — Copy a block of data into user space, with less checking.
__copy_from_user — Copy a block of data from user space, with less checking.
strlen_user — Get the size of a string in user space.
__strncpy_from_user — Copy a NUL terminated string from userspace, with less checking.
strncpy_from_user — Copy a NUL terminated string from userspace.
clear_user — Zero a block of memory in user space.
__clear_user — Zero a block of memory in user space, with less checking.
strnlen_user — Get the size of a string in user space.
copy_to_user — Copy a block of data into user space.
copy_from_user — Copy a block of data from user space.

IOCTLs



Interesting to fuzz

Very interesting to fuzz

int ioctl(int FD, unsigned log cmd, (...) *arg)

the command: `ION_IOC_ALLOC`

File Descriptor

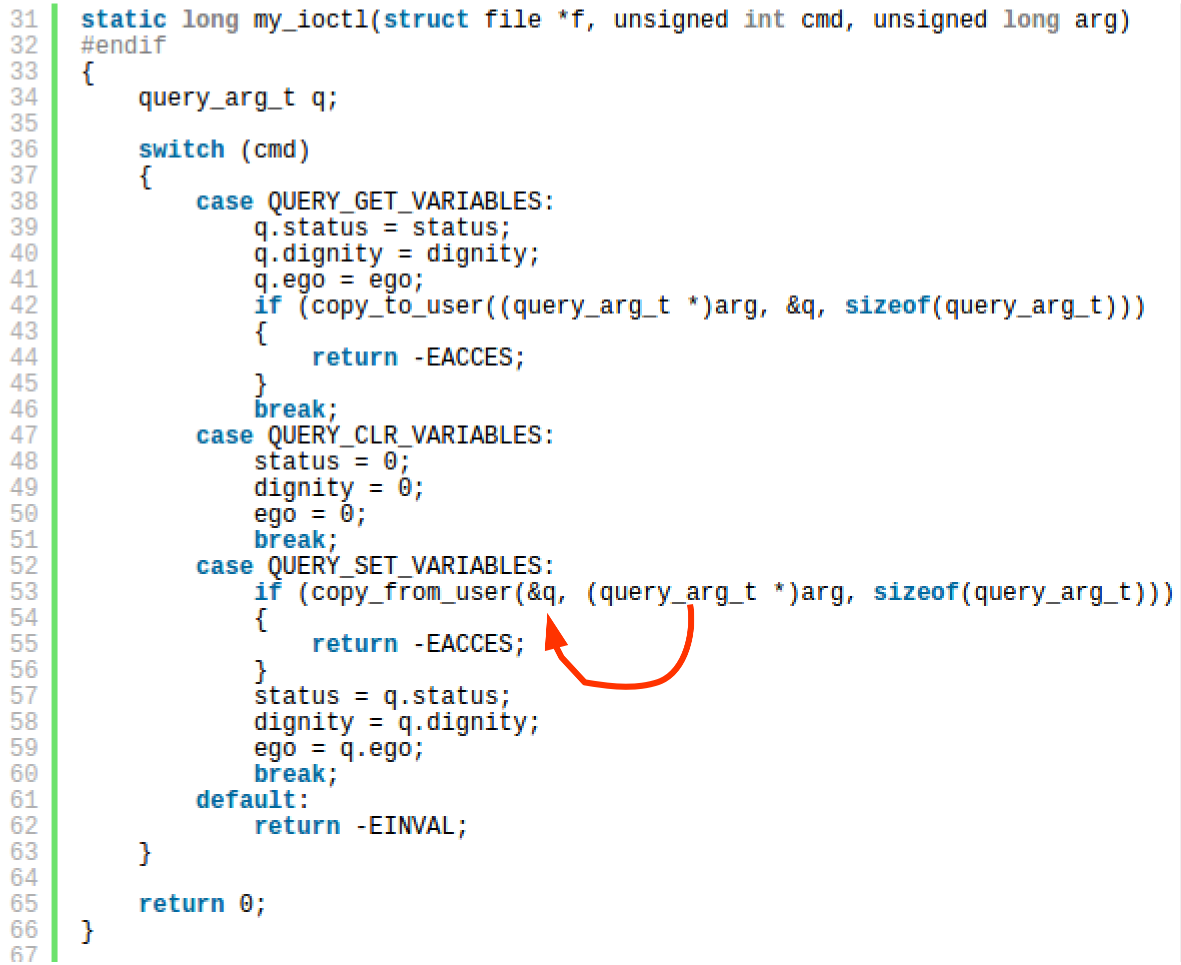
(our driver exported device e.g. `/dev/ion`)

Arguments, usually a pointer to a struct

```
struct ion_allocation_data {
        size_t len;
        size_t align;
        unsigned int flags;
        struct ion_handle *handle;
}
```

ION is a buffer sharing drivers, similar to DMABUF but with an userspace device. It replaces PMEM in Android for memory pools provisioning

Propagation of the taint

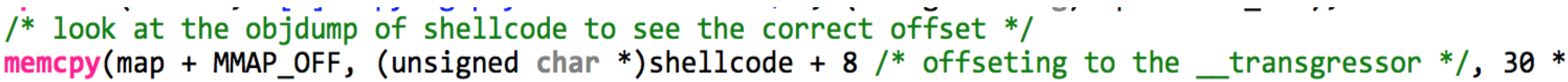


```
static long my_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
#endif
{
    query_arg_t q;

    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;
    }

    return 0;
}
```

- cmd is switched
- copy_from_user of *arg
- Inner code to call copy_from_user again for inner pointers of *arg

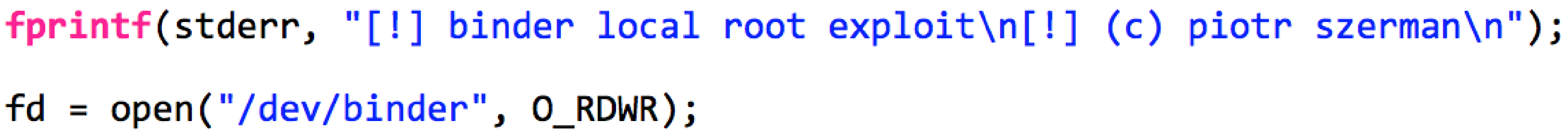
Vulnerability Explained

- Userspace memory mapped with shellcode and NOPs

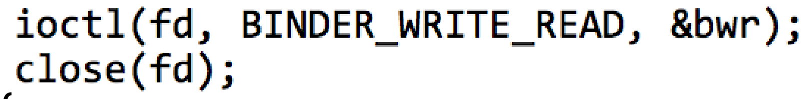


```
/* look at the objdump of shellcode to see the correct offset */
memcpy(map + MMAP_OFF, (unsigned char *)shellcode + 8 /* offseting to the __transgressor */, 30 *
```

- Over again, IOCTL call to binder

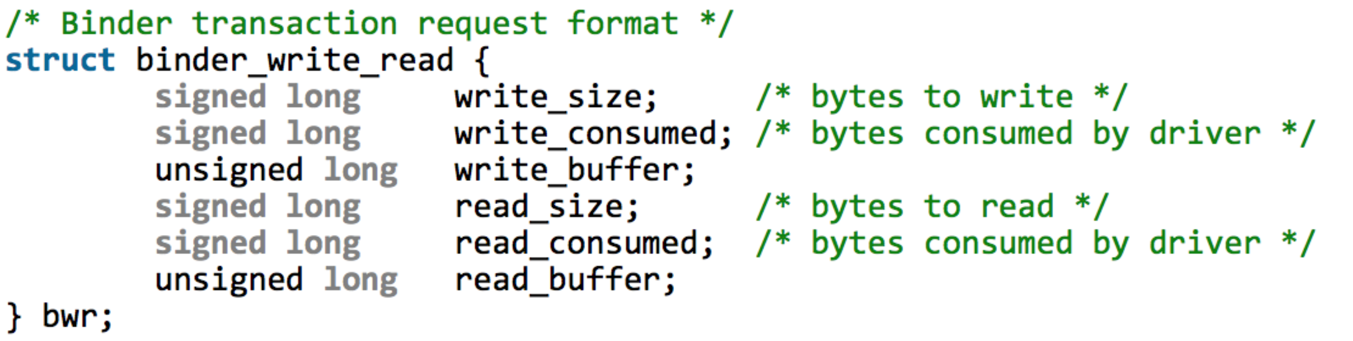


```
fprintf(stderr, "[!] binder local root exploit\n[!] (c) piotr szerman\n");

fd = open("/dev/binder", O_RDWR);
```

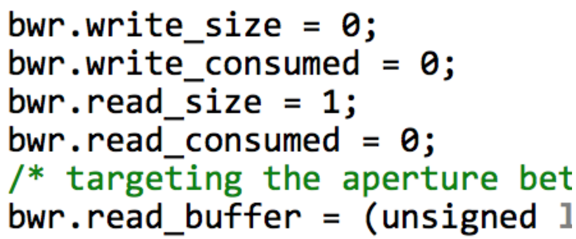


```
ioctl(fd, BINDER_WRITE_READ, &bwr);
close(fd);
```

- But &bwr is properly defined



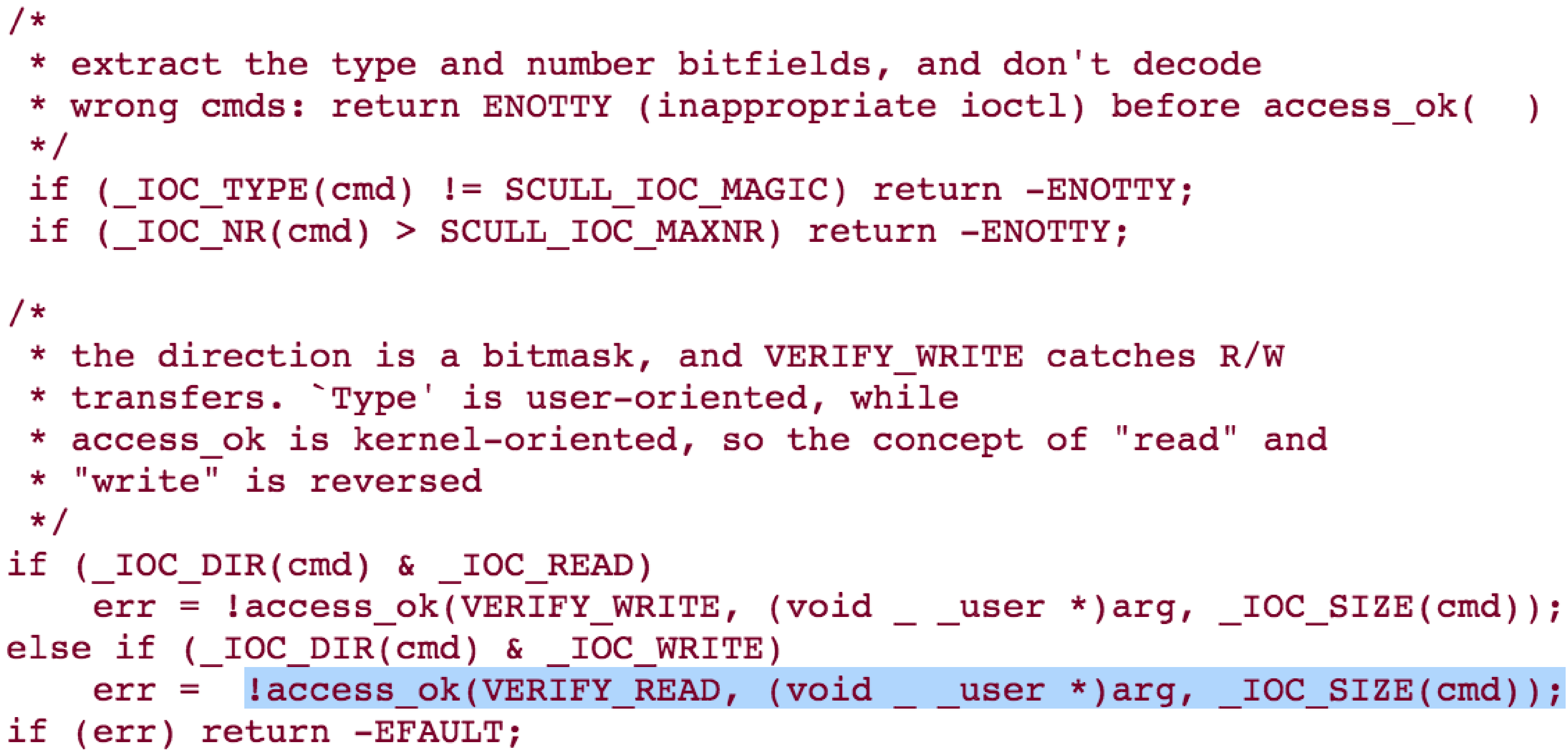
```
/* Binder transaction request format */
struct binder_write_read {
        signed long     write_size;     /* bytes to write */
        signed long     write_consumed; /* bytes consumed by driver */
        unsigned long   write_buffer;
        signed long     read_size;      /* bytes to read */
        signed long     read_consumed;  /* bytes consumed by driver */
        unsigned long   read_buffer;
} bwr;
```



```
bwr.write_size = 0;
bwr.write_consumed = 0;
bwr.read_size = 1;
bwr.read_consumed = 0;
/* targeting the aperture bet
bwr.read_buffer = (unsigned l
```

Review of access_ok (updated)

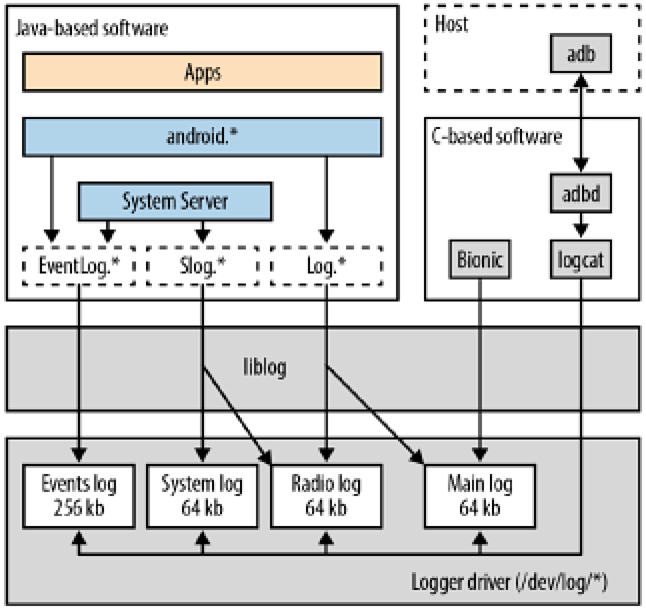
- size verification, access to the memory, more checks can be done before entering sensitive areas like copy_from_user or copy_to_user
- Access_ok is called by copy_from_user in new kernels and only __copy_from_user is direct without using it
- Be carefull to which function you are using and if your access_ok is vulnerable! Access_ok has an important vuln in old kernels.



```
/*
 * extract the type and number bitfields, and don't decode
 * wrong cmds: return ENOTTY (inappropriate ioctl) before access_ok(   )
 */
 if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC) return -ENOTTY;
 if (_IOC_NR(cmd) > SCULL_IOC_MAXNR) return -ENOTTY;

/*
 * the direction is a bitmask, and VERIFY_WRITE catches R/W
 * transfers. `Type' is user-oriented, while
 * access_ok is kernel-oriented, so the concept of "read" and
 * "write" is reversed
 */
if (_IOC_DIR(cmd) & _IOC_READ)
    err = !access_ok(VERIFY_WRITE, (void _ _user *)arg, _IOC_SIZE(cmd));
else if (_IOC_DIR(cmd) & _IOC_WRITE)
    err =  !access_ok(VERIFY_READ, (void _ _user *)arg, _IOC_SIZE(cmd));
if (err) return -EFAULT;
```

Androidization of the Kernel

- Wakelocks
- Low memory killer
- Binder
- Anonymous shared memory (ashmem)
- Alarm (extending Real Time Clock)
- Logger
- Ram Console



```
CONFIG_ASHMEM=y
CONFIG_STAGING=y
CONFIG_ANDROID=y
CONFIG_ANDROID_BINDER_IPC=y
CONFIG_ANDROID_LOGGER=y
CONFIG_ANDROID_RAM_CONSOLE=y
CONFIG_ANDROID_LOW_MEMORY_KILLER=y
```

Research strategy

Dumb fuzzing (dronity)

- General random syscall, random parameters, non targeted
- Targeted syscall, random parameters
- General IOCTLs against all possible devices (and netlink)
- General IOCTLs against a specific driver
- Specific IOCTLs commands against the specific driver - most effective code testing approach

unprivileged user (shell user via simple adb access)

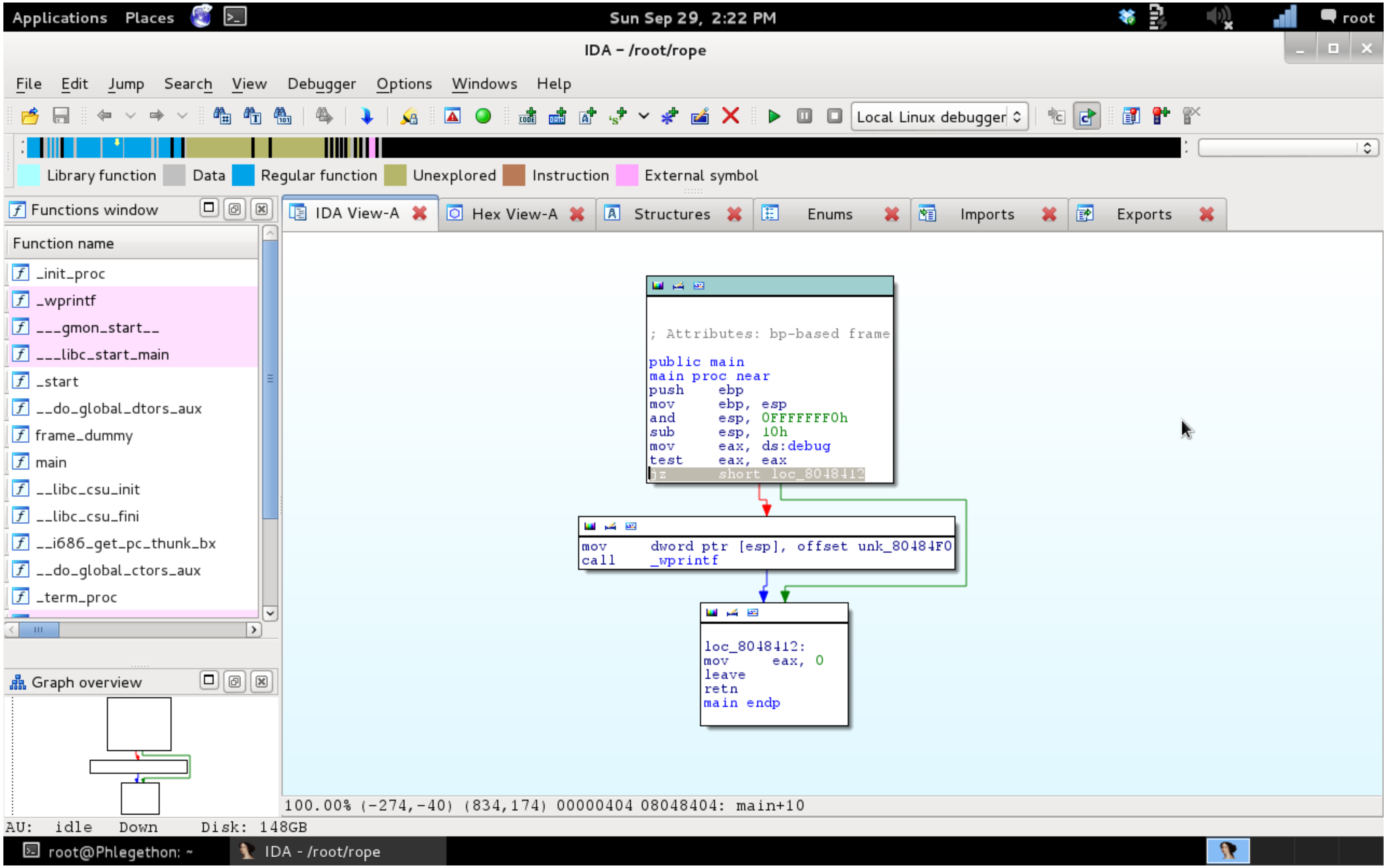
privileged user (only in privileged mode when the driver was not world read/writable)

Fuzzing 2.0 (working on..)

- In line kernel fuzzing with color-taint approach - using the **Zion**
- Userspace ioctls calls hooking (in LD-PRELOAD) + recursive lookup **Android Userspace fuzzer**.

Reverse Engineering

Reverse engineering



Function name
- _init_proc
- _wprintf
- ___gmon_start__
- ___libc_start_main
- _start
- __do_global_dtors_aux
- frame_dummy
- main
- __libc_csu_init
- __libc_csu_fini
- __i686_get_pc_thunk_bx
- __do_global_ctors_aux
- _term_proc

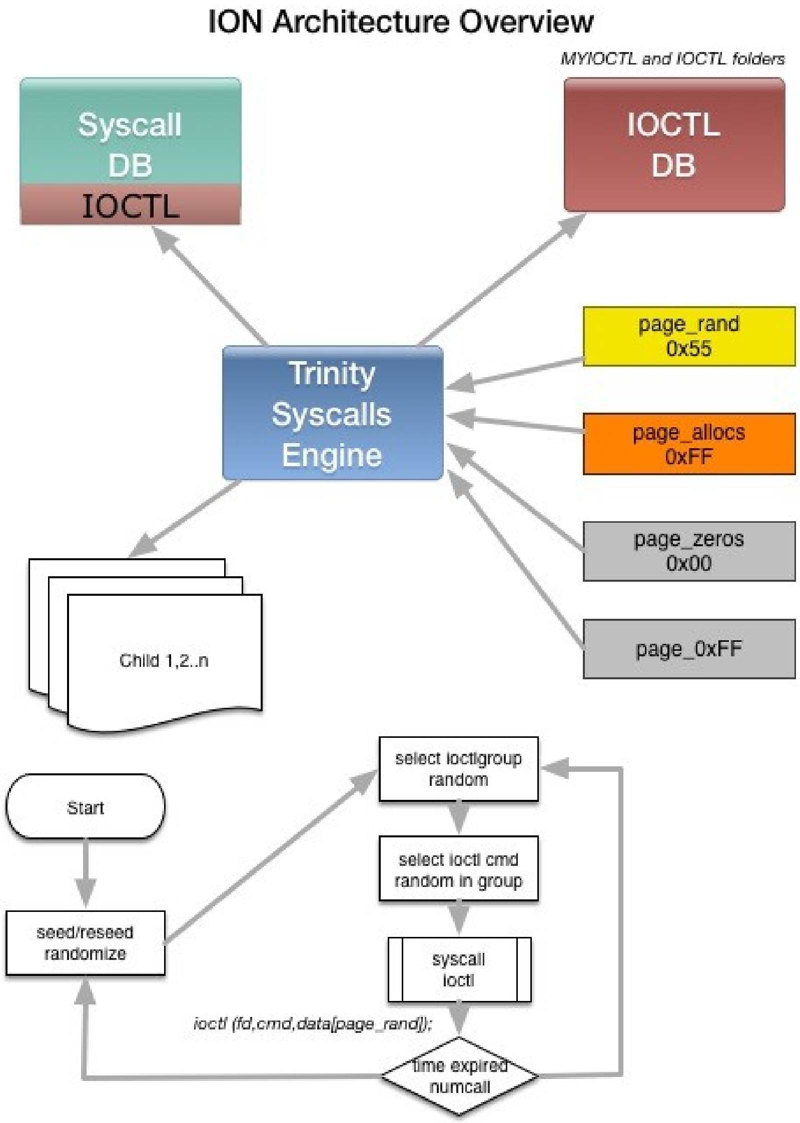
```
; Attributes: bp-based frame

public main
main proc near
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 10h
mov     eax, ds:debug
test    eax, eax
jz      short loc_8048412
```

```
mov     dword ptr [esp], offset unk_80484F0
call    _wprintf
```

```
loc_8048412:
mov     eax, 0
leave
retn
main endp
```

```
100.00% (-274,-40) (834,174) 00000404 08048404: main+10
AU:  idle   Down    Disk: 148GB
```

Trinity arch high level view



Trinity is a Linux Kernel fuzzer developed by Dave Jones who did a splendind job compared to older projects like “iknowyou”, etc.

Based on the idea of fuzzing all the syscalls, was used in the past to discover several vulnerabilities.

Attention on it dropped since a couple of years in which the amount of vuln found dropped down.

Other features of trinity

- Multi thread (forks on jobs)
- Enumeration of all possible system fd (not only file descriptors, so it gets almost everything)

- Extended logging now
- Extended support for kernel perfmon (excluded from dronity)
- Extended support for call_dumps

Trinity internals

- Definition of ARCH_IS_BIARCH

  this parameter will enable all the 32 bit syscalls on 64 bit kernels.

  32-on-32 will just use syscall() directly from do_syscall() because do32bit flag is biarch only.

- Generic syscall is performed like this:

```
if (rec->do32bit == FALSE)
        ret = syscall(call, rec->a1, rec->a2, rec->a3, rec->a4, rec->a5, rec->a6);
else
        ret = syscall32(call, rec->a1, rec->a2, rec->a3, rec->a4, rec->a5, rec->a6);
```

- Generic initialization
- Syscalls params are generic before created as follow:

```
rec->a1 = (unsigned long) rand64();
rec->a2 = (unsigned long) rand64();
rec->a3 = (unsigned long) rand64();
rec->a4 = (unsigned long) rand64();
rec->a5 = (unsigned long) rand64();
rec->a6 = (unsigned long) rand64();
```

Trinity internals

- Generic SANITIZE (where Trinity gets really awesome...)

  generic_sanitize look into the *syscall* definition and *argtypes* to decide what they are: e.g. ARG_FD, ARG_ADDRESS

  note: ARG_MMAP and ARG_ADDRESS will return the cool stuff pages generated

- **for *args* of *ioctls* nothing is defined, default is unsigned long rand64()!!!**

- Finally, customized call on the specific driver = (see specific syscall/ioctl.c) to see how

```
if (entry->sanitise)
        entry->sanitise(rec);
```

- ioctl specific sanitize
  – a) cmd is “bit” twicked (mangle_cmd)
  – b) arg is randomly assigned address (NULL excluded) – (mangle_arg)

Trinity development

http://codemonkey.org.uk/2015/07/12/future-trinity/

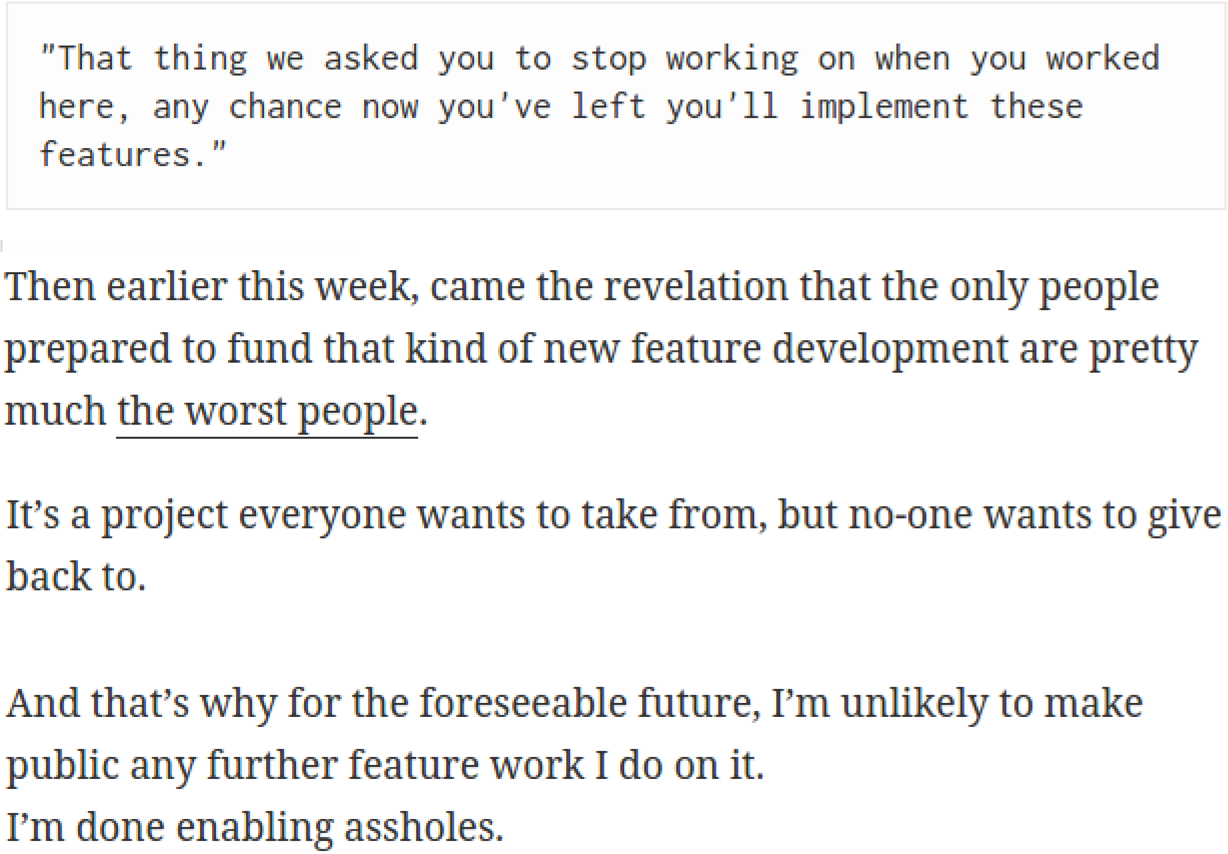
Dave Jones
kernelslacker

Boston, MA
davej@codemonkey.org.uk
http://www.codemonkey.org.uk
Joined on Nov 3, 2011

Contributions | Re

Popular repositories

trinity



```
"That thing we asked you to stop working on when you worked
here, any chance now you've left you'll implement these
features."
```

Then earlier this week, came the revelation that the only people prepared to fund that kind of new feature development are pretty much the worst people.

It's a project everyone wants to take from, but no-one wants to give back to.

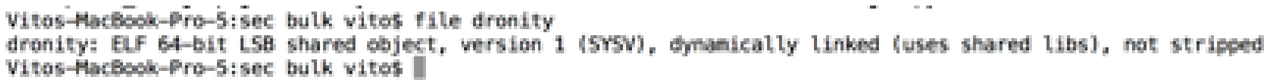
And that's why for the foreseeable future, I'm unlikely to make public any further feature work I do on it.
I'm done enabling assholes.

Introducing **Dronity**

The first, real, true porting of Trinity to Android

- patched to support Bionic native under android at level 21 (arm64 and 32)
- google NDK compatible (compiles with google ndk native toolchain)
- no glibc anymore, means no static-linked distribution
- improved makefile for android to work with ndk standalone (not ndk-build)
- compiled with native .h headers and bionic, memory structures, page (PAGE_MASK, etc) are fully compliant and compatible with android/bionic
- looked into arm64 optimization and support (need to be further improved looking to each single syscall)
- support the <constructor> macro for gcc (not C11 compatible) that prevents ioctl drivers to work with Android compilers
- support for PIE and PIC needed for Android-L to run
- gets all the benefits and feature of trinity 1.4 except perfmonitoring and backtrace (working on improving backtrace, I am up to, see changelog for more)

Dronity is pure dynamic linked aarch64 code:



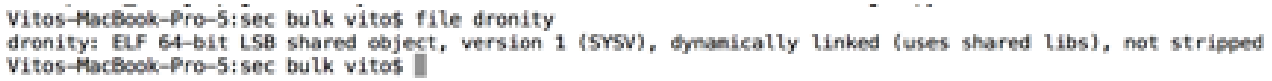
```
Vitos-MacBook-Pro-5:sec bulk vito$ file dronity
dronity: ELF 64-bit LSB shared object, version 1 (SYSV), dynamically linked (uses shared libs), not stripped
Vitos-MacBook-Pro-5:sec bulk vito$ 
```

Dronity first and second changelog

- fixed missing types (all around), adapting code to BIONIC
- disabling backtrace feature on child (unavailable on bionic)
- adjusted FD handling process adding missing FWM enum
- Fixing performance monitor, disabling it as it is unavailable on android (need to fix include in ia64/perfmonctl.c) *must do: review fd-perf.c
- Removing some small unsupported features from syscalls
- Hardcode NO support for HAVE_TERMIOS2
- Fixing network stuff and unsupported protocols (i.e. x25)
- fixing syscalls-aarch64.h for missing syscalls (1)
- Working on makefile to make it compatible with google standalone toolchain
- Extensive more changes to support Android Platform 19
  - Cope with missing functionalities of bionic
  - Removed support for fd-timerfd
  - Removed several other network protocols (incompatible and useless)
  - Patched unknown functions and removed support for cpu scheduler affinity
  - Removed swapon/off syscalls
  - ...

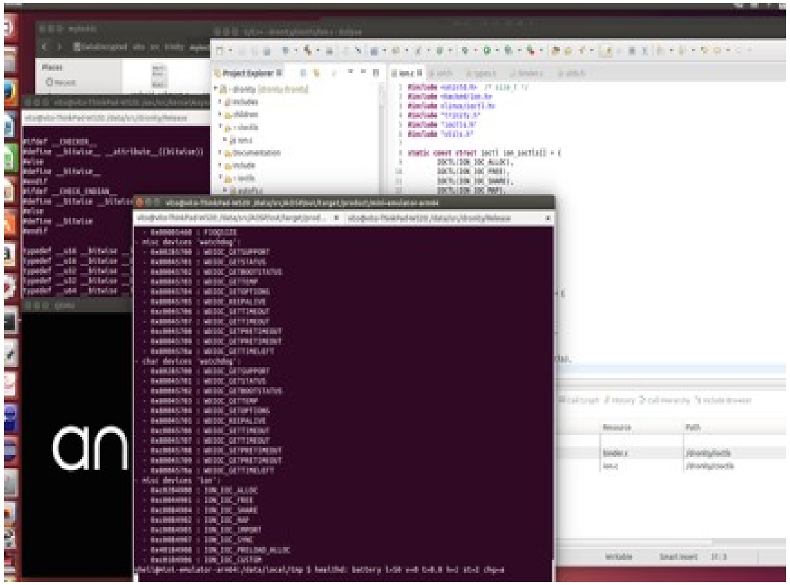
Dronity Community Edition

Dronity is a full bionic ported version of Trinity 1.4

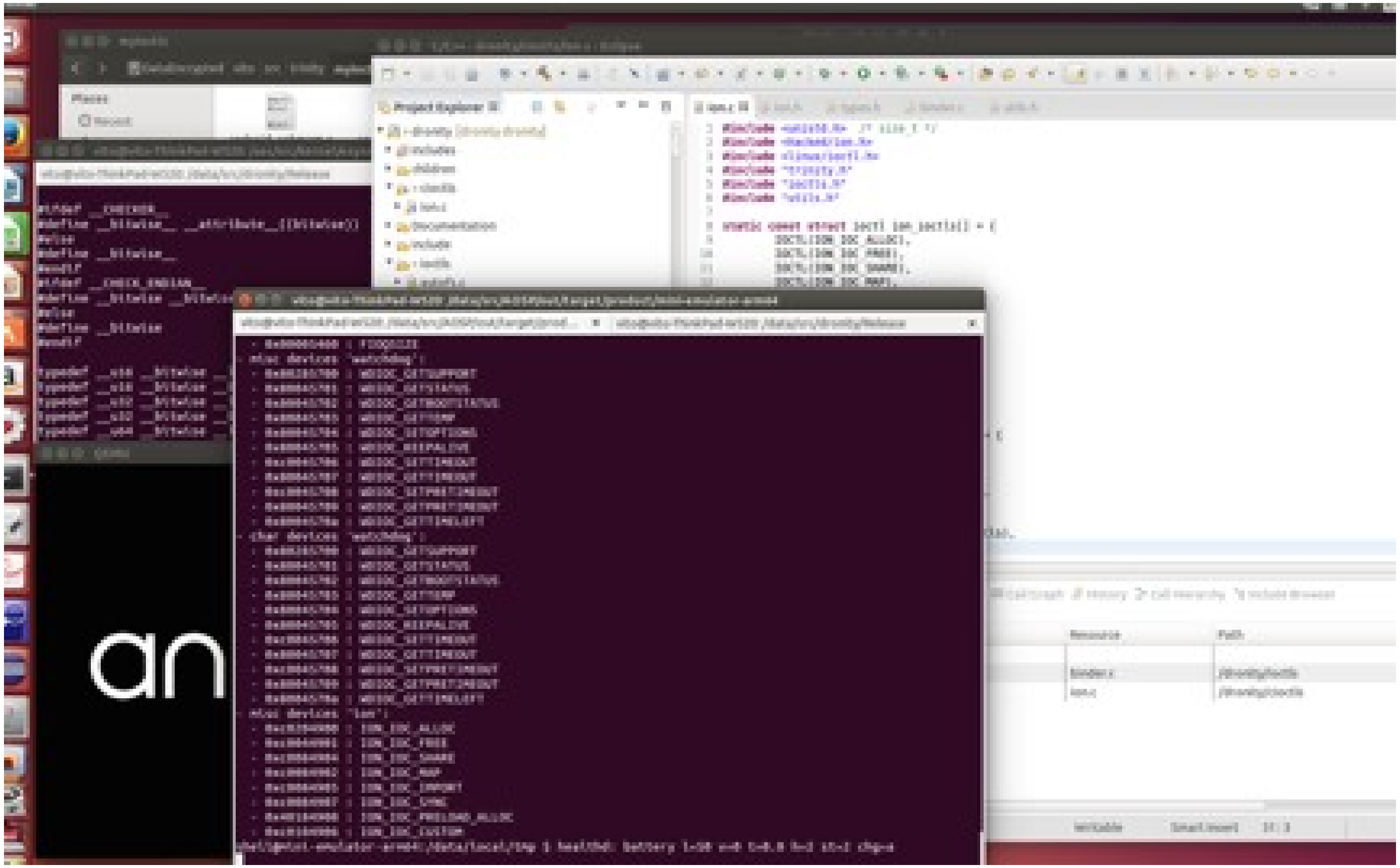


```
Vitos-MacBook-Pro-5:sec bulk vito$ file dronity
dronity: ELF 64-bit LSB shared object, version 1 (SYSV), dynamically linked (uses shared libs), not stripped
Vitos-MacBook-Pro-5:sec bulk vito$ 
```

- Dronity Community Edition, fancy MS style naming!
- New parameters
- Improved ioctl group management
- Improved fd management to point one file only
- Support for fixed pages
- Support for colored taint
- For this workshop: the version includes hacked kernel headers from Samsung ODROID original code base



Develop specific IOCTL drivers to support SE IOCTLs

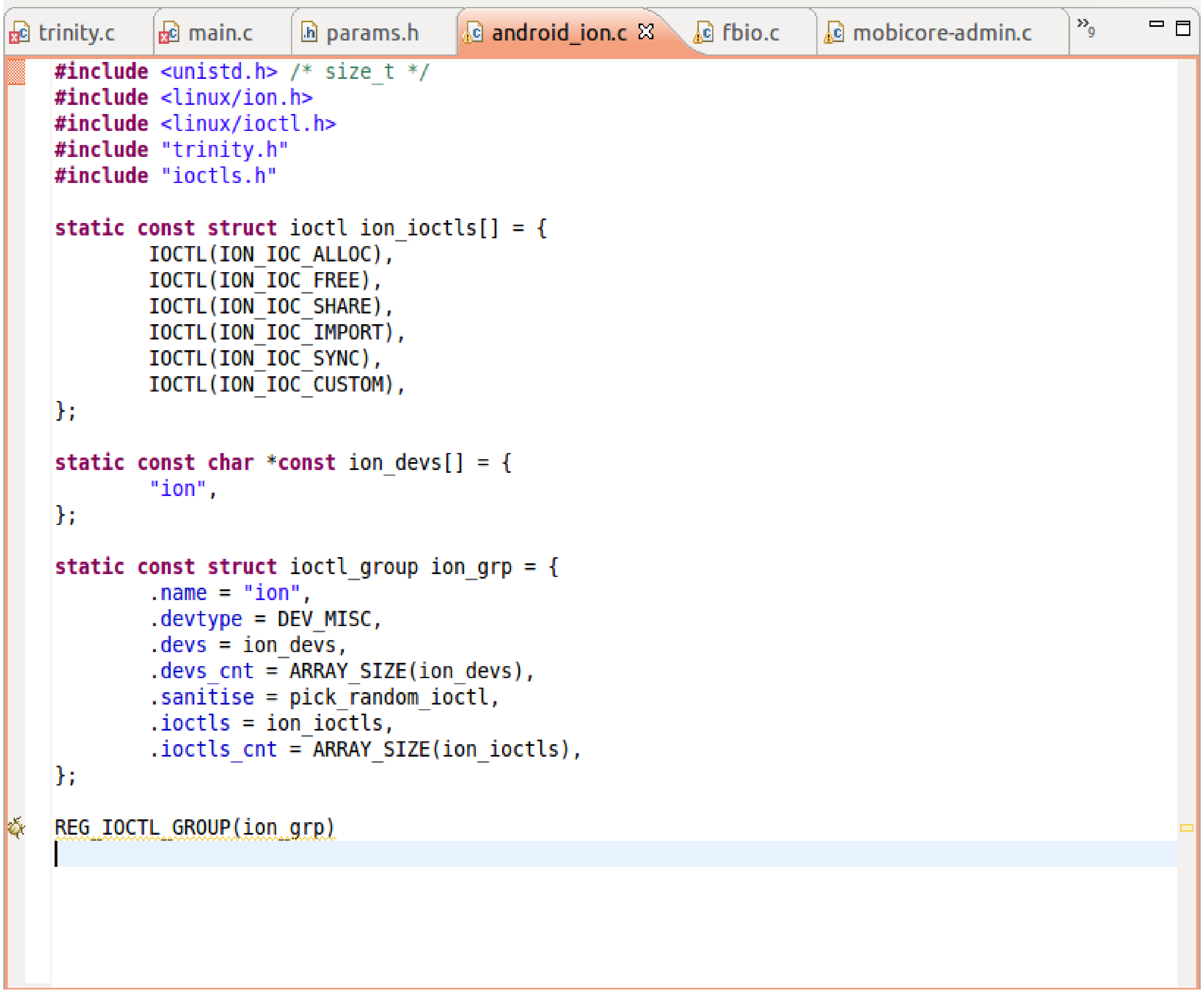


```
#include <stdint.h>  /* size_t */
#include <kernel/ion.h>
#include <linux/ioctl.h>
#include "trinity.h"
#include "ioctls.h"
#include "utils.h"

static const struct ioctl ion_ioctls[] = {
        IOCTL(ION_IOC_ALLOC),
        IOCTL(ION_IOC_FREE),
        IOCTL(ION_IOC_SHARE),
        IOCTL(ION_IOC_MAP),
```

```
#ifdef __CHECKER__
#define __bitwise__ __attribute__((bitwise))
#else
#define __bitwise__
#endif
#ifdef __CHECK_ENDIAN__
#define __bitwise __bitwise__
#else
#define __bitwise
#endif

typedef __u16 __bitwise __
typedef __u16 __bitwise __
typedef __u32 __bitwise __
typedef __u32 __bitwise __
typedef __u64 __bitwise __
```

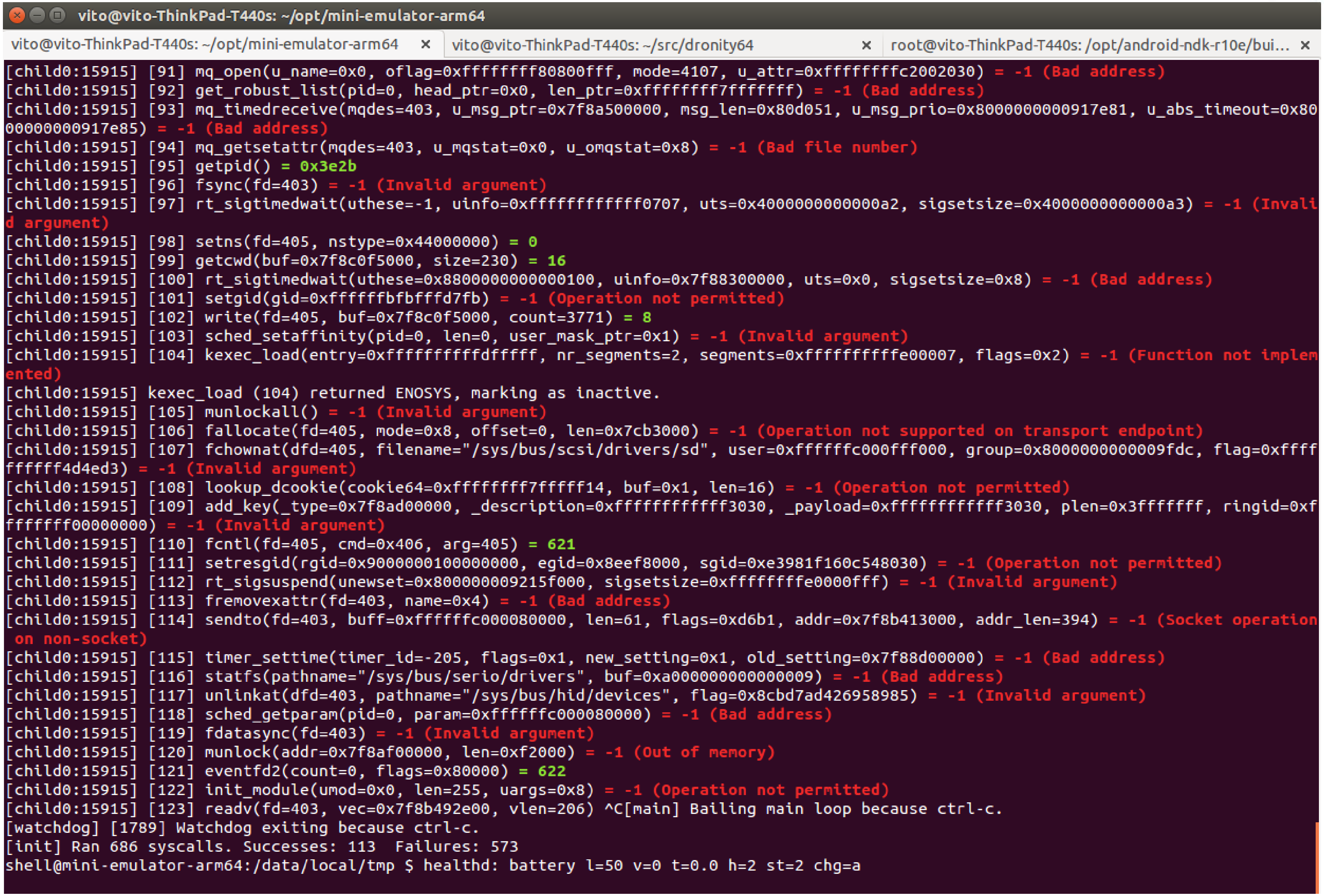
```
  - 0x00005460 : FIOQSIZE
- misc devices 'watchdog' :
  - 0x80285700 : WDIOC_GETSUPPORT
  - 0x80045701 : WDIOC_GETSTATUS
  - 0x80045702 : WDIOC_GETBOOTSTATUS
  - 0x80045703 : WDIOC_GETTEMP
  - 0x80045704 : WDIOC_SETOPTIONS
  - 0x80045705 : WDIOC_KEEPALIVE
  - 0xc0045706 : WDIOC_SETTIMEOUT
  - 0x80045707 : WDIOC_GETTIMEOUT
  - 0xc0045708 : WDIOC_SETPRETIMEOUT
  - 0x80045709 : WDIOC_GETPRETIMEOUT
  - 0x8004570a : WDIOC_GETTIMELEFT
- char devices 'watchdog' :
  - 0x80285700 : WDIOC_GETSUPPORT
  - 0x80045701 : WDIOC_GETSTATUS
  - 0x80045702 : WDIOC_GETBOOTSTATUS
  - 0x80045703 : WDIOC_GETTEMP
  - 0x80045704 : WDIOC_SETOPTIONS
  - 0x80045705 : WDIOC_KEEPALIVE
  - 0xc0045706 : WDIOC_SETTIMEOUT
  - 0x80045707 : WDIOC_GETTIMEOUT
  - 0xc0045708 : WDIOC_SETPRETIMEOUT
  - 0x80045709 : WDIOC_GETPRETIMEOUT
  - 0x8004570a : WDIOC_GETTIMELEFT
- misc devices 'ion' :
  - 0xc0204900 : ION_IOC_ALLOC
  - 0xc0044901 : ION_IOC_FREE
  - 0xc0084904 : ION_IOC_SHARE
  - 0xc0084902 : ION_IOC_MAP
  - 0xc0084905 : ION_IOC_IMPORT
  - 0xc0084907 : ION_IOC_SYNC
  - 0x40184908 : ION_IOC_PRELOAD_ALLOC
  - 0xc0104906 : ION_IOC_CUSTOM
shell@mini-emulator-arm64:/data/local/tmp $ healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
```

Trinity new ioctl support modules



```c
#include <unistd.h> /* size_t */
#include <linux/ion.h>
#include <linux/ioctl.h>
#include "trinity.h"
#include "ioctls.h"

static const struct ioctl ion_ioctls[] = {
        IOCTL(ION_IOC_ALLOC),
        IOCTL(ION_IOC_FREE),
        IOCTL(ION_IOC_SHARE),
        IOCTL(ION_IOC_IMPORT),
        IOCTL(ION_IOC_SYNC),
        IOCTL(ION_IOC_CUSTOM),
};

static const char *const ion_devs[] = {
        "ion",
};

static const struct ioctl_group ion_grp = {
        .name = "ion",
        .devtype = DEV_MISC,
        .devs = ion_devs,
        .devs_cnt = ARRAY_SIZE(ion_devs),
        .sanitise = pick_random_ioctl,
        .ioctls = ion_ioctls,
        .ioctls_cnt = ARRAY_SIZE(ion_ioctls),
};

REG_IOCTL_GROUP(ion_grp)
```

Fuzzing random syscalls


vito@vito-ThinkPad-T440s: ~/opt/mini-emulator-arm64
vito@vito-ThinkPad-T440s: ~/opt/mini-emulator-arm64
vito@vito-ThinkPad-T440s: ~/src/dronity64
root@vito-ThinkPad-T440s: /opt/android-ndk-r10e/bui...

```
[child0:15915] [91] mq_open(u_name=0x0, oflag=0xffffffff80800fff, mode=4107, u_attr=0xffffffffc2002030) = -1 (Bad address)
[child0:15915] [92] get_robust_list(pid=0, head_ptr=0x0, len_ptr=0xffffffff7fffffff) = -1 (Bad address)
[child0:15915] [93] mq_timedreceive(mqdes=403, u_msg_ptr=0x7f8a500000, msg_len=0x80d051, u_msg_prio=0x8000000000917e81, u_abs_timeout=0x80
00000000917e85) = -1 (Bad address)
[child0:15915] [94] mq_getsetattr(mqdes=403, u_mqstat=0x0, u_omqstat=0x8) = -1 (Bad file number)
[child0:15915] [95] getpid() = 0x3e2b
[child0:15915] [96] fsync(fd=403) = -1 (Invalid argument)
[child0:15915] [97] rt_sigtimedwait(uthese=-1, uinfo=0xffffffffffff0707, uts=0x40000000000000a2, sigsetsize=0x40000000000000a3) = -1 (Invali
d argument)
[child0:15915] [98] setns(fd=405, nstype=0x44000000) = 0
[child0:15915] [99] getcwd(buf=0x7f8c0f5000, size=230) = 16
[child0:15915] [100] rt_sigtimedwait(uthese=0x8800000000000100, uinfo=0x7f88300000, uts=0x0, sigsetsize=0x8) = -1 (Bad address)
[child0:15915] [101] setgid(gid=0xffffffbfbfffd7fb) = -1 (Operation not permitted)
[child0:15915] [102] write(fd=405, buf=0x7f8c0f5000, count=3771) = 8
[child0:15915] [103] sched_setaffinity(pid=0, len=0, user_mask_ptr=0x1) = -1 (Invalid argument)
[child0:15915] [104] kexec_load(entry=0xffffffffffdfffff, nr_segments=2, segments=0xfffffffffe00007, flags=0x2) = -1 (Function not implem
ented)
[child0:15915] kexec_load (104) returned ENOSYS, marking as inactive.
[child0:15915] [105] munlockall() = -1 (Invalid argument)
[child0:15915] [106] fallocate(fd=405, mode=0x8, offset=0, len=0x7cb3000) = -1 (Operation not supported on transport endpoint)
[child0:15915] [107] fchownat(dfd=405, filename="/sys/bus/scsi/drivers/sd", user=0xffffffc000fff000, group=0x8000000000009fdc, flag=0xffff
ffffff4d4ed3) = -1 (Invalid argument)
[child0:15915] [108] lookup_dcookie(cookie64=0xffffffff7fffff14, buf=0x1, len=16) = -1 (Operation not permitted)
[child0:15915] [109] add_key(_type=0x7f8ad00000, _description=0xffffffffffff3030, _payload=0xffffffffffff3030, plen=0x3fffffff, ringid=0xf
fffffff00000000) = -1 (Invalid argument)
[child0:15915] [110] fcntl(fd=405, cmd=0x406, arg=405) = 621
[child0:15915] [111] setresgid(rgid=0x9000000100000000, egid=0x8eef8000, sgid=0xe3981f160c548030) = -1 (Operation not permitted)
[child0:15915] [112] rt_sigsuspend(unewset=0x800000009215f000, sigsetsize=0xffffffffe0000fff) = -1 (Invalid argument)
[child0:15915] [113] fremovexattr(fd=403, name=0x4) = -1 (Bad address)
[child0:15915] [114] sendto(fd=403, buff=0xffffffc000080000, len=61, flags=0xd6b1, addr=0x7f8b413000, addr_len=394) = -1 (Socket operation
 on non-socket)
[child0:15915] [115] timer_settime(timer_id=-205, flags=0x1, new_setting=0x1, old_setting=0x7f88d00000) = -1 (Bad address)
[child0:15915] [116] statfs(pathname="/sys/bus/serio/drivers", buf=0xa000000000000009) = -1 (Bad address)
[child0:15915] [117] unlinkat(dfd=403, pathname="/sys/bus/hid/devices", flag=0x8cbd7ad426958985) = -1 (Invalid argument)
[child0:15915] [118] sched_getparam(pid=0, param=0xffffffc000080000) = -1 (Bad address)
[child0:15915] [119] fdatasync(fd=403) = -1 (Invalid argument)
[child0:15915] [120] munlock(addr=0x7f8af00000, len=0xf2000) = -1 (Out of memory)
[child0:15915] [121] eventfd2(count=0, flags=0x80000) = 622
[child0:15915] [122] init_module(umod=0x0, len=255, uargs=0x8) = -1 (Operation not permitted)
[child0:15915] [123] readv(fd=403, vec=0x7f8b492e00, vlen=206) ^C[main] Bailing main loop because ctrl-c.
[watchdog] [1789] Watchdog exiting because ctrl-c.
[init] Ran 686 syscalls. Successes: 113  Failures: 573
shell@mini-emulator-arm64:/data/local/tmp $ healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
```

Dronity new parameters

-f allows to target a specific device like /dev/binder. This extend -V (no need to use -V anymore) and exclude all the other non-file related file descriptors (output, buffers, network and so on..)

-i similar to the feature defined last year, it allows selecting a specific group of ioctl commands. this is now improved, selection of the group can happen by name or by device fd, works well with -f

-e exclude a specific ioctl command (not yet implemented)

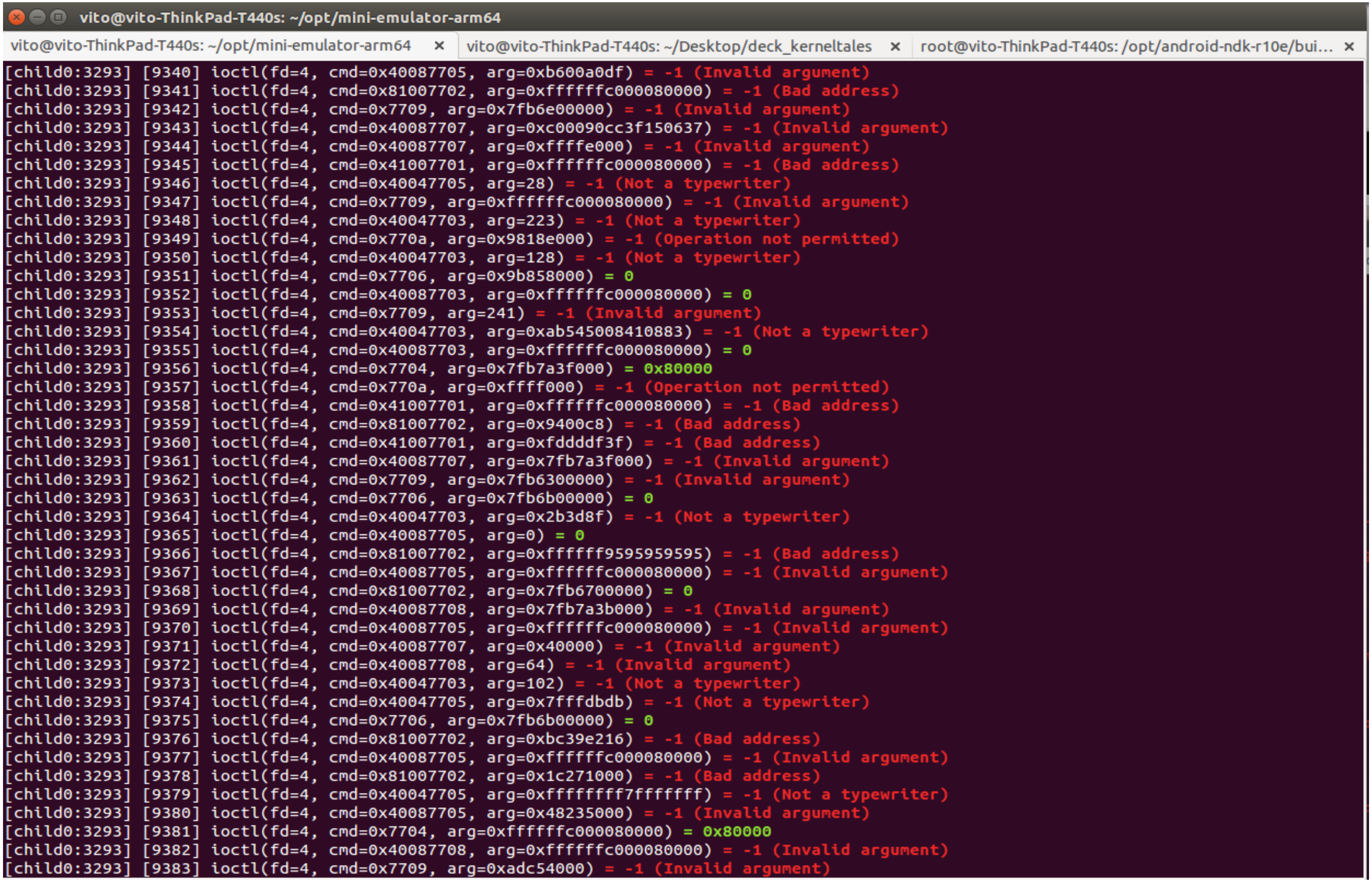
to test a specific device use:

```
dronity -f /dev/binder -i binder -c ioctl
```

-l was extended to get better touch of groups and fd descriptors!

-A specify a fixed table to send (only deadbeaf supported now for colored tainting)

Targeting IOCTLs, improved params


vito@vito-ThinkPad-T440s: ~/opt/mini-emulator-arm64
vito@vito-ThinkPad-T440s: ~/opt/mini-emulator-arm64 ×
vito@vito-ThinkPad-T440s: ~/Desktop/deck_kerneltales ×
root@vito-ThinkPad-T440s: /opt/android-ndk-r10e/bui... ×

```
[child0:3293] [9340] ioctl(fd=4, cmd=0x40087705, arg=0xb600a0df) = -1 (Invalid argument)
[child0:3293] [9341] ioctl(fd=4, cmd=0x81007702, arg=0xffffffc000080000) = -1 (Bad address)
[child0:3293] [9342] ioctl(fd=4, cmd=0x7709, arg=0x7fb6e00000) = -1 (Invalid argument)
[child0:3293] [9343] ioctl(fd=4, cmd=0x40087707, arg=0xc00090cc3f150637) = -1 (Invalid argument)
[child0:3293] [9344] ioctl(fd=4, cmd=0x40087707, arg=0xffffe000) = -1 (Invalid argument)
[child0:3293] [9345] ioctl(fd=4, cmd=0x41007701, arg=0xffffffc000080000) = -1 (Bad address)
[child0:3293] [9346] ioctl(fd=4, cmd=0x40047705, arg=28) = -1 (Not a typewriter)
[child0:3293] [9347] ioctl(fd=4, cmd=0x7709, arg=0xffffffc000080000) = -1 (Invalid argument)
[child0:3293] [9348] ioctl(fd=4, cmd=0x40047703, arg=223) = -1 (Not a typewriter)
[child0:3293] [9349] ioctl(fd=4, cmd=0x770a, arg=0x9818e000) = -1 (Operation not permitted)
[child0:3293] [9350] ioctl(fd=4, cmd=0x40047703, arg=128) = -1 (Not a typewriter)
[child0:3293] [9351] ioctl(fd=4, cmd=0x7706, arg=0x9b858000) = 0
[child0:3293] [9352] ioctl(fd=4, cmd=0x40087703, arg=0xffffffc000080000) = 0
[child0:3293] [9353] ioctl(fd=4, cmd=0x7709, arg=241) = -1 (Invalid argument)
[child0:3293] [9354] ioctl(fd=4, cmd=0x40047703, arg=0xab545008410883) = -1 (Not a typewriter)
[child0:3293] [9355] ioctl(fd=4, cmd=0x40087703, arg=0xffffffc000080000) = 0
[child0:3293] [9356] ioctl(fd=4, cmd=0x7704, arg=0x7fb7a3f000) = 0x80000
[child0:3293] [9357] ioctl(fd=4, cmd=0x770a, arg=0xffff000) = -1 (Operation not permitted)
[child0:3293] [9358] ioctl(fd=4, cmd=0x41007701, arg=0xffffffc000080000) = -1 (Bad address)
[child0:3293] [9359] ioctl(fd=4, cmd=0x81007702, arg=0x9400c8) = -1 (Bad address)
[child0:3293] [9360] ioctl(fd=4, cmd=0x41007701, arg=0xfddddf3f) = -1 (Bad address)
[child0:3293] [9361] ioctl(fd=4, cmd=0x40087707, arg=0x7fb7a3f000) = -1 (Invalid argument)
[child0:3293] [9362] ioctl(fd=4, cmd=0x7709, arg=0x7fb6300000) = -1 (Invalid argument)
[child0:3293] [9363] ioctl(fd=4, cmd=0x7706, arg=0x7fb6b00000) = 0
[child0:3293] [9364] ioctl(fd=4, cmd=0x40047703, arg=0x2b3d8f) = -1 (Not a typewriter)
[child0:3293] [9365] ioctl(fd=4, cmd=0x40087705, arg=0) = 0
[child0:3293] [9366] ioctl(fd=4, cmd=0x81007702, arg=0xffffff9595959595) = -1 (Bad address)
[child0:3293] [9367] ioctl(fd=4, cmd=0x40087705, arg=0xffffffc000080000) = -1 (Invalid argument)
[child0:3293] [9368] ioctl(fd=4, cmd=0x81007702, arg=0x7fb6700000) = 0
[child0:3293] [9369] ioctl(fd=4, cmd=0x40087708, arg=0x7fb7a3b000) = -1 (Invalid argument)
[child0:3293] [9370] ioctl(fd=4, cmd=0x40087705, arg=0xffffffc000080000) = -1 (Invalid argument)
[child0:3293] [9371] ioctl(fd=4, cmd=0x40087707, arg=0x40000) = -1 (Invalid argument)
[child0:3293] [9372] ioctl(fd=4, cmd=0x40087708, arg=64) = -1 (Invalid argument)
[child0:3293] [9373] ioctl(fd=4, cmd=0x40047703, arg=102) = -1 (Not a typewriter)
[child0:3293] [9374] ioctl(fd=4, cmd=0x40047705, arg=0x7fffdbdb) = -1 (Not a typewriter)
[child0:3293] [9375] ioctl(fd=4, cmd=0x7706, arg=0x7fb6b00000) = 0
[child0:3293] [9376] ioctl(fd=4, cmd=0x81007702, arg=0xbc39e216) = -1 (Bad address)
[child0:3293] [9377] ioctl(fd=4, cmd=0x40087705, arg=0xffffffc000080000) = -1 (Invalid argument)
[child0:3293] [9378] ioctl(fd=4, cmd=0x81007702, arg=0x1c271000) = -1 (Bad address)
[child0:3293] [9379] ioctl(fd=4, cmd=0x40047705, arg=0xffffffff7fffffff) = -1 (Not a typewriter)
[child0:3293] [9380] ioctl(fd=4, cmd=0x40087705, arg=0x48235000) = -1 (Invalid argument)
[child0:3293] [9381] ioctl(fd=4, cmd=0x7704, arg=0xffffffc000080000) = 0x80000
[child0:3293] [9382] ioctl(fd=4, cmd=0x40087708, arg=0xffffffc000080000) = -1 (Invalid argument)
[child0:3293] [9383] ioctl(fd=4, cmd=0x7709, arg=0xadc54000) = -1 (Invalid argument)
```

Is that everything you need?

Debugging the kernel

- kgdb
  - Based on gdb
  - Client-server approach
  - Need to patch the UART driver (support for polling)
- P-trace
- Kernel Printk
- last_kmsg
- Portmortem stats /data/dropbox
- Fuzzing tool log
- JTAG debugging – Segger Jlink, Trace32
  - Allows low level debugging for the bootloader

With a bit of luck, monitoring the easy way...

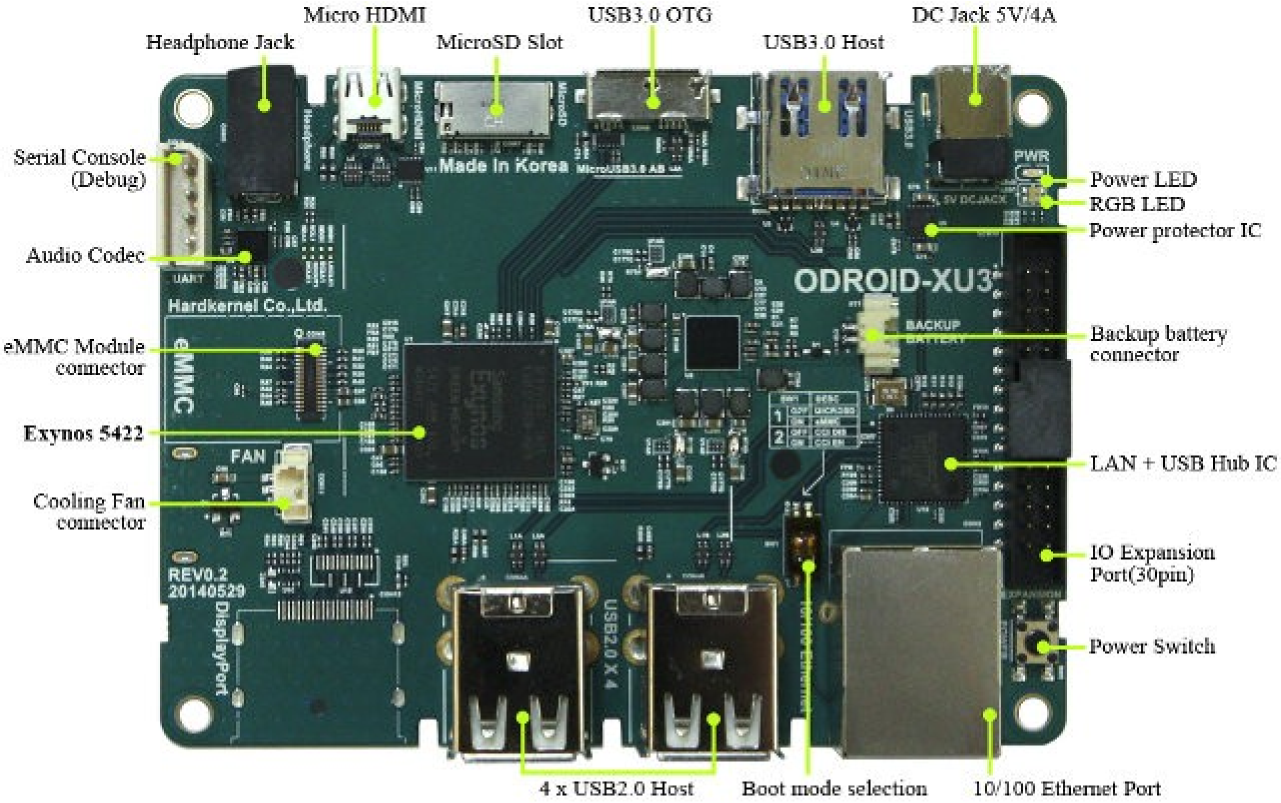
1) serial console... almost a must!
2) RAM CONSOLE, /proc/last_kmsg
3) After Android 5, kernel 3.10> /sys/fs/pstore

Setting up a devel environment

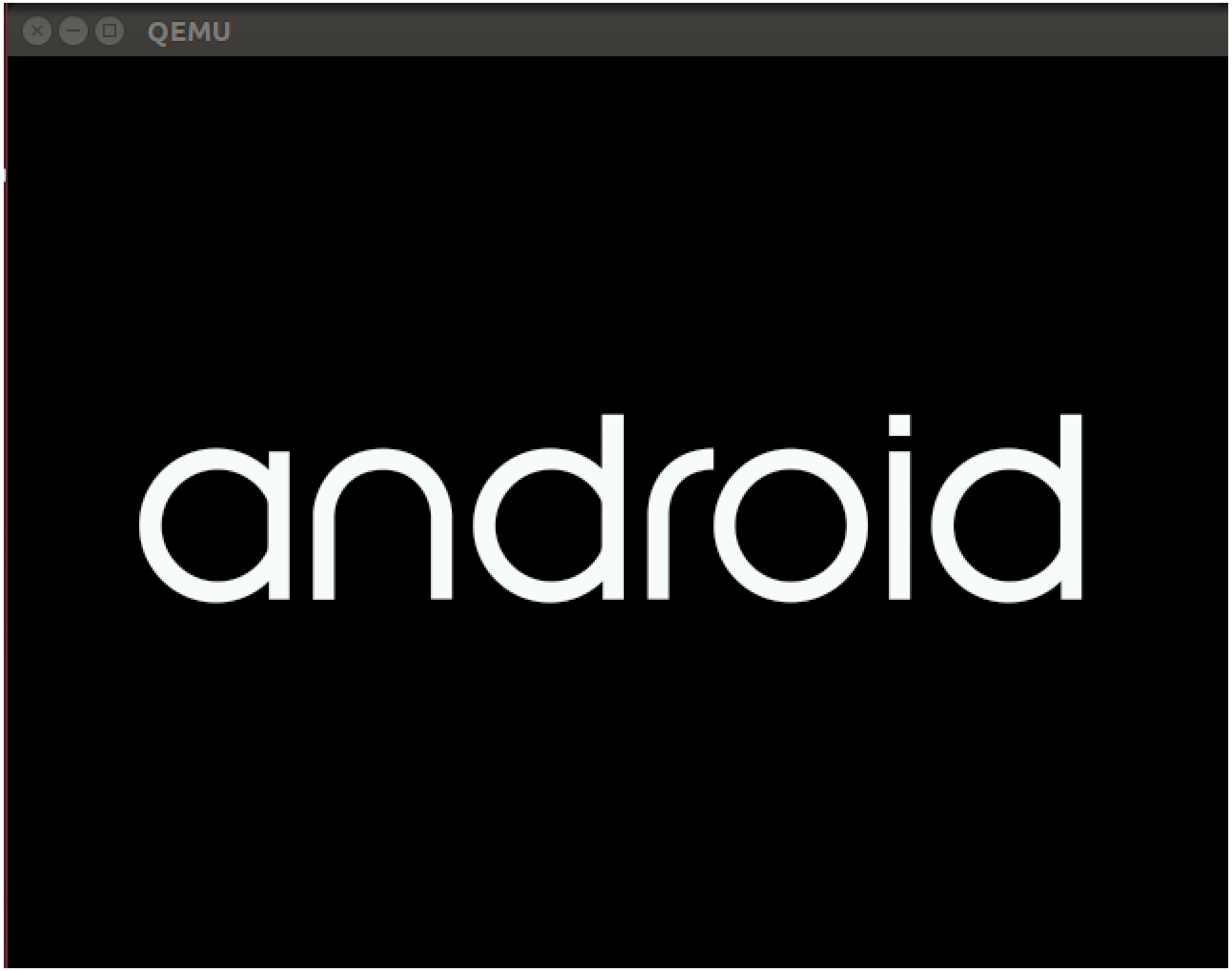
- Android SDK
- Android NDK
- Keep an eye on Linaro toolchains
- Understand about platforms and bionic
- Reading the code:
  - Eclipse
  - Understand
- Debugging: gdb.. eclipse?!

QUICK OVERVIEW DEMO

ODROID XU3


Micro HDMI
Headphone Jack
MicroSD Slot
USB3.0 OTG
USB3.0 Host
DC Jack 5V/4A
Serial Console (Debug)
Audio Codec
eMMC Module connector
Exynos 5422
Cooling Fan connector
Power LED
RGB LED
Power protector IC
Backup battery connector
LAN + USB Hub IC
IO Expansion Port(30pin)
Power Switch
4 x USB2.0 Host
Boot mode selection
10/100 Ethernet Port
ODROID-XU3

And for the aarch64?



Fuzzing demo

Compiling the kernel...

- make yourboard_defconfig ARCH=arm64
  - Check inside your arch/<wathever>/configs what is exactly your config
  - This will generate .config file
- make CROSS_COMPILE=your-kernel-toolchain ARCH=arm64 -j8
- Flash it on the board, “fuse” to use Samsung terminology! Easy with fastboot!

- Note: you need the toolchain in the path, Linaro toolchain is great to compile kernels. You can get it from apt repos or manually from the Linaro site.

DEMO

Cross Compile Dronity

1) get Android NDK from Google

2) run make-standalone-toolchain to get a standalone compiler

```
./make-standalone-toolchain.sh --toolchain=aarch64-linux-android-4.9 --platform=android-21 --install-dir=/opt/toolchain
--ndk-dir=/opt/android-ndk-r10c --system=linux-x86_64
```

3) important: set your toolchain bin in env PATH, add it to .bashrc

```
echo "PATH=$PATH:/opt/toolchain/bin" >> ~/.bashrc
source ~/.bashrc
```

4) compile dronity, you are now ready to go!

```
 make CROSS_COMPILE=aarch64-linux-android- CC=aarch64-linux-android-gcc
```

(specify here the type of compiler you want, this example is based on aarch64 compiler for android version 4.9 ndk version 10c from Google)

DEMO

Test it on the board

- adb push dronity /data/local/tmp
- Use the console or the adb shell
- ./dronity –help
- ./dronity -l (print out IOCTLs groups)

FIRST FUZZING DEMO!

What about aarch64...

Fuzzing aarch64 ranchu

- QEMU emulator64 is there but.. not there yet!
- There is a long story about “ranchu where are you”, article on my blog
  http://restart-thinking.vitorallo.com/2014/11/ranchu-where-are-you-kernel-and.html

- Need to compile emulator system images from AOSP >5

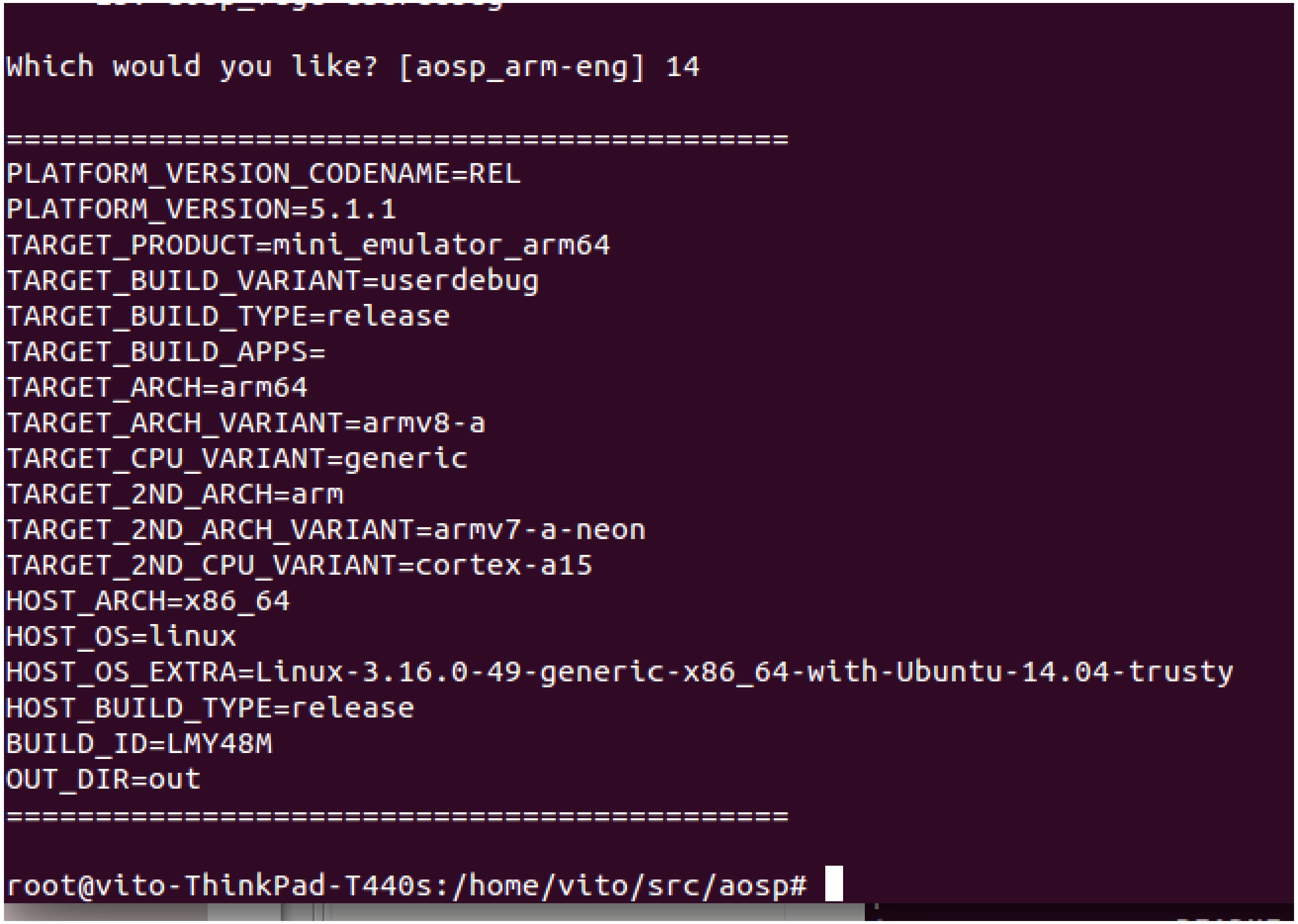
- Compiling the AOSP:

  source build/setenvironment.sh

  lunch (pick the mini_emulator64 which does not work :) :)
- My patch still works on 5.1.1

Ready to go?

- Go for a run, look for a girl...



```
Which would you like? [aosp_arm-eng] 14

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=5.1.1
TARGET_PRODUCT=mini_emulator_arm64
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=generic
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv7-a-neon
TARGET_2ND_CPU_VARIANT=cortex-a15
HOST_ARCH=x86_64
HOST_OS=linux
HOST_OS_EXTRA=Linux-3.16.0-49-generic-x86_64-with-Ubuntu-14.04-trusty
HOST_BUILD_TYPE=release
BUILD_ID=LMY48M
OUT_DIR=out
============================================

root@vito-ThinkPad-T440s:/home/vito/src/aosp#
```

Fuzzing on QEMU
&
Fuzzing on Nexus 9

You got a crash, now what?

CAUTION

This section is not reccomended for VIM power users and real kernel developers

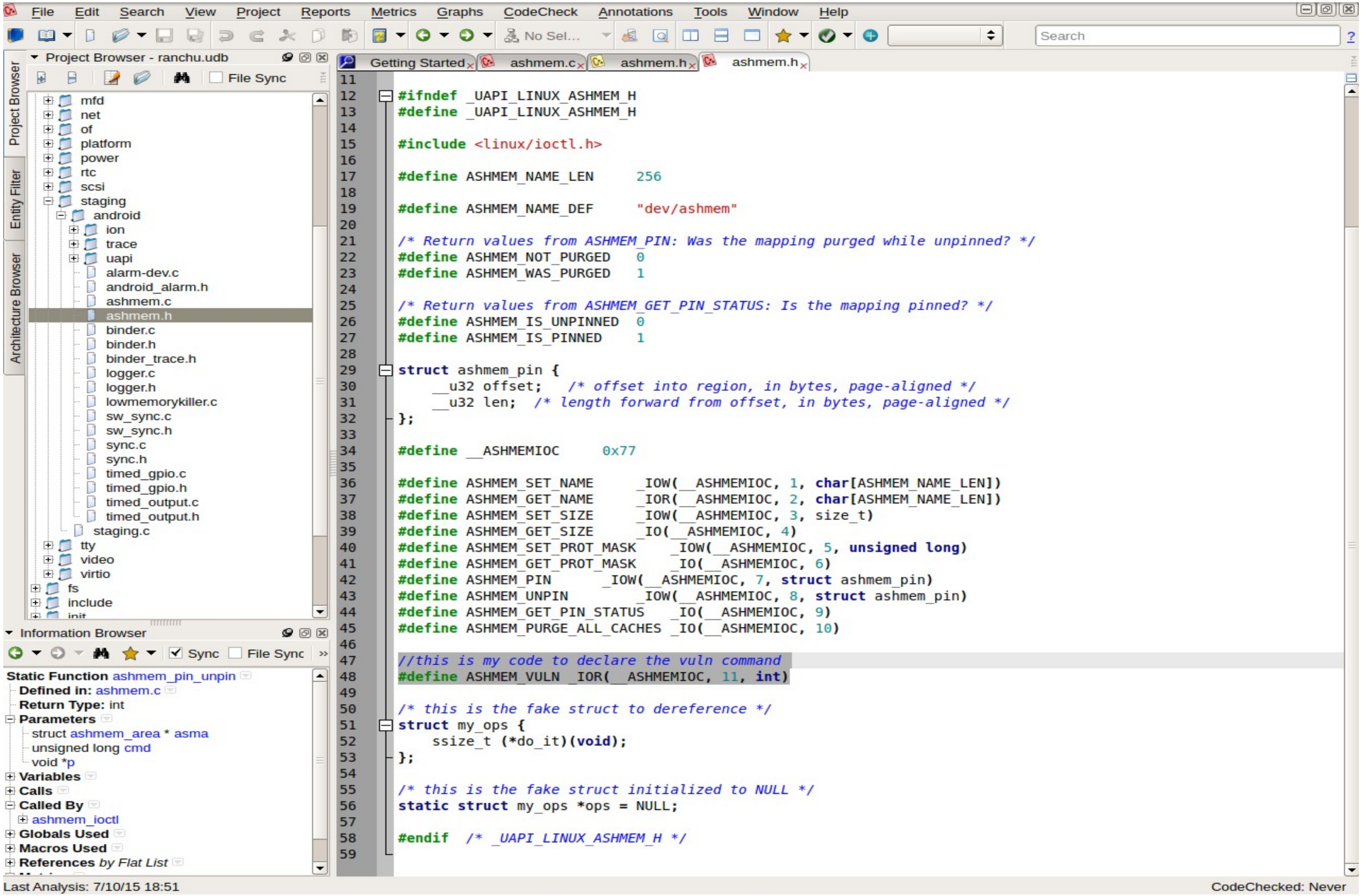
Loading your code into Eclipse to navigate it...

It almost works, helps you navigate it... not perfect but the indexer helps and works if properly configured with compiler specs!

http://restart-thinking.vitorallo.com/2014/11/tutorials-on-how-to-build-edit-and.html

DEMO ACCORDING TIME

“Understand” the code...



Static Function ashmem_pin_unpin
Defined in: ashmem.c
Return Type: int
Parameters
struct ashmem_area * asma
unsigned long cmd
void *p
Variables
Calls
Called By
ashmem_ioctl
Globals Used
Macros Used
References *by Flat List*

```
#ifndef _UAPI_LINUX_ASHMEM_H
#define _UAPI_LINUX_ASHMEM_H

#include <linux/ioctl.h>

#define ASHMEM_NAME_LEN     256

#define ASHMEM_NAME_DEF     "dev/ashmem"

/* Return values from ASHMEM_PIN: Was the mapping purged while unpinned? */
#define ASHMEM_NOT_PURGED   0
#define ASHMEM_WAS_PURGED   1

/* Return values from ASHMEM_GET_PIN_STATUS: Is the mapping pinned? */
#define ASHMEM_IS_UNPINNED  0
#define ASHMEM_IS_PINNED    1

struct ashmem_pin {
    __u32 offset;   /* offset into region, in bytes, page-aligned */
    __u32 len;  /* length forward from offset, in bytes, page-aligned */
};

#define __ASHMEMIOC     0x77

#define ASHMEM_SET_NAME     _IOW(__ASHMEMIOC, 1, char[ASHMEM_NAME_LEN])
#define ASHMEM_GET_NAME     _IOR(__ASHMEMIOC, 2, char[ASHMEM_NAME_LEN])
#define ASHMEM_SET_SIZE     _IOW(__ASHMEMIOC, 3, size_t)
#define ASHMEM_GET_SIZE     _IO(__ASHMEMIOC, 4)
#define ASHMEM_SET_PROT_MASK    _IOW(__ASHMEMIOC, 5, unsigned long)
#define ASHMEM_GET_PROT_MASK    _IO(__ASHMEMIOC, 6)
#define ASHMEM_PIN      _IOW(__ASHMEMIOC, 7, struct ashmem_pin)
#define ASHMEM_UNPIN        _IOW(__ASHMEMIOC, 8, struct ashmem_pin)
#define ASHMEM_GET_PIN_STATUS   _IO(__ASHMEMIOC, 9)
#define ASHMEM_PURGE_ALL_CACHES _IO(__ASHMEMIOC, 10)

//this is my code to declare the vuln command
#define ASHMEM_VULN _IOR(__ASHMEMIOC, 11, int)

/* this is the fake struct to dereference */
struct my_ops {
    ssize_t (*do_it)(void);
};

/* this is the fake struct initialized to NULL */
static struct my_ops *ops = NULL;

#endif  /* _UAPI_LINUX_ASHMEM_H */
```

Last Analysis: 7/10/15 18:51
CodeChecked: Never

Debugging on the QEMU ranchu

- Compile the kernel with debug info:

  CONFIG_DEBUG_INFO=y
- Run QEMU with -s param (open gdb server localhost 1234).. and that's not possible :) need to run without google wrapper:

```
/home/vito/opt/emulator64/qemu/linux-x86_64/qemu-system-aarch64 -cpu cortex-a57
-machine type=ranchu -m 2048 \
-kernel $1 -append 'console=/dev/ttyAMA0,115200' -monitor stdio -s -S \
-initrd $2/ramdisk.img -drive index=2,id=userdata,file=$2/userdata.img \
-device virtio-blk-device,drive=userdata -device virtio-blk-device,drive=cache \
-drive index=1,id=cache,file=$2/cache.img -device virtio-blk-device,drive=system \
-drive index=0,id=system,file=$2/system.img -netdev user,id=mynet \
-device virtio-net-device,netdev=mynet -show-cursor
```

Tip: console shoud be available by pressing Ctl+Alt+2 or by passing "-monitor stdio"

Debugging the board over serial

- Enable KGDB on the kernel .config:

```
CONFIG_HAVE_ARCH_KGDB=y
CONFIG_KGDB=y
CONFIG_KGDB_SERIAL_CONSOLE=y
```

- Start the kernel with the option kgdb=ttyS0,115200

```
setenv bootargs "kgdbwait kgdboc=ttySAC2,115200"
```

- or set it later with:

```
#echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc
```

- Trigger a sysrq to stop execution

```
#echo g > /proc/sysrq-trigger (or ctrl-a+f+g in minicom)
```

-

gdb debugging on aarch64 - demo

Vulnerable ashmem

How to trigger the vuln

1) Compile trigger.c
2) Load it on the QEMU or Board running the vulnerable kernel
    1)For the board fastboot flash kernel <kernel>
    2)For the emulator use the launch64_nowrap
3) To trigger the NULL pointer deref
    ./trigger -101 (an integer < -100)

Conclusions

- Dronity will be available on my github (only if..)
- I would really like to get cool developers input
- Try to talk more with Mr. Dave Jones
- Dreaming enough, would be nice to create a community and develop more alternative methods...
- ...