



SDC · 2020

# 安全开发者峰会

10.23 | 上海

# 麒麟框架：现代化的逆向分析体验

孔子乔 武晨旭 京东牧者安全实验室

# 目录

1. 演讲者介绍
2. 麒麟框架介绍
3. MBR插桩分析
4. 麒麟框架调试层支持
5. 反OLLVM平坦化
6. 展望

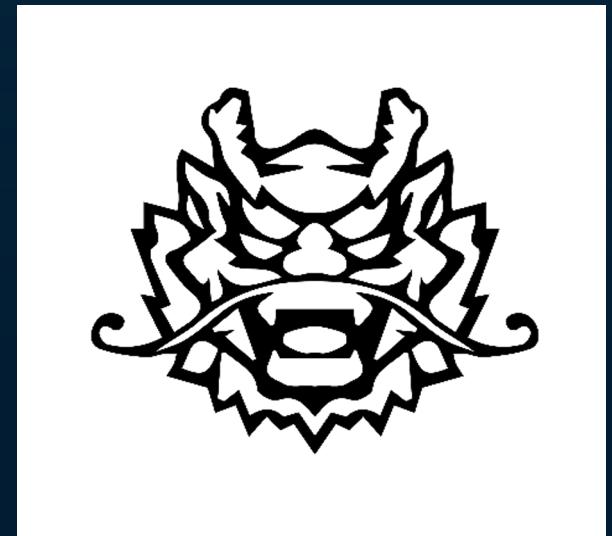
# 演讲人介绍

## 演讲者介绍

孔子乔。京东牧者安全实验室安全工程师，Lancet战队成员，GeekPwn 2019名人堂，Blackhat演讲者，麒麟框架核心成员之一。

武晨旭。京东牧者安全实验室安全工程师，Blackhat演讲者，麒麟框架核心成员之一。

KJ。京东牧者安全实验室创始人，麒麟框架创始人，各种安全会议演讲常客。



# 麒麟框架介绍

## 麒麟框架介绍：总览

- 在沙箱环境内模拟执行二进制文件。
- 在模拟的基础上提供统一的分析API。
- 插桩分析、快照、系统调用和API劫持等。
- 跨系统：Windows, MacOS, Linux, BSD, UEFI, MBR
- 跨架构：x86, x86\_64, arm, arm64, mips, 8086
- 跨二进制：PE, MachO, ELF, UEFI(PE), COM
- GDB 调试支持以及我们自己的调试器 Qdb。
- IDA 插件赋予 IDA 动态插桩分析能力。

	8086	x86	x86-64	ARM	ARM64	MIPS
Windows (PE)	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
Linux (ELF)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
MacOS (MachO)	-	<input type="checkbox"/>	<input checked="" type="checkbox"/>	-	-	-
BSD (ELF)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
UEFI	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	-	-	-
DOS (COM)	<input checked="" type="checkbox"/>	-	-	-	-	-
<b>Architecture independent</b>						
MBR				<input checked="" type="checkbox"/>		

## 麒麟框架介绍：总览

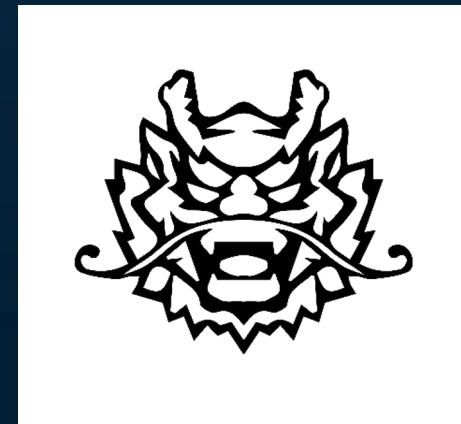
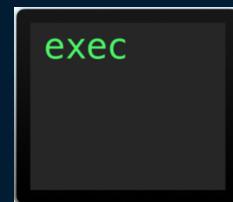
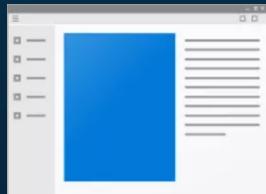
Windows PE

DOS COM

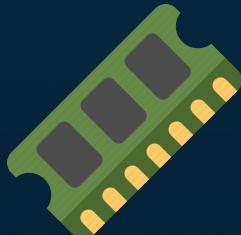
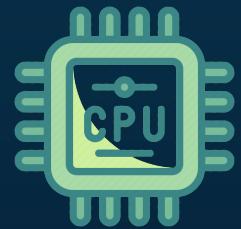
UEFI Executable

Linux ELF

MacOS MachO



Qiling Framework



Instrumentation API

Memory API

Snapshot API

Disk API

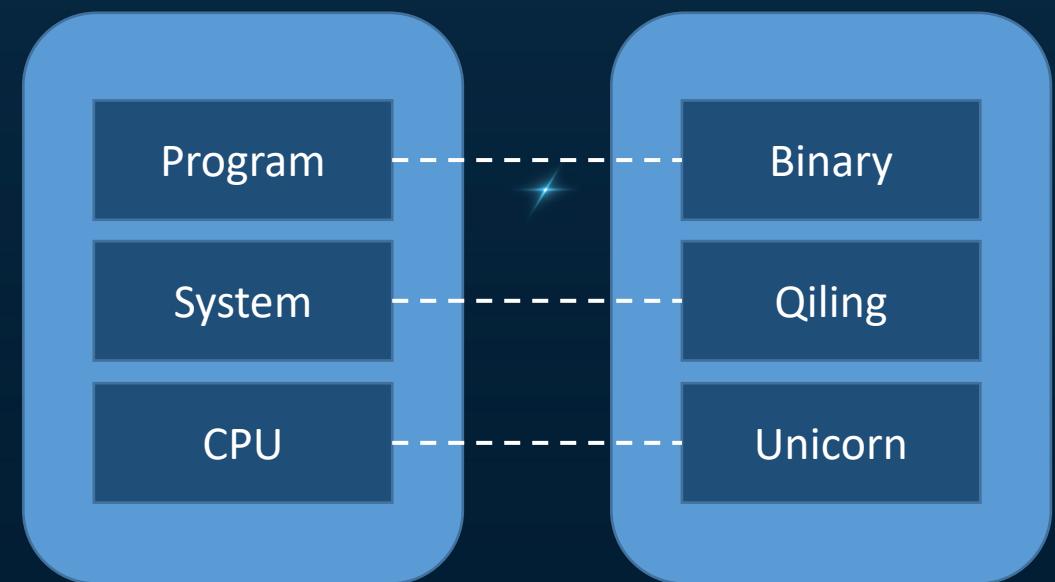
## 麒麟框架介绍：模拟实现

### Unicorn

- 基于 Qemu 的 CPU 模拟器
- 对上层系统无任何感知

### Qiling

- 解析二进制，设置内存
- 实现关键的系统调用
- 从二进制文件的入口点开始执行



# 麒麟框架介绍：架构设计

## Arch

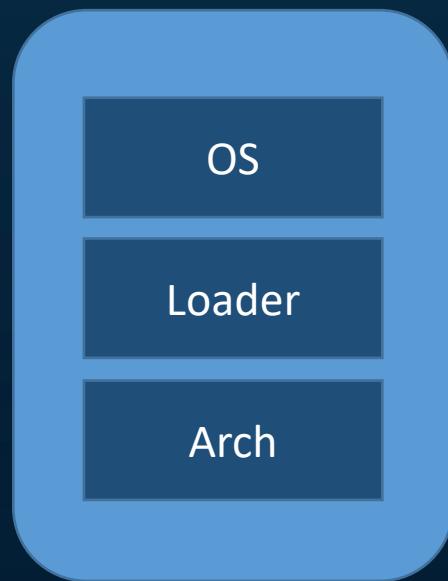
- 设置寄存器、内存、栈等
- 负责架构相关的底层实现，比如x86的GDT

## Loader

- 解析二进制，判断系统类型和架构类型，重定位
- 解析代码段加载进内存，设置堆栈
- 为运行程序做好准备工作

## OS

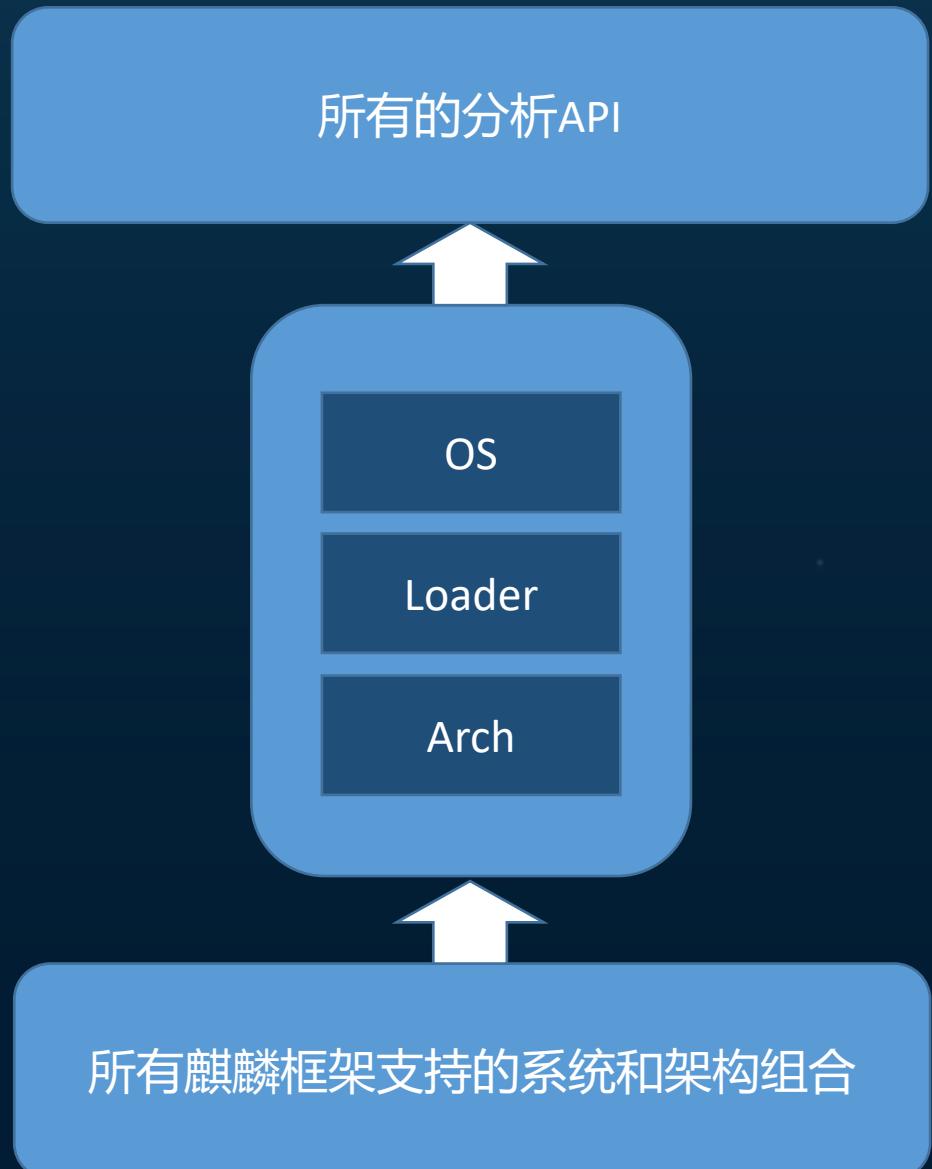
- 核心实现，包括了系统调用的实现
- 提供操作系统相关抽象，比如磁盘、文件路径等等



Qiling

## 麒麟框架介绍：分析API

- Registers API 读取/设置寄存器
  - ql.reg.eax = 1
- Memory API 读取/设置/搜索 内存
  - ql.mem.read / ql.mem.write
  - ql.mem.search
- Hook API 插桩分析
  - ql.hook\_code / ql.hook\_mem
  - ql.set\_syscall / ql.set\_api
- Disk API 磁盘/路径模拟
  - ql.os.add\_fs\_mapper
- Snapshot API 快照保存/恢复
  - ql.save / ql.restore



# M B R 插 桩 分 析

## MBR插桩分析

- MBR(Master Boot Record)
- Arch层实现 20 位地址线的转换
- Loader层解析COM文件或者MBR Dump
  - 同时设置硬盘的映射
- OS层实现了BIOS中断
  - 通过 ncurses 来实现BIOS图形服务和键盘服务
  - 通过硬盘映射来实现磁盘服务
- 目前唯一一个可以对MBR进行插桩分析的框架

## MBR插桩分析：样本分析

- Flare-On Challenge 8 – doogie
- MBR file
- 使用 qltool 可以直接运行
  - python3 qltool run -f examples/rootfs/8086/doogie/doogie.bin --rootfs examples/rootfs/8086/ --console False
- 输入后会得到一些乱码
- 注意到有一个日期 Feb 06, 1990

```
~/q/e/r/8/doogie (doogie|...) $ file doogie.bin
doogie.bin: DOS/MBR boot sector; partition 1 : ID=0x7, activ
e, start-CHS (0x0,32,33), end-CHS (0x3ff,254,63), startsecto
r 2048, 41938944 sectors
~/q/e/r/8/doogie (doogie|...) $
```

```
python3 /Users/mio/qiling
```

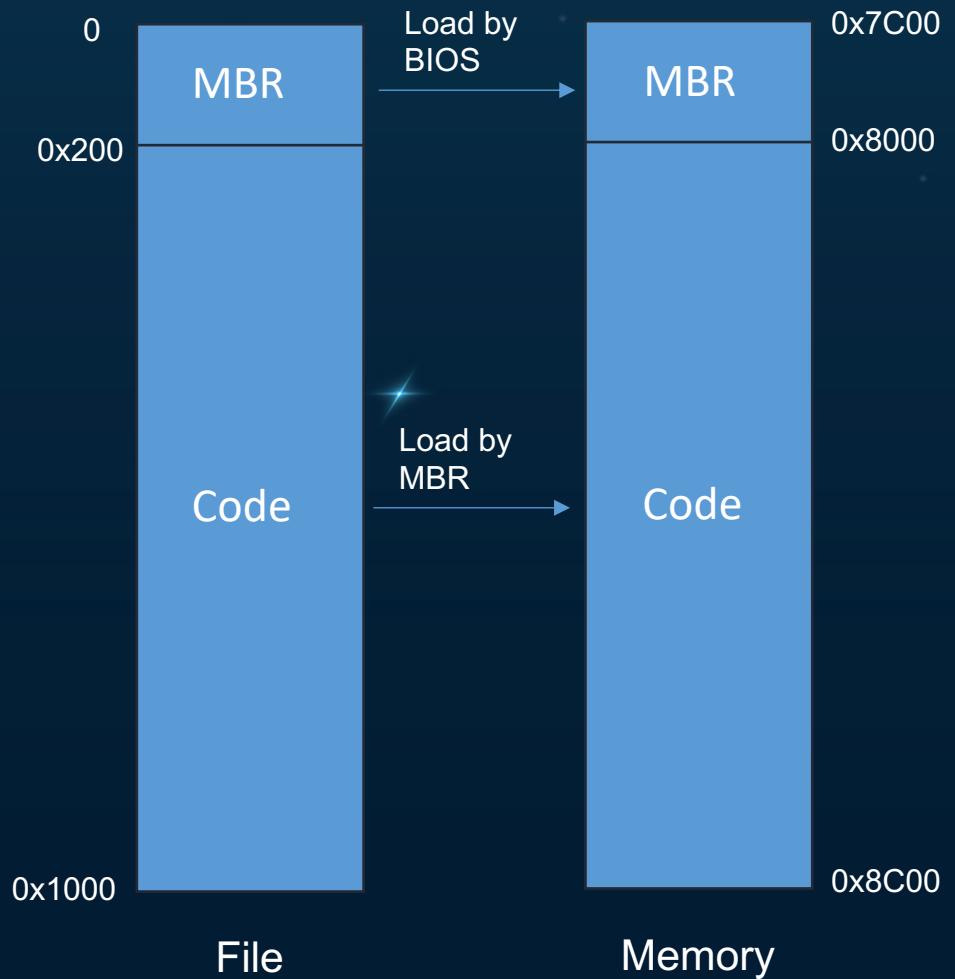
```
February 06, 1990... Despite being a 16-year-old reverse
engineering genius, I seem to have forgotten the password to
my PC. Can you help me???
```

```
fish /Users/mio/qiling
```

```
Y FF 0A }0~Vdr\ c0^?mK sJ cE a@ tX dU ukL iV gwS xm jD ^?? 1Z-Gtf3 ^OT nH hd io
10 ^FA^
~/qiling (doogie_fix_crlf|...) $
```

## MBR插桩分析：样本分析

- 和其他MBR代码一样，运行分为两阶段
  - 第一阶段 BIOS 把 MBR 加载到 0x7c00开始执行
  - 第二阶段 MBR 把剩余代码通过磁盘服务加载到内存，然后跳转执行



# MBR插桩分析：核心逻辑

- 分析的重点在第二阶段的代码
  - 通过系统调用获取当前时间
  - 把当前时间和固定字符串异或
  - 读取输入
  - 把输入和刚才的结果异或
  - 输出两次异或的结果

```

seg000    segment byte public 'CODE' use16
assume cs:seg000
;org 8000h
assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
call sub_805B
push ds:word_87F2
call get_date_and_write_to_87EE
push 4
push offset date_87EE
call xor8809
add sp, 4
push 87F4h
call read_input
add sp, 4
push ax
push 87F4h
call xor8809
add sp, 4
call initialize_screen
push 0
push 8809h
call print_ascii_art
add sp, 4
mov cx, 2607h
mov ah, 1
int 10h          ; - VIDEO - SET CURSOR CHARACTERISTICS
; CH bits 0-4 = start line for cursor in character cell
; bits 5-6 = blink attribute
; CL bits 0-4 = end line for cursor in character cell
loc_803D:        hlt
; ----- ; CODE XREF: seg000:803E↓j
;           jmp short loc_803D

```

# MBR插桩分析：实现

- 麒麟框架的工作：
  - 模拟MBR文件运行
  - Hook BIOS中断 INT 1A 返回 Feb 06, 1990
  - 动态读写内存获取缓冲区的字符串
  - 通过部分执行和程序快照自动测试所有可能的 Key
- 破解的过程分为三个阶段

- Hook API
- Partial execution
- FS Mapper
- Memory API

```

if __name__ == "__main__":
    ql = first_stage()
    # resume terminal
    curses.endwin()
    keys = second_stage(ql)
    for key in keys:
        print(f"Possible key: {key}")
    # The key of this challenge is not unique. The real
    # result depends on the last ascii art.
    print("Going to try every key.")
    time.sleep(3)
    third_stage(keys)
    # resume terminal
    curses.endwin()
  
```

- Hook API
- Partial execution
- Memory API
- Snapshot API

获取和指定日期异或后的字符串

读取内存并且搜索所有可能的Key

测试每一个Key

# MBR插桩分析：第一阶段

- ql.set\_api hook BIOS 中断来设置时间
- ql.os.fs\_mapper 映射磁盘
- 在和日期异或完后停止执行

```
# In this stage, we get the encrypted data which xored with the specific date.
def first_stage():
    ql = Qiling(["rootfs/8086/doogie/doogie.bin"],
                "rootfs/8086",
                console=False,
                log_dir=".")  
    ql.add_fs_mapper(0x80, QlDisk("rootfs/8086/doogie/doogie.bin", 0x80))
    # Doogie suggests that the datetime should be 1990-02-06.
    ql.set_api((0x1a, 4), set_required_datetime, QL_INTERCEPT.EXIT)
    # A workaround to stop the program.
    hk = ql.hook_code(stop, begin=0x8018, end=0x8018)
    ql.run()
    ql.hook_del(hk)
    return read_until_zero(ql, 0x8809)
```

February 06, 1990... Despite being a 16-year-old reverse engineering genius, I seem to have forgotten the password to my PC. Can you help me???

```
def set_required_datetime(ql: Qiling):
    ql.nprint("Setting Feburary 06, 1990")
    ql.reg.ch = BIN2BCD(19)
    ql.reg.cl = BIN2BCD(1990%100)
    ql.reg.dh = BIN2BCD(2)
    ql.reg.dl = BIN2BCD(6)
```

## MBR插桩分析：第二阶段

- 使用 Memory API 读取内存
- 猜测 Key 的长度然后在假设最后结果是可打印 ASCII 码的情况下搜索所有有可能的 Key

```
# In this stage, we crack the encrypted buffer.
def second_stage(ql: Qiling):
    data = bytes(read_until_zero(ql, 0x8809))
    key_size = guess_key_size(data) # Should be 17
    seqs = []
    for i in range(key_size):
        seq = b""
        j = i
        while j < len(data):
            seq += bytes([data[j]])
            j += key_size
        seqs.append(seq)
    seqs_keys = cal_count_for_seqs(seqs)
    keys = search_possible_key(seqs, seqs_keys)
    return keys

def read_until_zero(ql: Qiling, addr):
    buf = b""
    ch = -1
    while ch != 0:
        ch = ql.mem.read(addr, 1)[0]
        buf += pack("B", ch)
        addr += 1
    return buf
```

# MBR插桩分析：第三阶段

- 部分执行直接跳过输入读取，然后保存快照
- 通过 Memory API 填入 Key
- 从读取后开始部分执行到打印字符
- 观察结果可以得到 Flag

A screenshot of a terminal window titled "python3 /Users/mio/qiling/examples". The window displays a continuous loop of the character sequence "888...888" followed by a current key indicator "Current key: b'ioperateonmalware'". The terminal window has a dark background with light-colored text.

```

def show_once(ql: Qiling, key):
    klen = len(key)
    ql.reg.ax = klen
    ql.mem.write(0x87F4, key)
    # Partial execution to skip input reading
    ql.run(begin=0x801B, end=0x803d)
    echo_key(ql, key)
    time.sleep(3)

# In this stage, we show every key.
def third_stage(keys):
    # To setup terminal again, we have to restart the whole program.
    ql = Qiling(["rootfs/8086/doogie/doogie.bin"],
                rootfs="8086",
                console=False,
                log_dir="."))
    ql.add_fs_mapper(0x80, QlDisk("rootfs/8086/doogie/doogie.bin", 0x80))
    ql.set_api((0x1a, 4), set_required_datetime, QL_INTERCEPT.EXIT)
    hk = ql.hook_code(stop, begin=0x8018, end=0x8018)
    ql.run()
    ql.hook_del(hk)
    # Snapshot API.
    ctx = ql.save()
    for key in keys:
        show_once(ql, key)
        ql.restore(ctx)

```

# 麒麟框架调试层支持

# 逆向调试的痛点

- PC
  - 缺乏HOOK工具
  - OS限制
- IoT
  - 环境搭建困难
  - 调试困难

QEMU

```
[ OK ] Started Regular background program processing daemon.
Starting dhcpcd on all interfaces...
Starting Login Service...
Starting Disable WiFi if country not set...
[ OK ] Started triggerhappy global hotkey daemon.
[ OK ] Started Disable WiFi if country not set.
[ OK ] Started System Logging Service.
[ OK ] Started Avahi mDNS/DNS-SD Stack.
[ OK ] Started Login Service.
[ OK ] Started LSB: Switch to ondemand cpufreq (unless shift key is pressed).
[ OK ] Started Raise network interfaces.

gef> b main
Breakpoint 1 at 0x1040c
gef> c
Continuing.

[ registers ]
$r0      0x00000001 $r1      0xffffef134 $r2      0xffffef13c $r3      0x00010404
$r4      0xfffffeff8 $r5      0x00000000 $r6      0x00000000 $r7      0x00000000
$r8      0x00000000 $r9      0x00000000 $r10     0xff7ee000 $r11     0xfffffefe4
$r12     0xffffef060 $sp      0xfffffefe0 $lr      0xff6ccfe7 $pc      0x0001040c
$cpsr   0x40080010

Flags: [negative ZERO carry overflow interrupt fast thumb]

[ stack ]
[!] [Errno 2] No such file or directory: '/proc/42000'
Python Exception <class 'OSError'> cannot open remote path /proc/42000/maps:
Python Exception <class 'gdb.error'> Error occurred in Python command: cannot
open remote path /proc/42000/maps:
Python Exception <class 'gdb.error'> Error occurred in Python command: Error o
ccurred in Python command: cannot open remote path /proc/42000/maps:

Breakpoint 1, 0x0001040c in main ()
```

[https://blog.csdn.net/song\\_lee](https://blog.csdn.net/song_lee)

# GDB Server

qdb

# IDA Plugin

The screenshot shows the IDA Pro interface with the following panes:

- Library function**: Shows the function tree with nodes like `main`, `init`, `fin`, `gmon_start`, and `gmon_start_`.
- Instruction**: Shows the assembly code for the `main` function. The assembly is color-coded by section:
  - `main`: `.text`
  - `init`: `.text`
  - `fin`: `.text`
  - `gmon_start`: `.text`
  - `gmon_start_`: `.text`
  - `main`: `.text`
  - `init`: `.text`
  - `fin`: `.text`
  - `gmon_start`: `.text`
  - `gmon_start_`: `.text`
- External symbol**: Shows symbols from the `lumina` library.
- Hex View-1**: Shows the raw hex and ASCII data at address `0x000000000048524`.
- Structures**: Shows the structure definition for `lumina`.
- Enums**: Shows the enum definitions for `lumina`.
- Imports**: Shows imported functions.
- Exports**: Shows exported functions.
- IDA View-A**: Shows the assembly view for the current function.
- Line 13 of 22**: Shows the current line of assembly.
- Graph overview**: Shows the control flow graph.
- Registers**: Shows the CPU register values.
- Stack**: Shows the stack contents at `0x7FF3C860`.
- Output window**: Shows assembly output.
- QL Register View**: Shows QL register values.
- QL Stack View**: Shows QL stack values.
- QL Stack View**: Shows QL stack values.
- Python**: Shows the Python console.

# Qiling GDBServer

## Terminal

```

set.SOCK_STREAM)

gdb
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote 127.0.0.1:9999
Remote debugging using 127.0.0.1:9999
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x00409a16 in ?? ()
(gdb) break *0x00408192
Breakpoint 1 at 0x408192
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x00408192
(gdb) c
Continuing.

Breakpoint 1, 0x00408192 in ?? ()
(gdb) disas 0x0408192,0x0408198
Dump of assembler code from 0x408192 to 0x408198:
=> 0x00408192: push %ecx
  0x00408193: push %esi
  0x00408194: call *0x40a138
End of assembler dump.
(gdb) x/s $ecx
0xffffcf14: "http://www.iuquerfsodp9ifjaposdfjhgosurijfaewrwerwgea.com"
(gdb)

```

## GUI

IDA View-EIP

```

text:00408140 sub esp, 50h
text:00408143 push esi
text:00408144 push edi
.text:00408145 mov ecx, 0Eh
text:0040815A mov esi, offset aHTTPWwwIuerfers ; http://www.iuquerfsodp9ifjaposdfjhgosurijfaewrwerwgea.com
text:0040815B lea edi, [esp+0Ch+szURL]
text:0040815C xor eax, eax
text:0040815D rep movsd
text:0040815E movsb
text:0040815F mov [esp+50h+var_17], eax
text:00408160 mov [esp+50h+var_13], eax
text:00408160 mov [esp+50h+var_F], eax
text:00408164 mov [esp+50h+var_B], eax
text:00408168 mov [esp+50h+var_7], eax
text:0040816C mov [esp+50h+var_3], ax
text:00408171 push eax ; dwFlags
text:00408172 push eax ; lpsoProxyBypass
text:00408173 push eax ; lpsoProxy
text:00408174 push 1 ; dwAccessType
text:00408176 push eax ; szsAgent
text:00408177 mov [esp+6Ch+var_1], al
text:00408178 call ds:InternetOpenA
text:00408181 push 0 ; dwContext
text:00408183 push 0 ; dwFlags
text:00408188 push 0 ; dwHeadersLength
text:0040818A lea ecx, [esp+64h+szURL]
text:0040818C mov esi, eax
text:00408190 push 0 ; lpsoHeaders
text:00408192 push ecx ; lpsoUrl
text:00408193 push esi ; hInternet
text:00408194 call ds:InternetOpenUrlA
text:0040819A mov edi, eax
text:0040819C push esi ; hInternet
text:0040819D mov esi, ds:InternetCloseHandle
text:0040819E test edi, edi
text:004081A1 nop
text:004081A6 pop eax
text:004081A7 call esi . InternetCloseHandle
text:004081A9 push 0 ; hInternet
text:004081A9 call esi . InternetCloseHandle

```

Hex View-1

```

00408130 C3 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 Ä.....
00408146 83 EC 58 56 57 B9 0E 00 00 00 BE D0 13 43 00 80 f1PWN...X0.C...
00408156 7C 24 08 33 C0 F3 A4 89 44 24 41 89 44 24 45 $J@XyM@SABDE
00408166 89 44 24 49 89 44 24 4D 89 44 24 51 89 44 24 R$J@XyM@SABDE
00408176 50 50 56 6A 01 50 88 44 24 60 FF 15 34 A1 40 UPPPj.P'DSkY.4!@.
00408186 00 6A 00 68 00 00 84 6A 00 8D 4C 24 18 FF 00 .J...,.J..L$.i.
00408196 6A 00 51 56 FF 15 38 A1 40 00 88 F8 56 88 35 30 j.QVY.8!@.øV<5
004081A6 A1 40 00 85 FF 90 90 F0 6A 00 FF D6 E8 DE FE jE...y..yj.yøhp
004081B6 FF FF F5 33 C0 5E C3 C4 50 C2 10 00 FF D6 57 FF yY..3A^fAPA..yWY
004081C6 D6 57 33 C0 E3 C4 50 C2 10 00 90 90 90 90 90 90 Ø.3A^fAPA.....
004081D6 51 56 88 F1 84 46 50 89 44 24 08 E8 1D 16 00 QVñf.PøD$ë...
004081E6 00 83 C4 03 33 C0 89 46 04 89 46 08 89 46 0C SE fJ..3Aëf.Bë.ë...
004081F6 59 C3 90 90 90 90 90 90 90 90 90 90 90 90 90 YÄ...
00408206 88 44 24 08 88 54 24 04 53 55 56 BB F1 8A 08 57 D$...SUvñS.W

```

Stack view

```

FFFCFC0C 00000000 MEMORY:FFFFCF0C (Synchronized with EIP)
FFFCFC10 05834052 MEMORY:05034052
FFFCFC14 00000000 MEMORY:00000000
FFFCFC18 00000000 MEMORY:00000000
FFFCFC1C 00000000 MEMORY:00000000
FFFCFC20 00000000 MEMORY:00000000
FFFCFC24 00000000 MEMORY:00000000
FFFCFC28 00000000 MEMORY:00000000
FFFCFC2C 00000000 MEMORY:00000000
FFFCFC30 00000000 MEMORY:00000000
FFFCFC34 00000000 MEMORY:00000000
FFFCFC38 00000000 MEMORY:00000000
FFFCFC40 00000000 MEMORY:00000000

```

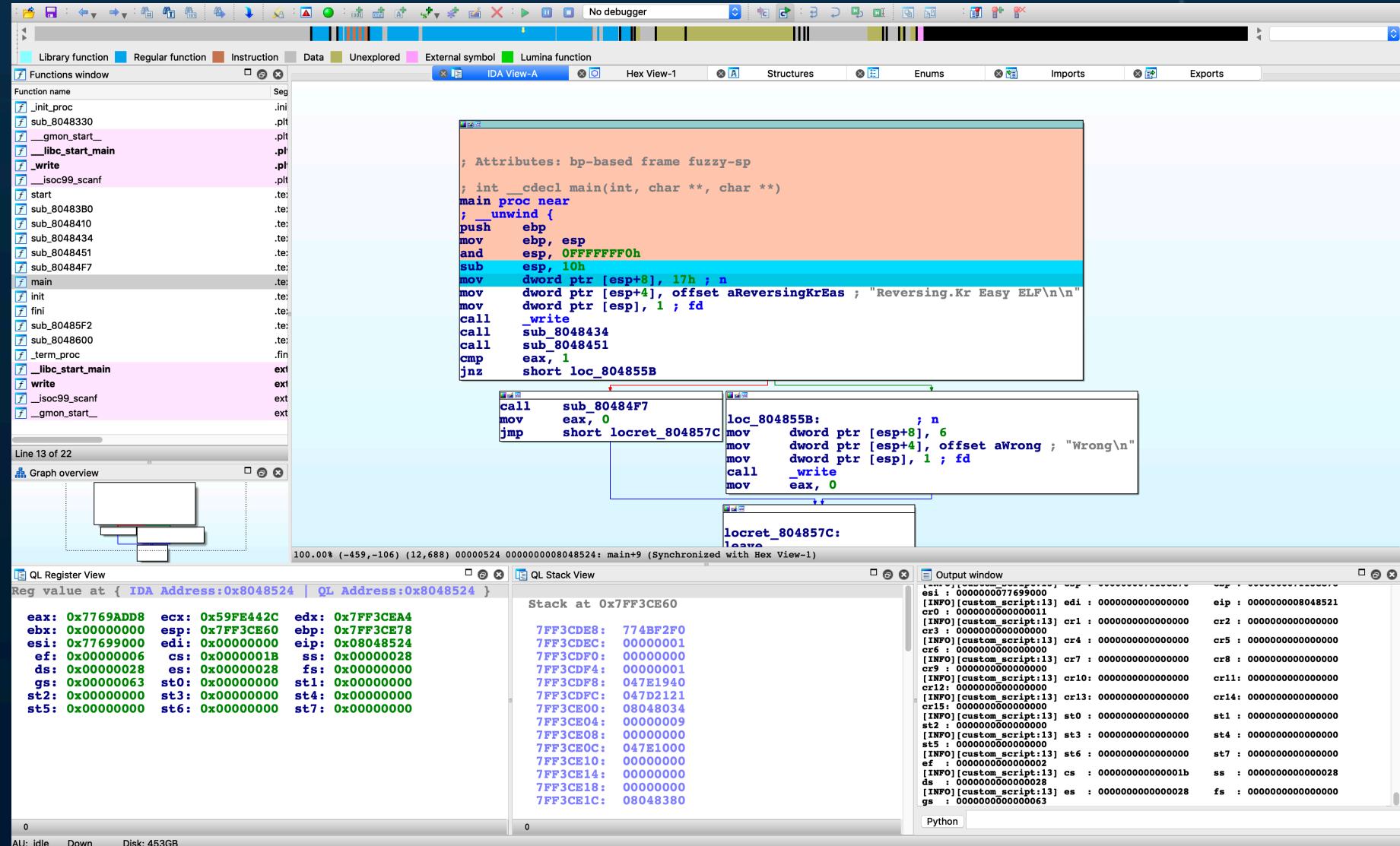
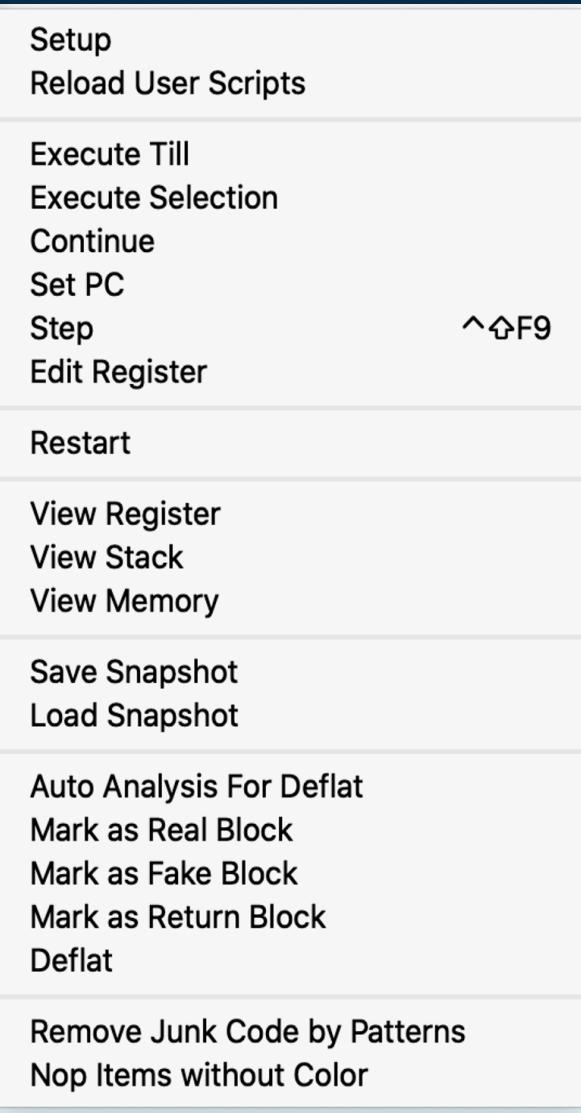
Output window

```

Loading type libraries...
Autoanalysis subsystem has been initialized.

```

# Qiling IDA Plugin



# 程序执行路径绘制

```
.text:0804851B
.text:0804851B
.text:0804851B ; Attributes: bp-based frame fuzzy-sp
.text:0804851B ; int __cdecl main(int, char **, char **)
.text:0804851B main proc near
.text:0804851B ; _ unwind {
.text:0804851B push    ebp
.text:0804851C mov     ebp, esp
.text:0804851E and    esp, 0FFFFFFF0h
.text:08048521 sub    esp, 10h
.text:08048524 mov    dword ptr [esp+8], 17h ; n
.text:0804852C mov    dword ptr [esp+4], offset aReversingKrEas ; "Reversing.Kr Easy ELF\n\n"
.text:08048534 mov    dword ptr [esp], 1 ; fd
.text:0804853B call   _write
.text:08048540 call   sub_8048434
.text:08048545 call   sub_8048451
.text:0804854A cmp    eax, 1
.text:0804854D jnz    short loc_804855B

.text:0804854F call   sub_80484F7
.text:08048554 mov    eax, 0
.text:08048559 jmp    short locret_804857C

.text:0804855B loc_804855B:          ; n
.text:0804855B mov    dword ptr [esp+8], 6
.text:08048563 mov    dword ptr [esp+4], offset aWrong ; "Wrong\n"
.text:0804856B mov    dword ptr [esp], 1 ; fd
.text:08048572 call   _write
.text:08048577 mov    eax, 0

.text:0804857C locret_804857C:
.text:0804857C leave
.text:0804857D retn
.text:0804857D ; } // starts at 804851B
.text:0804857D main endp
.text:0804857D
```

# Qiling IDA插件自定义脚本

使用 Qiling 和 IDAPython 提供的所有API

Execute Till

Execute Selection

Continue

Set PC

Step

Edit Register

↑↓F9

```
from qiling import *
import logging

class QILING_IDA():
    def __init__(self):
        pass

    def custom_prepare(self, ql:Qiling):
        logging.info('Context before starting emulation:')
        self._show_context(ql)

    def custom_continue(self, ql:Qiling):
        logging.info('custom_continue hook.')
        self._show_context(ql)
        hook = []
        return hook

    def custom_step(self, ql:Qiling):
        def step_hook(ql, addr, size):
            logging.info(f"Executing: {hex(addr)}")
            self._show_context(ql)

        logging.info('custom_step hook')
        hook = []
        hook.append(ql.hook_code(step_hook))
        return hook

    def custom_execute_selection(self, ql:Qiling):
        logging.info('custom execute selection hook')
        hook = []
        return hook
```

# Qiling IDA插件自定义脚本

```
def show_func_size(ql:Qiling):
    func_size_bytes = bytes(ql.mem.read(ql.reg.read('RAX')+0x8, 0x4))
    func_size_str = hex(struct.unpack('<I', func_size_bytes)[0])
    print('Function size is -> '+func_size_str)
    global func_size
    func_size = int(func_size_str, 16)

def patch_to_ida(ql:Qiling):
    global func_size
    base_addr = ql.reg.read('RCX')
    print('Encode function from -> '+hex(base_addr)+ ' to -> '+hex(base_addr+func_size))
    enc_func_bytes_list = list(bytes(ql.mem.read(ql.reg.read('RCX'), func_size)))

    for i in range(func_size):
        patch_byte(base_addr+i, enc_func_bytes_list[i])

def force_jmp(ql:Qiling):
    ql.reg.write('RAX', 0x1)

def rewrite_mem(ql:Qiling):
    ql.mem.write(ql.reg.read('RAX'), b'a'*128)

def show_enc_func(ql:Qiling):
    global func_size, func_size_list
    base_addr = ql.reg.read('RCX')
    create_data(base_addr, byte_flag(), func_size, BADNODE)
    is_create_func = add_func(base_addr, base_addr+func_size)

    if not is_create_func:
        for i in range(func_size):
            del_items(base_addr+i)
        create_insn(base_addr)
        add_func(base_addr, base_addr+func_size)

    if 'Verify_' not in get_func_name(base_addr) and func_size not in set(func_size_list):
        set_name(base_addr, 'Verify_'+hex(base_addr), SN_CHECK)
        func_size_list.append(func_size)

ql.hook_address(show_func_size, 0x402EA6)
ql.hook_address(patch_to_ida, 0x402F03)
ql.hook_address(force_jmp, 0x402F08)
ql.hook_address(rewrite_mem, 0x403B3E)

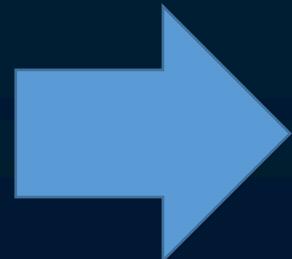
custom_continue(self, ql:Qiling):
    hook = []
    hook.append(ql.hook_address(show_enc_func, 0x402F06))
```

# 反 LLVM 平坦化

## 反OLLVM平坦化：介绍

- 取出所有的基本块
- 用一个大 while 和 switch 来控制基本块之间的跳转
  - 例子中引入了变量 b
- 随着逻辑变复杂，case快速增长，可以很好的迷惑逆向工程师。

```
int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    if(a == 0)
        return 1;
    else
        return 10;
    return 0;
}
```



```
int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    int b = 0;
    while(1) {
        switch(b) {
            case 0:
                if(a == 0)
                    b = 1;
                else
                    b = 2;
                break;
            case 1:
                return 1;
            case 2:
                return 10;
            default:
                break;
        }
    }
    return 0;
}
```

# 反OLLVM平坦化：样本

```

int func3(int v){
    return v+v/2+v*v;
}

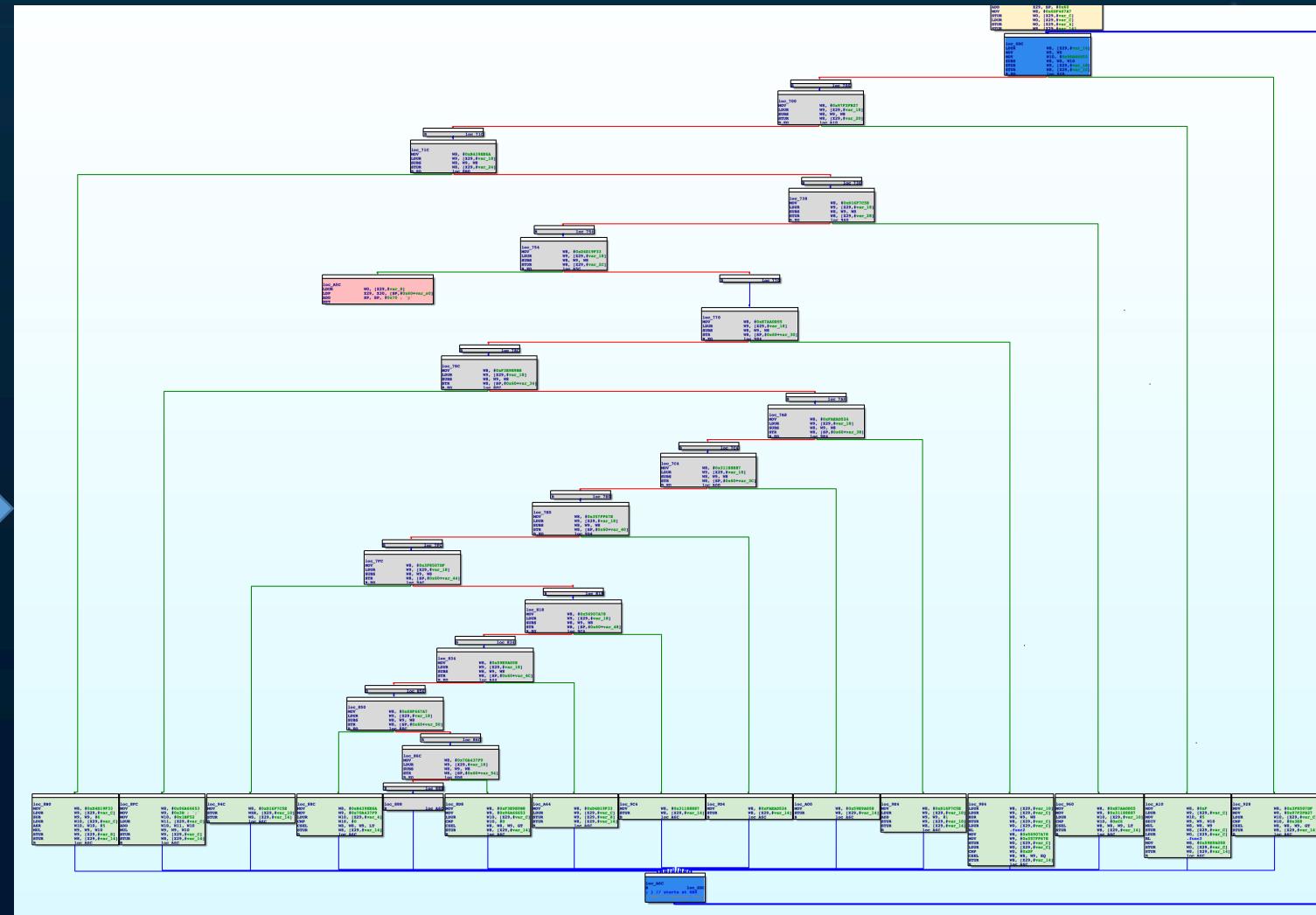
int func2(int v){
    return v*v*4;
}

int func(int v){
    if(v<0)
        return (v - 6)*(v>>5);
    if(v>0)
        v = 59 * (v + 114514);
    if(v > 1000){
        for(int i =0;i<198;i++){
            v ^= i;
            v = func2(v);
            if(v == 223)
                break;
        }
    }else{
        v = 15 * (v / 5);
        v = func3(v);
    }
    return v;
}

int main(){
    func(1);
}

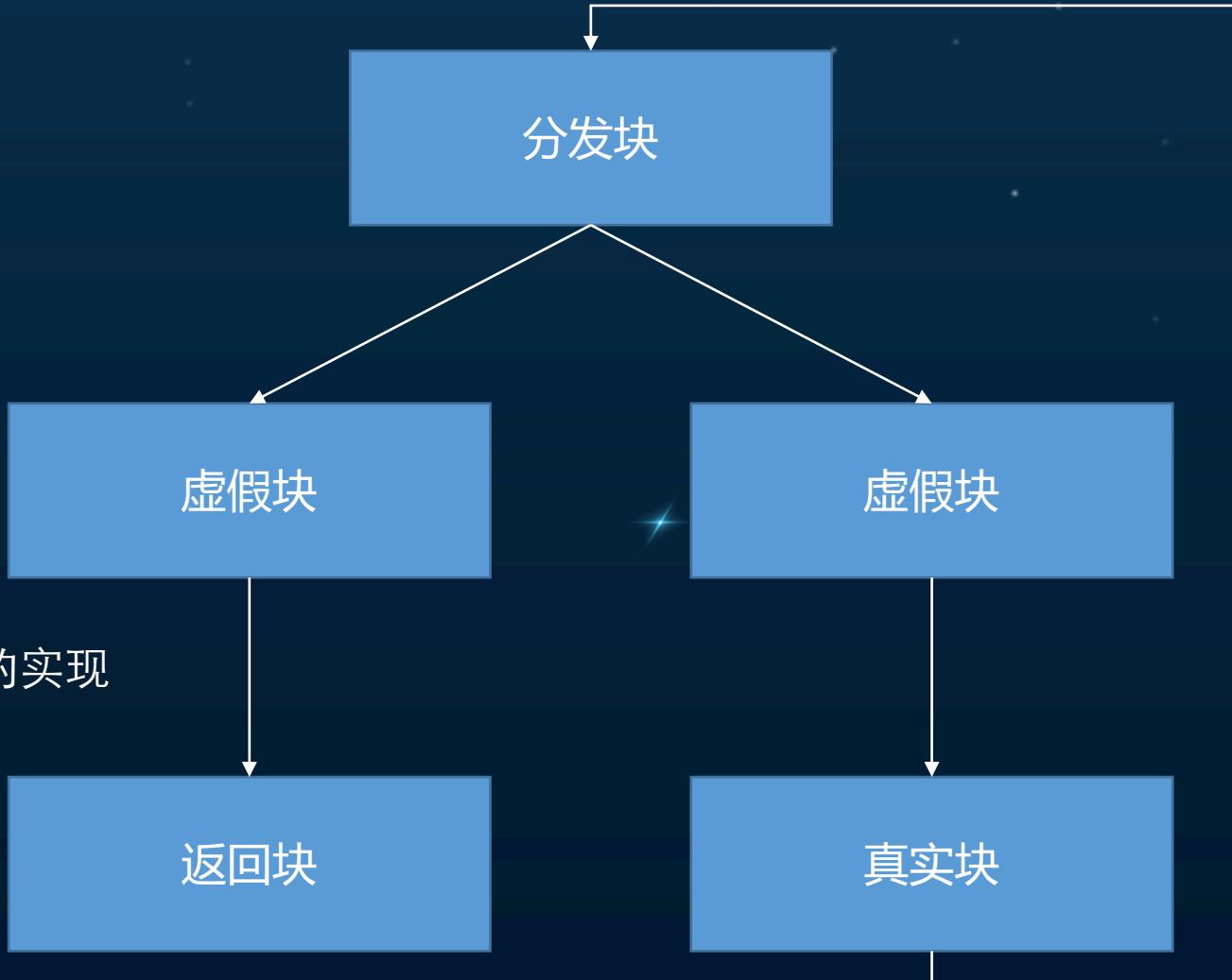
```

NDK配合  
ollvm编译



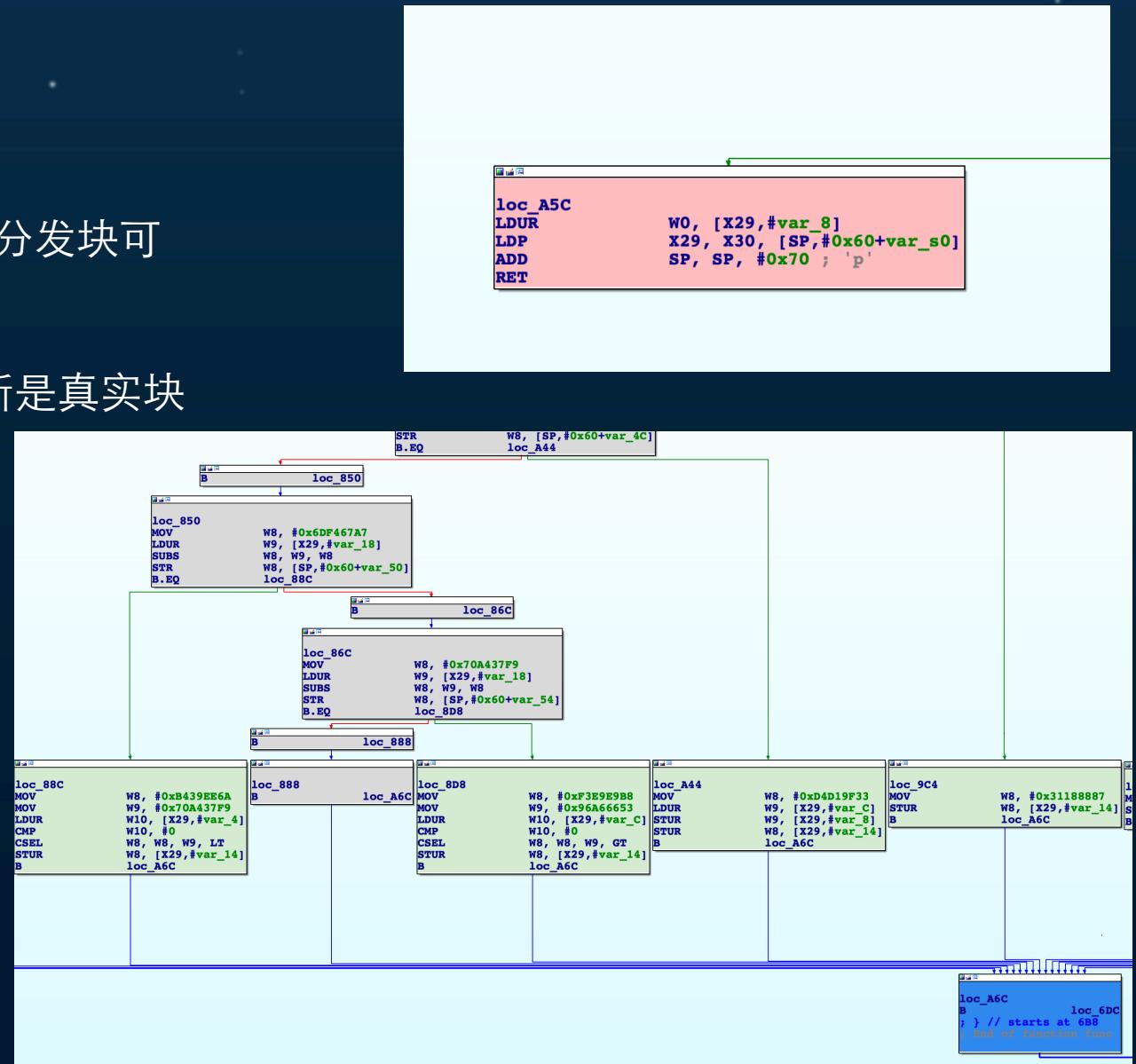
## 反OLLVM平坦化：识别

- 变形后的基本块可以分为四种
- 真实块
  - 包含了混淆前程序真实的逻辑
  - 可能包含一次条件语句
- 虚假块
  - 没有任何原逻辑，仅负责 switch case 的实现
- 返回块
  - 结束函数的执行
- 分发块
  - 相当于 switch 语句，决定控制流走向
  - 反混淆的核心是识别分发块



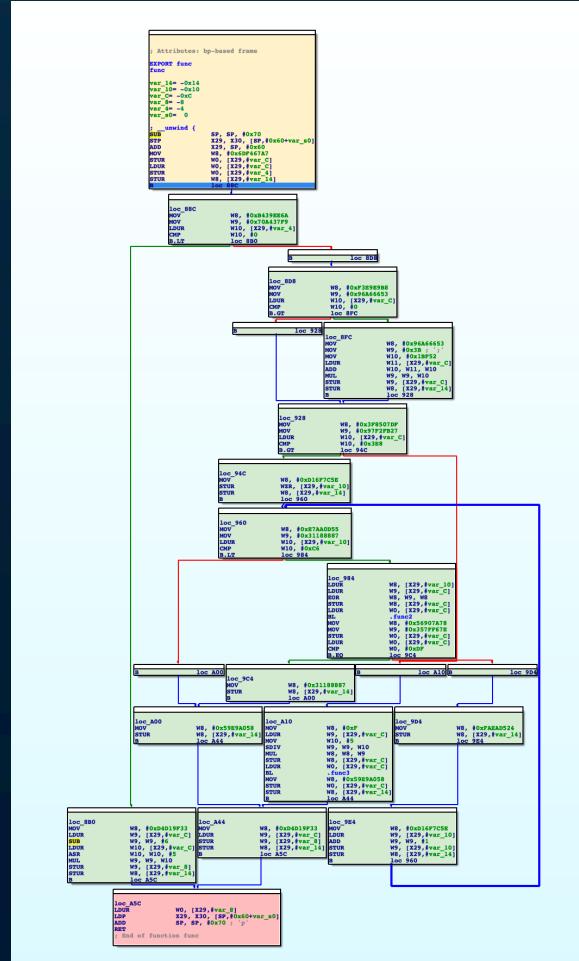
## 反OLLVM平坦化：识别

- 由于所有的真实块后继都是分发块，因此分发块可以简单通过计算引用次数查找
- 后继是真实块而且指令大于1条的可以判断是真实块
- 包含了返回语句比如 ret 的是返回块
- 其余均为虚假块
- 在 IDA 中四种代码块用不同的颜色表示
  - 方便用户手动调整自动分析的结果



# 反OLLVM平坦化：模拟执行

- 识别出各个块后，需要搜索真实控制流
- 利用麒麟的部分执行和快照来快速搜索
  - 保存快照然后从真实块的起始开始执行
  - 遇到分支则分别各执行一次
  - 遇到返回块或者另一个真实块停止
  - 记录控制流然后恢复快照
  - 此外需要 Hook 内存读写来跳过对堆的读写
- 通过 Keystone 来 patch
  - 最后程序流图里应该只剩下真实块和返回块



```

int64 __fastcall func(int a1)
{
    int i; // [xsp+50h] [xbp-10h]
    int v3; // [xsp+54h] [xbp-Ch]
    unsigned int v4; // [xsp+58h] [xbp-8h]

    v3 = a1;
    if ( a1 >= 0 )
    {
        if ( a1 > 0 )
            v3 = 59 * (a1 + 114514);
        if ( v3 > 1000 )
        {
            for ( i = 0; i < 198; ++i )
            {
                v3 = func2(v3 ^ (unsigned int)i);
                if ( v3 == 223 )
                    break;
            }
        }
        else
        {
            v3 = func3((unsigned int)(15 * (v3 / 5)));
        }
        v4 = v3;
    }
    else
    {
        v4 = (a1 - 6) * (a1 >> 5);
    }
    return v4;
}
  
```

# 反OLLVM平坦化：IR

- 一些细节和难点
  - 真实块内有外部函数调用
  - 不同架构的条件指令有区别

## - 解决办法：IR(Intermediate Representation)

- 实现中使用了 IDA 7.1 引入的微码
- 通过微码可以直接识别函数调用然后设置PC跳过
- 此外通过匹配微码的特征来避免硬编码识别不同架构的条件指令

```
.text:000000000000088C loc_88C
.text:000000000000088C MOV     W8, #0xB439EE6A ; CODE XREF: func+1AC↑j
.text:0000000000000894 MOV     W9, #0x70A437F9
.text:000000000000089C LDUR   W10, [X29,#var_4]
.text:00000000000008A0 CMP    W10, #0
.text:00000000000008A4 CSEL   W8, W8, W9, LT |
.text:00000000000008A8 STUR   W8, [X29,#var_14]
.text:00000000000008AC B      loc_A6C
```



```
=====0x86c===== id=16 0x8a4 #0.4
0x8a4: jge    %var_4.4 {1} ,
>>>>>>>>>>>>0x884<<<<<<<<<<<
=====0x8a4===== id=17 0x8a4
0x8a4: mov    #0xB439EE6A.4 ,
0x8a4: goto   @19
>>>>>>>>>>>>>0x8a8<<<<<<<<<<
=====0x8a4===== id=18 0x8a8
0x8a4: mov    #0x70A437F9.4 , w8.4
```

## 展望

- 底层模拟同步Qemu5支持的架构和指令集
- Android Java层模拟和插桩
- IOS模拟支持
- 完善Windows支持
  - 引入wine&&cygwin来进行完整支持
- 智能合约模拟
- 单片机模拟
- arm mac

## Q & A Time

项目地址: <https://github.com/qilingframework/qiling>

Thanks