ANGRY MODULE EXCAVATION

LET'S PLAY WITH DUCT TAPE.

Stanislas 'P1kachu' Lejay

LSE Week - July 14, 2016

LSE Security System
Laboratory of Epita

MODULE EXCAVATION ?

Use concolic analysis to explore kernel modules and get informations about their IOCTLs

WHAT IS AN IOCTL ?

```
long random_ioctl(int fd, unsigned int cmd, unsigned long arg);
```

- A syscall to get custom operations on a resource
- Device specific commands, code or specs needed
- Unavailable for private drivers

BUT, WHY ?

CHECK IF HEADERS AND IOCTLS MATCH

```
// linux/include/uapi/linux/firewire-cdev.h

#define FW_CDEV_IOC_GET_INFO           _IOWR('#', 0x00, struct fw_cdev_get_info)
#define FW_CDEV_IOC_SEND_REQUEST        _IOW('#', 0x01, struct fw_cdev_send_request)
#define FW_CDEV_IOC_ALLOCATE           _IOWR('#', 0x02, struct fw_cdev_allocate)
#define FW_CDEV_IOC_DEALLOCATE          _IOW('#', 0x03, struct fw_cdev_deallocate)
#define FW_CDEV_IOC_SEND_RESPONSE       _IOW('#', 0x04, struct fw_cdev_send_response)
#define FW_CDEV_IOC_INITIATE_BUS_RESET  _IOW('#', 0x05, struct fw_cdev_initiate_bus_reset)
#define FW_CDEV_IOC_ADD_DESCRIPTOR     _IOWR('#', 0x06, struct fw_cdev_add_descriptor)
#define FW_CDEV_IOC_REMOVE_DESCRIPTOR   _IOW('#', 0x07, struct fw_cdev_remove_descriptor)
#define FW_CDEV_IOC_CREATE_ISO_CONTEXT _IOWR('#', 0x08, struct fw_cdev_create_iso_context)
#define FW_CDEV_IOC_QUEUE_ISO          _IOWR('#', 0x09, struct fw_cdev_queue_iso)
#define FW_CDEV_IOC_START_ISO           _IOW('#', 0x0a, struct fw_cdev_start_iso)
#define FW_CDEV_IOC_STOP_ISO            _IOW('#', 0x0b, struct fw_cdev_stop_iso)
```

IOCTL COMMANDS CONTAIN DATA

```
// linux/include/uapi/linux/firewire-cdev.h
#define FW_CDEV_IOC_GET_INFO           _IOWR('#', 0x00, struct fw_cdev_get_info)
```

```
// linux/include/uapi/asm-generic/ioctl.h
#define _IOC(dir,type,nr,size) \
        (((dir)  << _IOC_DIRSHIFT) | \
         ((type) << _IOC_TYPESHIFT) | \
         ((nr)   << _IOC_NRSHIFT) | \
         ((size) << _IOC_SIZESHIFT))

#ifndef __KERNEL__
#define _IOC_TYPECHECK(t) (sizeof(t))
#endif

/* used to create numbers */
#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)))
```

STILL DOESN'T TELL US WHY...

- To find bugs
- To find vulnerabilities (Yay)
- To discover IOCTLs from private drivers

AND, AS A BONUS

Experience and challenge this kind of analysis in a new context

A.K.A not in a userland CTF exercise

THE PEELER: STEPS

- Find the functions accurately
- Find which commands are valid
- Find a way to determine the type of 'arg'

ANGR

Framework developped by the UC Santa Barbara's Computer Security Lab, and their associated CTF team, Shellphish.

WHAT IS IT ?

angr is a framework for analyzing binaries. It focuses on both static and dynamic symbolic ("concolic") analysis, making it applicable to a variety of tasks.

Participated in the DARPA CGC (Autonomous Hacking) - One of the 7 team qualified for the finals

Submodules: CLE, claripy, simuvex...

CONCOLIC ?

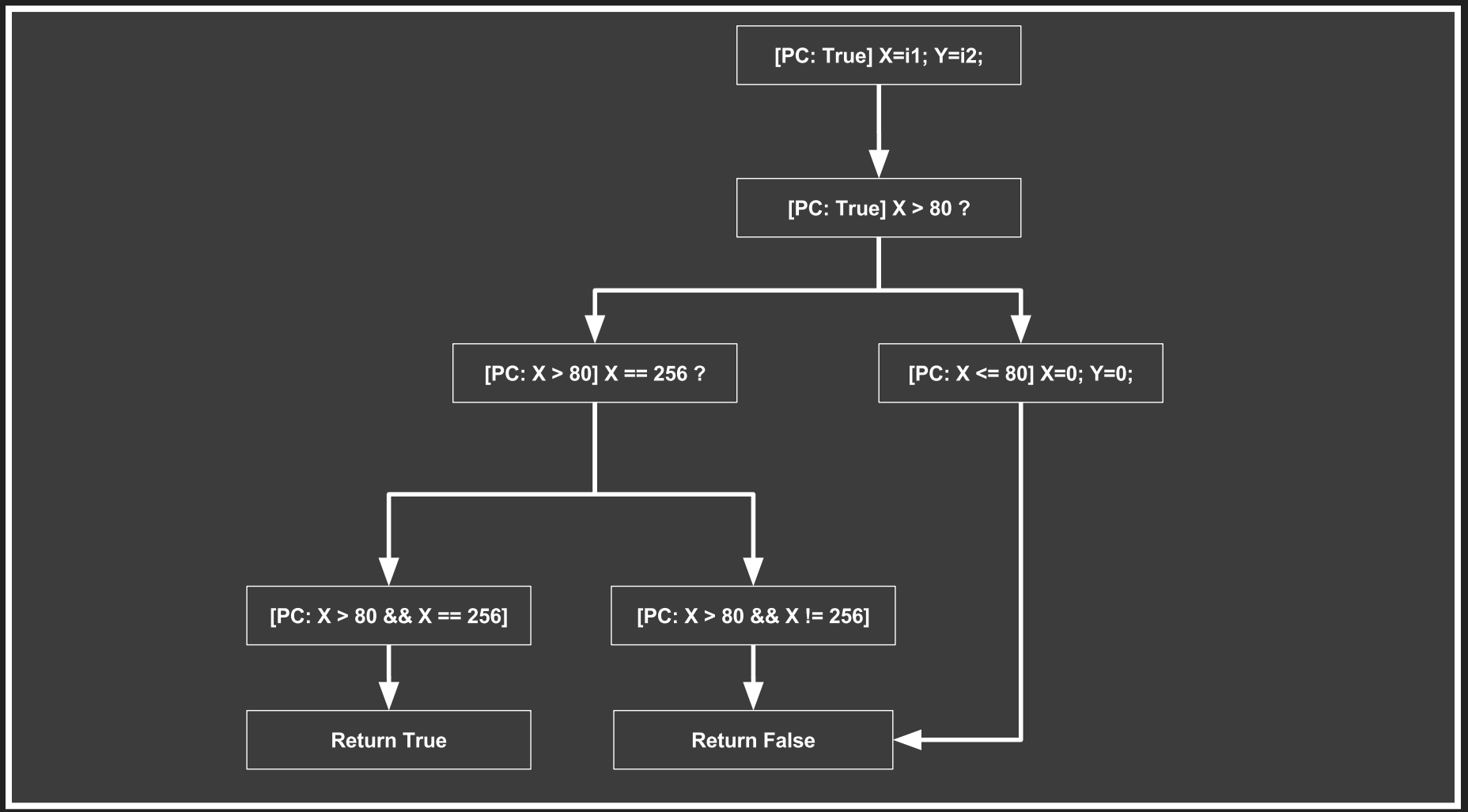
Concrete execution + Symb**olic** execution

- Concrete execution: Program being executed
- Symbolic execution allows at a time T to determine for a branch all conditions necessary to take the branch or not

EXAMPLE

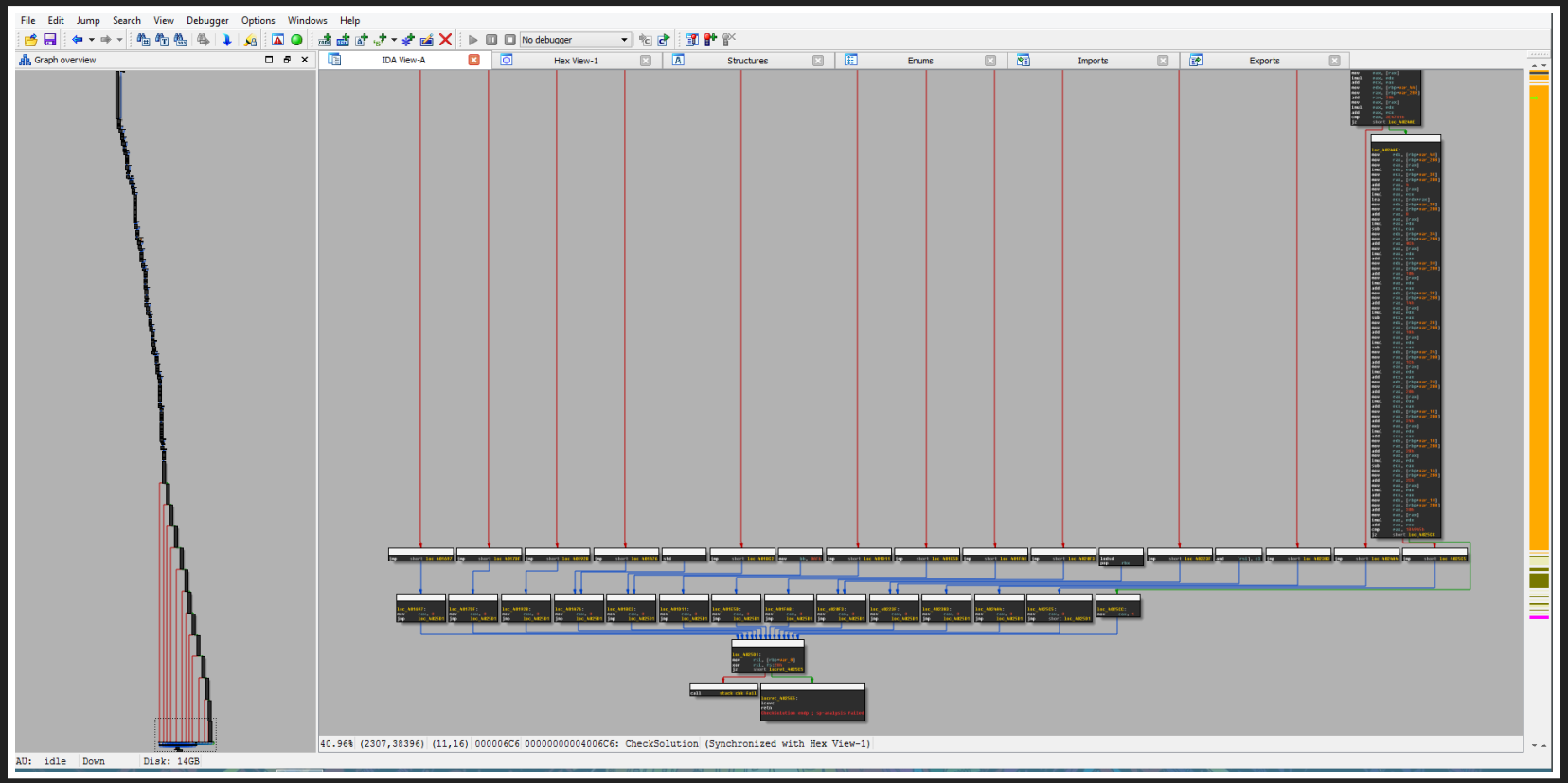
```
int example(int x, int y)
{
    int x = i1;
    int y = i2;

    if (x > 80) {
        if (x == 256)
            return True;
    } else {
        x = 0;
        y = 0;
    }
    return False;
}
```

GIVES US


[PC: True] X=i1; Y=i2;
[PC: True] X > 80 ?
[PC: X > 80] X == 256 ?
[PC: X <= 80] X=0; Y=0;
[PC: X > 80 && X == 256]
[PC: X > 80 && X != 256]
Return True
Return False

PRACTICAL EXAMPLE

Defcon Quals 2016 - babyre



Solved in 5 minutes with angr:

```
main          = 0x4025e7

p = angr.Project('baby-re')
init = p.factory.blank_state(addr=main)
```

```
# Taken from IDA's xrefs
scanf_off = [0x4d, 0x85, 0xbd, 0xf5, 0x12d, 0x165, 0x19d, 0x1d5,
             0x20d, 0x245, 0x27d, 0x2b5, 0x2ed]

def scanf(state):
    state.mem[state.regs.rsi:] = state.se.BVS('c', 8)

for o in scanf_off:
        p.hook(main + o, func=scanf, length=5)
```

```
pgp = p.factory.path_group(init, threads=8)

win           = 0x4028e9
fail          = 0x402941
ex = pgp.explore(find=(win), avoid=(fail))
```

```
s = ex.found[0].state

flag_addr     = 0x7fffffffffeff98 # First rsi from scanf
flag = s.se.any_str(s.memory.load(flag_addr, 50))
print("The flag is '{0}'".format(flag))
```

SO ?

- It seems to do everything we ask for
- Good results in CTF
- Most of the work has been put in the ELF handling

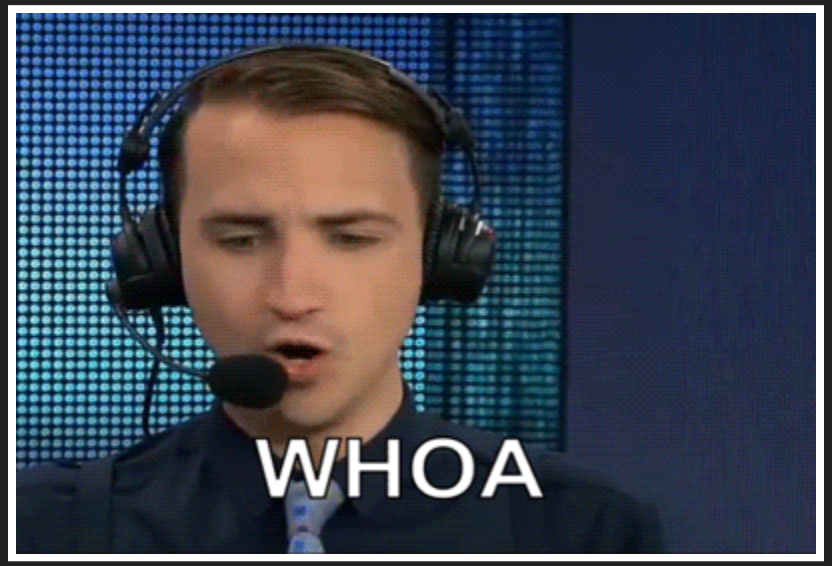
BUT

(YES, THERE IS A BUT...)

(AGAIN...)

Apparently it doesn't like kernel modules, you need to write a custom loader

-- Gaby



PROBLEMS

- Object files (modules) are different from executables
- Relocations had to be done

RELOCATIONS

- References to symbols in other sections
- Need to be resolved at link time

EXAMPLE

```
;; x.o

.text:
    f:
        call external_func      ;; Relocation to external func
        lea eax, inter_section, ;; Inter section relocation
        ret

.data:
    inter_section:
        .long 12
```

```
;; y.o

.text:
    main:
        call f                  ;; Inter object relocation
```

LET'S EXPLORE

Peeler behavior overview:

```
ioctls = find_ioctls("peel_me_sensually.bin");
for (ioctl in ioctls)
{
    endpoints = find_endpoints(ioctl);
    ex = explorer();
    for (endpoint in endpoints)
    {
        paths = get_paths(ex, ioctl.entry, endpoint);
        for (path in paths)
        {
            if (get_ret_val(path) > -1)
                do_stuff(path);
        }
    }
}
```

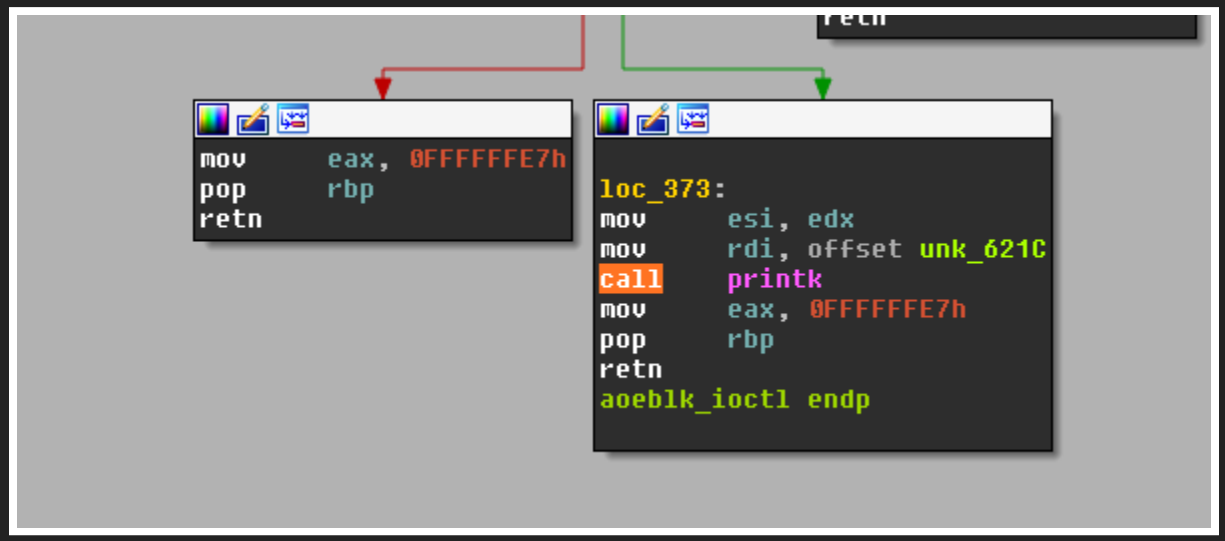
```c
int my_false_ioctl(int fd, unsigned long cmd, void* arg) {

    int ret = -1;

    switch (cmd) {
        case 0xcafe:
            ret = 1 * 2 + 98 - 3000;

            if (ret + fd - 23 + cmd == 0xa110c)
                ret = 1;
    }
    return ret;
}
```

gives us

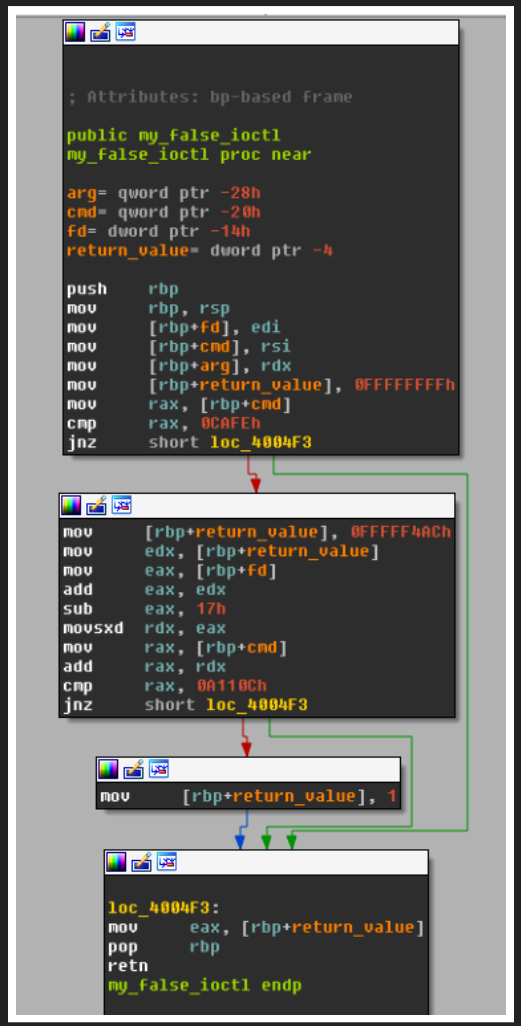
```
Path from 0x4005dc to 4006a7
  Required conditions (constraints):
    <Bool reg_40_5_64 == 0xcafe>
    <Bool (reg_40_5_64 + SignExt(32, ((reg_48_4_64 + 0xfffff4ac) - 0x17))) == 0xa110c>
  Simplified: <Bool (reg_40_5_64 == 0xcafe)  \
                && (reg_48_4_64 == 0x95179) \
                && ((0xfffff495 + reg_48_4_64[31:0])[31:31] == 0))>
  return value: 0x1
```

MINOR FIXES

The intra-block address patch



HEY, IT WOR-- ... WAIT A MINUTE.

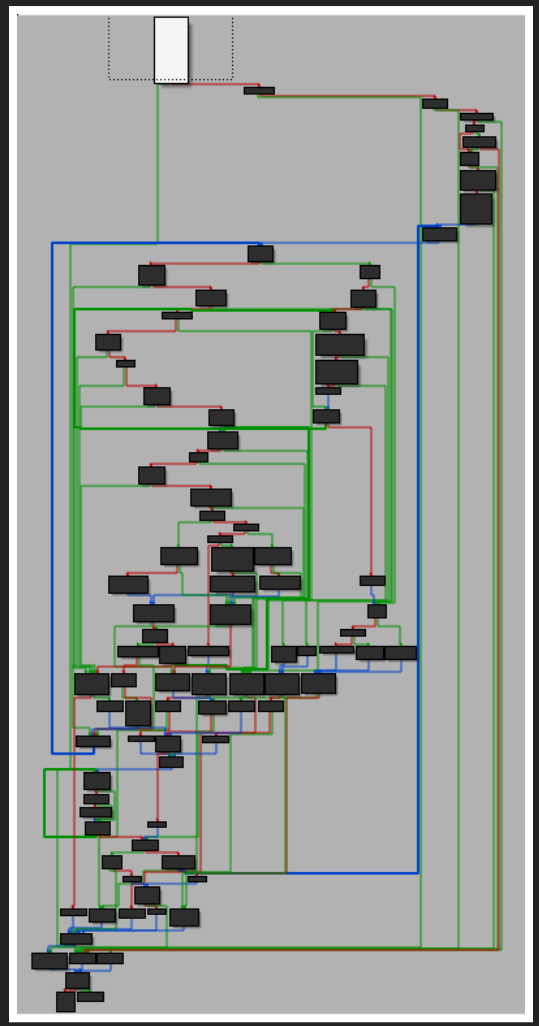


```
; Attributes: bp-based frame

public my_false_ioctl
my_false_ioctl proc near

arg= qword ptr -28h
cmd= qword ptr -20h
fd= dword ptr -14h
return_value= dword ptr -4

push    rbp
mov     rbp, rsp
mov     [rbp+fd], edi
mov     [rbp+cmd], rsi
mov     [rbp+arg], rdx
mov     [rbp+return_value], 0FFFFFFFFh
mov     rax, [rbp+cmd]
cmp     rax, 0CAFEh
jnz     short loc_4004F3
```

```
mov     [rbp+return_value], 0FFFFF4ACh
mov     edx, [rbp+return_value]
mov     eax, [rbp+fd]
add     eax, edx
sub     eax, 17h
movsxd  rdx, eax
mov     rax, [rbp+cmd]
add     rax, rdx
cmp     rax, 0A110Ch
jnz     short loc_4004F3
```

```
mov     [rbp+return_value], 1
```

```
loc_4004F3:
mov     eax, [rbp+return_value]
pop     rbp
retn
my_false_ioctl endp
```



(drm.ko)

DRM_MODE_ATOMIC_IOCTL

```
p1kachu@GreenLabOfGazon:src$ ./pyfinder.py drm.ko -f drm_mode_atomic_ioctl -q
[ ]   INFOS   Peeling drm's ioctls

[ ]   INFOS   Analyzing function drm_mode_atomic_ioctl at 0x421b30
[ ]   INFOS   Launching path_group explorer
[ ]   INFOS   Explorer: <PathGroup with 1 deadended, 1 found>

[ ]   INFOS   Analyzing 1 found paths
[ ]   INFOS   Path from 0x421b30 to 0x421f12L (1/1)
[ ]   INFOS   Return value would be 0xfffffffeaL - Skipping

[ ]   INFOS   Analyzing 1 deadended paths
[ ]   INFOS   Path from 0x421b30 to 0x421ba7L (1/1)
[-]    FAIL   Something went wrong in se.min/max: Unsat Error
[ ]   INFOS   End of analysis
```

DRM_COMPAT_IOCTL

```
p1kachu@GreenLabOfGazon:src$ ./pyfinder.py drm.ko -f drm_compat_ioctl -q
[ ]   INFOS   Peeling drm's ioctls

[ ]   INFOS   Analyzing function drm_compat_ioctl at 0x422490
[ ]   INFOS   Launching path_group explorer
[ ]   INFOS   Explorer: <PathGroup with 5 deadended, 2 active, 1 found>

[ ]   INFOS   Analyzing 1 found paths
[ ]   INFOS   Path from 0x422490 to 0x4224bdL (1/1)
[ ]   INFOS   Return value would be 0xfffffffffffffedL - Skipping

[ ]   INFOS   Analyzing 5 deadended paths
[ ]   INFOS   Path from 0x422490 to 0x405b44L (1/5)
[-]    FAIL   Something went wrong in se.min/max: Unsat Error
[ ]   INFOS   Path from 0x422490 to 0x4059f5L (2/5)
[-]    FAIL   Something went wrong in se.min/max: Unsat Error
[ ]   INFOS   Path from 0x422490 to 0x423e4eL (3/5)
[ ]   INFOS   Return value would be 0xfffffff2L - Skipping
[ ]   INFOS   Path from 0x422490 to 0x405b44L (4/5)
[-]    FAIL   Something went wrong in se.min/max: Unsat Error
[ ]   INFOS   Path from 0x422490 to 0x4059f5L (5/5)
[-]    FAIL   Something went wrong in se.min/max: Unsat Error
[ ]   INFOS   Explorer: <PathGroup with 2 deadended>
```

__KSTRTAB_DRM_IOCTL_PERMIT

```
[ ]   INFOS   Analyzing function __kstrtab_drm_ioctl_permit at 0x4399ce
Traceback (most recent call last):
  File "./pyfinder.py", line 201, in <module>
      recover_function(f, cfg, addr)
  File "/home/p1kachu/peeling-ioctls/src/excavator.py", line 195, in recover_function
      ins = blk.capstone.insns[last_ins]
IndexError: list index out of range
```

WHAT NOW ?

- We need to enhance and strengthen the peeler
- angr cannot work without some human pre-work
- How to save resources (time and memory) ?
  - Automate verifications
  - Discard useless stuff
  - Analyze interesting functions only

FIND IOCTLS SMARTLY

HOW DO WE DO THAT ?

IOCTL REGISTRATION PROCESSUS

- Create a struct file_operations
  - Multiple function pointers
  - Used to register operations on the device
- Load the structure in memory (using a register function)
- Classic operations will now be handled by these functions

```
static const struct file_operations i8k_fops = {
        .owner          = THIS_MODULE,
        .open           = i8k_open_fs,
        .read           = seq_read,
        .llseek         = seq_lseek,
        .release        = single_release,
        .unlocked_ioctl = i8k_ioctl,
};
```

LEGEND

```
For the next slides, please refer to this legend:
- fops           : file_operations struct containing our ioctl
- register_ioctl : register function that will load fops
- call_me_addr   : address of 'call register_ioctl'
- Caller         : function containing call_me_addr
```

GOAL: FIND FOPS

```
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = (void*)my_ioctl,
    .compat_ioctl = (void*)my_ioctl
};
```

Data of interest: Its address in memory.

LOOK FOR REGISTER_IOCTL

Iterate over imported symbols to look for one of these:

```
register_chrdev(DRM_MAJOR, "drm", &drm_stub_fops);
```

```
register_functions = [
                     '__register_chrdev',
                     'misc_register',
                     'cdev_init'
]
```

Data of interest: Exact address of 'call register_ioctl' (call_me_addr).

FIND WHICH FUNCTION CALLS REGISTER_IOCTL (CALLER)

```
"""
- The 'call register_ioctl' will always be in the init function.
- The init function will always be called init_module.
"""
```

```
"""
- The 'call register_ioctl' will always be in the init function.
- The init function will always be called init_module.

EDIT : No, and no.
"""

for _, sym in elf.symbols_by_addr.iteritems():
    bottom = sym.rebased_addr
    top    = sym.rebased_addr + sym.size
    if register_ioctl > bottom and register_ioctl <= top:
        Caller = sym.name
```

Data of interest: Entry point of Caller.

AND NOW ?

```
We:
* have Caller's entry point
* have register_ioctl's call address (call_me_addr, in Caller)
* know that when register_ioctl is called, fops is in a register
```

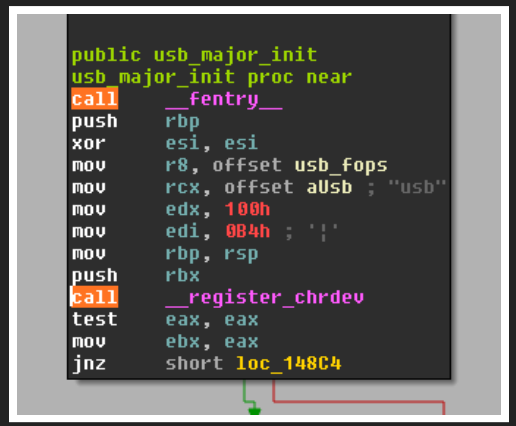
```
So we:
* Launch a path explorer from Caller's entry point to call_me_addr
* Break just before the call
* Analyze the passed arguments to get fops address
```

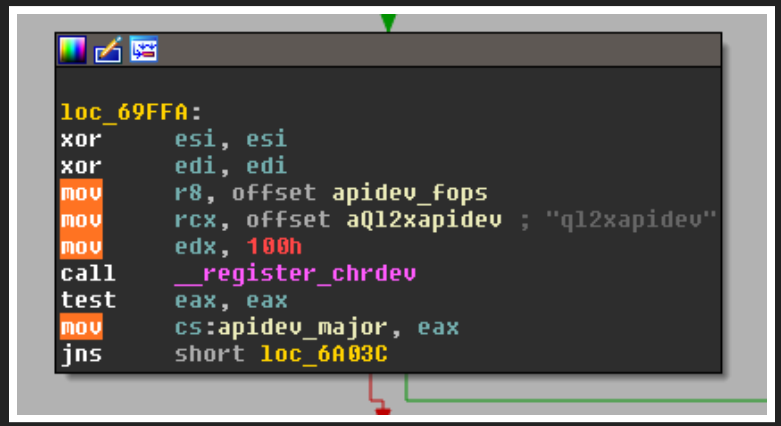
PROBLEMS

- Very long Callers
- Lots of unresolved functions
- Memory and time consuming
- Calling 'conventions'

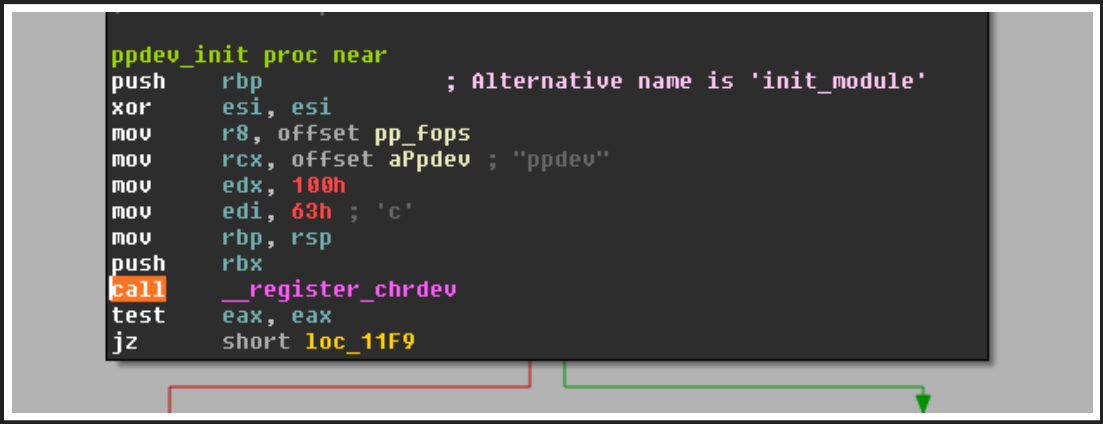
LET'S TRY TO BE CLEVER

Assignations and call usually are in the same block





- Create a CFG of Caller
- Find the basic block containing call_me_addr
- Path explorer from the beginning of the block only



Right.

It doesn't work.

TROUBLES WITH INCOMPLETE CFG

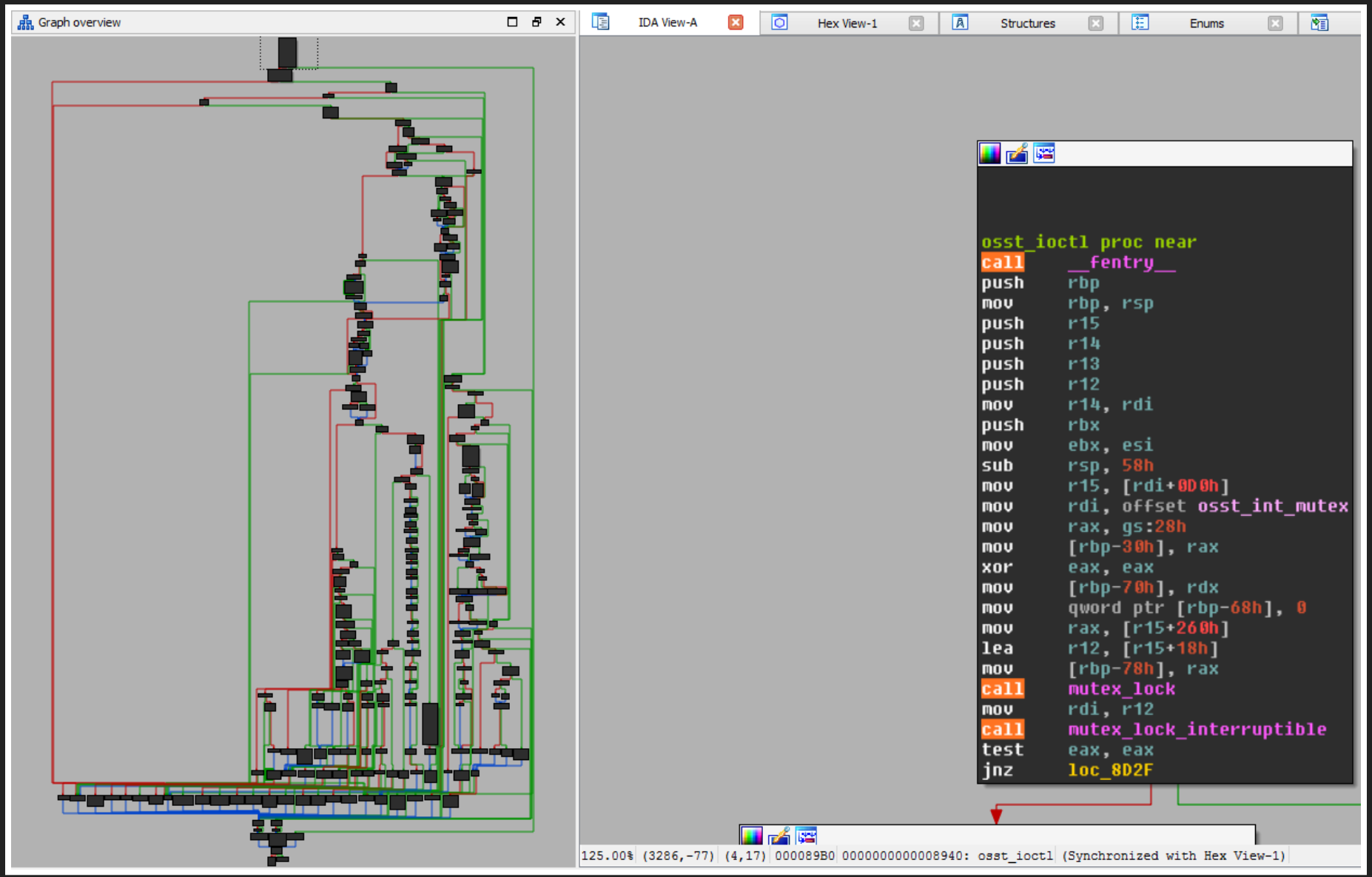
```
import angr
p = angr.Project('/home/p1kachu/Desktop/modules/chrdev/osst.ko')
for x, y in p.loader.main_bin.symbols_by_addr.iteritems():
    if 'osst_ioctl' in y.name:
        print(hex(y.rebased_addr), y.name)
```

```
('0x408970', u'osst ioctl')
```

```
f = p.analyses.CFGAccurate(starts=[0x408970]).functions[0x408970]
```

```
print([(hex(x.addr), x.size) for x in f.blocks])
```

```
[('0x408970L', 5), ('0x408975L', 83)]
```


Graph overview
IDA View-A
Hex View-1
Structures
Enums
osst_ioctl proc near
call __fentry__
push rbp
mov rbp, rsp
push r15
push r14
push r13
push r12
mov r14, rdi
push rbx
mov ebx, esi
sub rsp, 58h
mov r15, [rdi+0D0h]
mov rdi, offset osst_int_mutex
mov rax, gs:28h
mov [rbp-30h], rax
xor eax, eax
mov [rbp-70h], rdx
mov qword ptr [rbp-68h], 0
mov rax, [r15+260h]
lea r12, [r15+18h]
mov [rbp-78h], rax
call mutex_lock
mov rdi, r12
call mutex_lock_interruptible
test eax, eax
jnz loc_8D2F
125.00% (3286,-77) (4,17) 000089B0 0000000000008940: osst_ioctl (Synchronized with Hex View-1)

Problems with relative calls, unresolved symbols, symbolic memory...

ANOTHER WAY: DWARF DEBUG INFOS

If present, DWARF infos *can not* fail

Iterate over them, find fops in memory, and get the IOCTLs

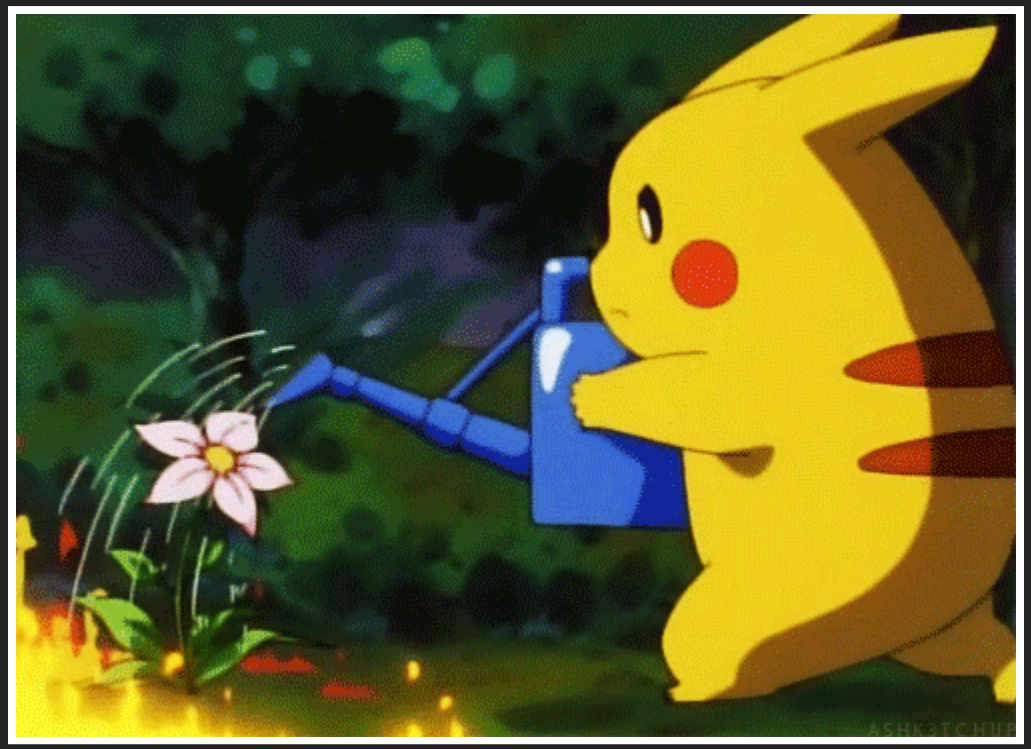
BEST EFFORT STRATEGY

1. look in debug infos (DWARF)
    * Accurate and fast
    * Need to have access to the source code

2. Fallback on the file_operations structure
    * Slow
    * Requires an angr explorer and CFG for itself
    * Often fails at some point

3. Fallback on symbols names
    * Some IOCTLs aren't named like that
    * More functions to analyze

WHAT'S LEFT TO DO ?



```
* Get infos about the *arg* parameter
    * What's its type ?
    * Which operations are applied on it ?
```

POSSIBLE IMPROVEMENT

- Efficiency boost (merge paths, discard others, ...)
- Allow user input for testing
- Sanity checking by parsing headers

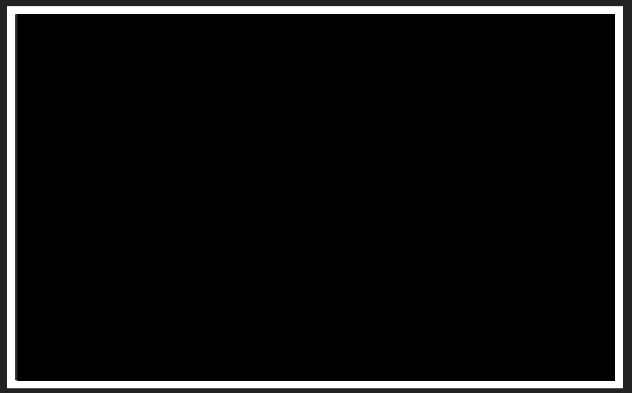
SUM UP

- Not usable for big/complicated modules
- Would need more layers of fallbacks
- Everything is too unstable to be used at once

However, still interesting to see angr on real life problems

- More details:
  - Linux Device Drivers - Chapter 6
  - http://github.com/angr
  - SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis (UCSB)
  - Binary analysis - Concolic Execution (Jonathan Salwan)

Thank you



p1kachu@lse.epita.fr - @0xP1kachu