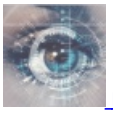
QilingLab练习

原创

CodeStarr 已于 2022-04-29 20:33:36 修改 1383 收藏

分类专栏： bin # re 文章标签： 安全

于 2022-04-25 19:51:47 首次发布



文章目录

Github：

- qilingframework/qiling: Qiling Advanced Binary Emulation Framework (github.com)
- 子项目：qilingframework/rootfs (github.com)

文档：Qiling Framework Documentation

官网：Qiling Framework

1. 简介

Qiling is an advanced binary emulation framework. By JingDong @Defcon 2019

内置examples可以用来学习。

安装

Python环境至少需要3.8（不然会报各种错误）。

```
sudo apt-get install python3.8-dev
```

最简单的pip安装：

```
python3 -m pip install qiling
```

Pip没有安装examples目录，而且无法调试，所以入门且想要调试的话，建议从github把两个仓库下载下来手动安装：

```
python3 setup.py install
```

根据.gitmodules文件，有子项目依赖，所以建议用用git --recursiv：

```
git clone https://github.com/qilingframework/qiling --recursiv
python3 setup.py install
```

如果遇到网速问题，可以手动下载后放进examples。

如果要模拟windows，需要执行dllscollector.bat收集系统dll放入rootfs。

2. QuickStart

Demo - Qiling Framework Documentation

Getting Started - Qiling Framework Documentation

Demo示例大多是windows程序，所以别忘执行dllscollector.bat。有的示例路径不对，需要加个examples啥的~

大概步骤：

1. ql = Qiling()初始化，目标可以是二进制文件或shellcode；
2. 设置一些参数，如map, debug, verbose;
3. ql.run()， 开始模拟，可以指定代码begin和end。

```
$ python3 test.py
[+]     Profile: Default
[+]     Map GDT at 0x30000 with GDT_LIMIT=4096
[+]     Write to 0x30018 for new entry b'\x00\xf0\x00\x00\x00\xfeO\x00'
[+]     Write to 0x30028 for new entry b'\x00\xf0\x00\x00\x00\x96O\x00'
[+]     Write to 0x30070 for new entry b'\x00`\x00`\x00\xf6@\x00'
[+]     Write to 0x30078 for new entry b'\x00\x00\x00\x00\x00\xf6@\x06'
[+]     Windows Registry PATH: examples/rootfs/x86_windows/Windows/registry
[=]     Initiate stack address at 0xfffdd000
[=]     Loading examples/rootfs/x86_windows/bin/RegDemo.exe to 0x400000
[=]     PE entry point at 0x401381
[=]     TEB addr is 0x6000
[=]     PEB addr is 0x6044
[=]     Loading examples/rootfs/x86_windows/Windows/System32/ntdll.dll ...
[!]     Warnings while loading examples/rootfs/x86_windows/Windows/System32/ntdll.dll:
```

```
[!]      - SizeOfHeaders is smaller than AddressOfEntryPoint: this file cannot run under Windows 8.
[!]      - AddressOfEntryPoint lies outside the sections' boundaries. AddressOfEntryPoint: 0x0
[+]     DLL preferred base address: 0x4b280000
[=]     Done with loading examples/rootfs/x86_windows/Windows/System32/ntdll.dll
[=]     Loading examples/rootfs/x86_windows/Windows/System32/kernel32.dll ...
[+]     DLL preferred base address: 0x6b800000
[=]     Done with loading examples/rootfs/x86_windows/Windows/System32/kernel32.dll
[=]     Loading examples/rootfs/x86_windows/Windows/System32/advapi32.dll ...
[+]     DLL preferred base address: 0x4c300000
[=]     Done with loading examples/rootfs/x86_windows/Windows/System32/advapi32.dll
[=]     Loading examples/rootfs/x86_windows/Windows/System32/msvcr110.dll ...
[+]     DLL preferred base address: 0x10000000
[=]     Done with loading examples/rootfs/x86_windows/Windows/System32/msvcr110.dll
[+]     Done with loading examples/rootfs/x86_windows/bin/RegDemo.exe
[+]     0x6b81f390: GetSystemTimeAsFileTime(lpSystemTimeAsFileTime = 0xffffcfec)
[+]     0x6b81df10: GetCurrentThreadId() = 0x0
[+]     0x6b822e90: GetCurrentProcessId() = 0x7cc
[+]     0x6b81df40: QueryPerformanceCounter(lpPerformanceCount = 0xffffcfe4) = 0x0
[+]     0x10053f90: _initterm_e(pfbegin = 0x4020b8, pfend = 0x4020c8) = 0x0
[+]     0x10053f30: _initterm(pfbegin = 0x4020ac, pfend = 0x4020b4)
[+]     Key 2147483649 Software
[+]     Value key HKEY_CURRENT_USER\Software not present
[+]     0x4c31ed30: RegOpenKeyW(hKey = 0x80000001, lpSubKey = "Software", phkResult = 0xffffcfb4) = 0x2
Create Key Error[+]     0x10062380: printf(format = "Create Key Error") = 0x10
[+]     0x10054160: exit(status = 0)
[+]     Syscalls called:
[+]     GetSystemTimeAsFileTime:
[+]       {"params": {"lpSystemTimeAsFileTime": 4294954988}, "retval": null, "address": 1803678608, "retaddr": 4
200091, "position": 0}
[+]     GetCurrentThreadId:
[+]       {"params": {}, "retval": 0, "address": 1803673360, "retaddr": 4200106, "position": 1}
[+]     GetCurrentProcessId:
[+]       {"params": {}, "retval": 1996, "address": 1803693712, "retaddr": 4200115, "position": 2}
[+]     QueryPerformanceCounter:
[+]       {"params": {"lpPerformanceCount": 4294954980}, "retval": 0, "address": 1803673408, "retaddr": 4200128,
 "position": 3}
[+]     _initterm_e:
[+]       {"params": {"pfbegin": 4202680, "pfend": 4202696}, "retval": 0, "address": 268779408, "retaddr": 41990
46, "position": 4}
[+]     _initterm:
[+]       {"params": {"pfbegin": 4202668, "pfend": 4202676}, "retval": null, "address": 268779312, "retaddr": 41
99098, "position": 5}
[+]     RegOpenKeyW:
[+]       {"params": {"hKey": 2147483649, "lpSubKey": "Software", "phkResult": 4294954932}, "retval": 2, "addres
s": 1278340400, "retaddr": 4198443, "position": 6}
[+]     printf:
[+]       {"params": {"format": "Create Key Error"}, "retval": 16, "address": 268837760, "retaddr": 4198458, "po
sition": 7}
[+]     exit:
[+]       {"params": {"status": 0}, "retval": null, "address": 268779872, "retaddr": 4199217, "position": 8}
[+]     Registries accessed:
[+]     HKEY_CURRENT_USER\Software:
[+]     Strings:
[+]     Software: 6
[+]     Create: 7
[+]     Key: 7
[+]     Error: 7
```

有一个wannaycry的zip，解压口令在readme里，注意别在windows实体机上玩~

输出日志：`ql.log.info`，而且支持正则过滤。

3. qltool

项目根目录提供了这个qltool python脚本，用来快速模拟运行shellcode或exec文件。

基本参数：

- run, 执行exec文件；
- code，执行shellcode；
- examples，显示demo。

执行时发现qdb.py报错，使用了py3.8的赋值运算符，可以用with语句改一下，比如：

```
# if (bp := self.bp_list.get(address, None)):
with self.bp_list.get(address, None) as bp:
    # pass
```

不过这个`:=`运算符用得太多了，，最后我选择把python从3.6升级到3.8，并修改qltool开头的解释器：

```
#!/usr/bin/python3.8
```

依赖库：

```
sudo apt-get install python3.8-dev
sudo python3.8 -m pip install gevent
```

具体用法执行`./qltools examples`参考即可。

4. QilingLab练习

地址：Shielder - QilingLab – Release

有x86_64和aarch64两个版本，我选择了aarch64：

```
$ file qilinglab-aarch64
qilinglab-aarch64: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter
/lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0, BuildID[sha1]=c573ba5f5450dc8d134835d4ddb5f93e28138c0c, not str
ipped
```

注意LSB，说明是小端。

如果选择x86_64版本，rootfs就要使用`examples/rootfs/x8664_linux`

解题模板：

```
from qiling import *
from qiling.const import QL_VERBOSE

def challenge1(ql:Qiling):
    pass;

if __name__ == "__main__":
    argv = ["./qilinglab-aarch64"]
    rootfs = "examples/rootfs/arm64_linux"
    ql = Qiling(argv, rootfs, verbose=QL_VERBOSE.DEBUG, bigendian=False)
    # ql.debugger = "gdb:0.0.0.0:9999"
    # ql.debugger = "idapro:0.0.0.0:9999"
 challenge1(ql);
    ql.run();
```

直接运行的话会报一堆错，都是正常的。如果卡住，可以强杀：

```
ps -elf | grep "python3.8" | grep -v "grep" | awk -F" " '{print $4}' | xargs sudo kill -9
```

如果启用调试器，程序应该会停在入口点。Gdb可以用gdb-multiarch：

```
set architecture aarch64
target remote localhost:9999
```

如果python小于3.8，则不支持调试。

试了下Ida调试，没成功，，直接导入idapro模块失败了，没查出原因。IDA连接gdbserver模块是没问题的，但单步一下就跑飞了，，，不太好用。

```
$ python3.8 qilinglab.py
Welcome to QilingLab.
Here is the list of challenges:
Challenge 1: Store 1337 at pointer 0x1337.
Challenge 2: Make the 'uname' syscall return the correct values.
Challenge 3: Make '/dev/urandom' and 'getrandom' "collide".
Challenge 4: Enter inside the "forbidden" loop.
Challenge 5: Guess every call to rand().
Challenge 6: Avoid the infinite loop.
Challenge 7: Don't waste time waiting for 'sleep'.
Challenge 8: Unpack the struct and write at the target address.
Challenge 9: Fix some string operation to make the iMpOsSiBlE come true.
Challenge 10: Fake the 'cmdline' line file to return the right content.
Challenge 11: Bypass CPUID/MIDR_EL1 checks.

Checking which challenge are solved...
Note: Some challenges will results in segfaults and infinite loops if they aren't solved.
[x]     CPU Context:
[x]     x0      : 0x0
[x]     x1      : 0x0
[x]     x2      : 0x1
[x]     x3      : 0x0
[x]     x4      : 0x0
[x]     x5      : 0x55555556a2b4
[x]     x6      : 0x696b636568430a0a
[x]     x7      : 0x686369687720676e
[x]     x8      : 0x40
[x]     x9      : 0x7320657261206567
[x]     x10     : 0x2068636968772067
...
```

简单分析一下程序，一共11关，start函数开头会对一个12字节的数组初始化为0：

```
.text:00000000000014BC loc_14BC                                ; CODE XREF: start+54↓j
.text:00000000000014BC                 LDRSW           X0, [SP,#0x40+var_24_i]
.text:00000000000014C0                 ADD             X1, SP, #0x40+var_18_arrayBytesFlags[0xB]
.text:00000000000014C4                 STRB            WZR, [X1,X0]
.text:00000000000014C8                 LDR             W0, [SP,#0x40+var_24_i]
.text:00000000000014CC                 ADD             W0, W0, #1
.text:00000000000014D0                 STR             W0, [SP,#0x40+var_24_i]
.text:00000000000014D4
.text:00000000000014D4 loc_14D4                                ; CODE XREF: start+2C↑j
.text:00000000000014D4                 LDR             W1, [SP,#0x40+var_24_i]
.text:00000000000014D8                 LDR             W0, [SP,#0x40+var_1C]
.text:00000000000014DC                 CMP             W1, W0
.text:00000000000014E0                 B.LT            loc_14BC ; i <= 0xB
```

每一个字节代表这一关是否通过，通过则为1。

每一关的函数声明大概像这样：

```
void challenge1(byte* pByFlag);
```

之后，由checker函数检查flag：

```
void checker(byte* pByFlags, int nChallenge);
```

challenge 1 - memory map

```
.text:0000000000000CC4                 SUB             SP, SP, #0x20
.text:0000000000000CC8                 STR             X0, [SP,#0x20+var_18]
.text:0000000000000CCC                 MOV             X0, #0x1337
.text:0000000000000CD0                 STR             X0, [SP,#0x20+var_8__0x1337]
.text:0000000000000CD4                 LDR             X0, [SP,#0x20+var_8__0x1337]
.text:0000000000000CD8                 LDR             W0, [X0]
.text:0000000000000CDC                 STR             W0, [SP,#0x20+var_C__mem[0x1337]]
.text:0000000000000CE0                 LDR             W0, [SP,#0x20+var_C__mem[0x1337]]
.text:0000000000000CE4                 CMP             W0, #1337
.text:0000000000000CE8                 B.NE            loc_CF8
.text:0000000000000CEC                 LDR             X0, [SP,#0x20+var_18]
.text:0000000000000CF0                 MOV             W1, #1
.text:0000000000000CF4                 STRB            W1, [X0]
.text:0000000000000CF8
.text:0000000000000CF8 loc_CF8                                 ; CODE XREF: challenge1+24↑j
.text:0000000000000CF8                 NOP
.text:0000000000000CFC                 ADD             SP, SP, #0x20 ; ' '
.text:0000000000000D00                 RET
```

翻译成c语言：

```
if(mem[0x1337] == 1337)
 *pByFlag = 1;
```

解

参考文档：Memory - Qiling Framework Documentation

```
def challenge1(ql:Qiling):
    ql.mem.map(0x1337//4096*4096, 4096, info="[challenge1]");   # 4k alignment
    ql.mem.write(0x1337, ql.pack32(1337))
```

关于map的更多使用，可以参考`qiling/loader/elf.py`。

challenge 2 - set_syscall

第二关调用uname，硬编码检查sysname和version:

```
.text:0000000000000D28                 ADD             X0, SP, #0x1F0+name ; name
.text:0000000000000D2C                 BL              .uname
...
.text:0000000000000D48                 ADRP            X0, #aQilingos@PAGE ; "QilingOS"
.text:0000000000000D4C                 ADD             X1, X0, #aQilingos@PAGEOFF ; "QilingOS"
.text:0000000000000D50                 ADD             X0, SP, #0x1F0+buf
.text:0000000000000D54                 LDR             X2, [X1] ; "QilingOS"
.text:0000000000000D58                 STR             X2, [X0]
.text:0000000000000D5C                 LDRH            W1, [X1,#(aQilingos+8 - 0x1840)] ; ""
.text:0000000000000D60                 STRH            W1, [X0,#8]
.text:0000000000000D64                 ADRL            X0, aChallengestart ; "ChallengeStart"
...
```

通关条件：

```
uname.sysname == "QilingOS";
uname.version == "ChallengeStart";
```

通过qiling提供的系统调用hook，修改返回地uname结构体即可。

解

参考文档：

uname(3p) - Linux manual page (man7.org)

Hijack - Qiling Framework Documentation

uname结构体定义如下：

```
/* Structure describing the system and machine.  */
struct utsname
{
    /* Name of the implementation of the operating system.  */
    char sysname[65];

    /* Name of this node on the network.  */
    char nodename[65];

    /* Current release level of this implementation.  */
    char release[65];
    /* Current version level of this release.  */
    char version[65];

    //...
};
```

Qiling脚本如下：

```
#########################
# https://man7.org/linux/man-pages/man3/uname.3p.html
# int uname(struct utsname *name);
# /* Structure describing the system and machine.  */
# struct utsname
# {
#     /* Name of the implementation of the operating system.  */
#     char sysname[65];

#     /* Name of this node on the network.  */
#     char nodename[65];

#     /* Current release level of this implementation.  */
#     char release[65];
#     /* Current version level of this release.  */
#     char version[65];

#     //...
# };
def my_uname(ql:Qiling, pStcUname, *args,**kwargs):
    ql.mem.write(pStcUname, b"QilingOS".ljust(65,b"\x00"))  # "QilingOS\0\0\0\0..."

    ql.mem.write(pStcUname + 65*3, b'ChallengeStart'.ljust(65, b'\x00'))

    return 0

def challenge2(ql:Qiling):
    ql.set_syscall("uname", my_uname )
```

如果是其它qiling没有实现的syscall，就需要指定syscall number，比如write为4.

challenge 3 - add_fs_mapper

伪代码：

```
fd = open("/dev/urandom");
read(fd, urandom_buf, 0x20);
read(fd, &byUrandom, 0x1);
getrandom(getrandom_buf, 0x20, 1);

int nTmp = 0;
for(int i = 0; i <= 0x1f; ++i)
{
    if( (urandom_buf[i] == getrandom_buf[i]) && (byUrandom != urandom_buf[i]))
        ++nTmp;
}
if(nTmp == 0x20)
    *pFlag = 1;
```

解

参考文档：

- getrandom(2) - Linux manual page (man7.org)
- Hijack - Qiling Framework Documentation

getrandom是一个系统调用，其实默认也是从`/dev/urandom`取值，flag设为1可以防止熵池为空导致程序阻塞。

```
/* Flags for use with getrandom.  */
#define GRND_NONBLOCK 0x01
#define GRND_RANDOM 0x02
ssize_t getrandom(void *buf, size_t buflen, unsigned int flags);
```

也就是说，程序涉及getrandom这个系统调用，以及`/dev/urandom`这个设备。除了利用上一关的set_syscall劫持getrandom，还需要用qiling的QlFsMappedObject类劫持urandom设备的读操作。

直接照抄文档的示例即可：

```
from qiling.os.mapper import QlFsMappedObject


def my_syscall_getrandom(ql:Qiling, write_buf, write_buf_size, flags, *args, **kw):
    ql.mem.write(write_buf, b"\x01" * write_buf_size)
    return write_buf_size;

class Fake_urandom(QlFsMappedObject):

    def read(self, size):
        if(size == 1):
            return b"\x02"  # byUrandom
        else:
            return b"\x01" * size

    def fstat(self): # syscall fstat will ignore it if return -1
        return -1

    def close(self):
        return 0

def challenge3(ql:Qiling):
    ql.add_fs_mapper("/dev/urandom", Fake_urandom())
    ql.set_syscall("getrandom", my_syscall_getrandom )
```

challenge 4 - hook_address

伪代码：

```
int var_4__l = 0;
int var_8__r = 0;
while(var_4__l < var_8__r)
{
    *pFlag = 1;
    ++var_4__l;
}
// .text:0000000000000FD8                LDR             W0, [SP,#0x20+var_8__r]
// .text:0000000000000FDC                LDR             W1, [SP,#0x20+var_4__l]
// .text:0000000000000FE0                CMP             W1, W0
```

那么在执行cmp时，让w0为1即可。

解

参考文档：

- Hook - Qiling Framework Documentation
- Register - Qiling Framework Documentation

Qiling提供了一个获取模块基址的函数`get_lib_base`，不过在文档里没找到。

```
import os
def hook_cmp_w0(ql:Qiling):
    ql.reg.write("w0", 0x1)
    return;

def challenge4(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.hook_address(hook_cmp_w0, pBase + 0xFE0)
```

challenge 5 - set_api

伪代码：

```
DWORD dwArr1[5] = {0};
DWORD dwArr2[5] = {};
// init dwArr2 by rand()
// .text:0000000000001038                BL              .rand
// .text:000000000000103C                MOV             W2, W0
for (int i = 0; i <= 4; ++i)
{
    if(dwArr1[i] != dwArr2[i])
    {
  *pFlag = dwArr2[i];
        return;
    }
 *pFlag = 1;
}
```

也就是说，劫持rand返回0就可以了。

解

参考文档：

- rand(3) - Linux manual page (man7.org)
- Hijack - Qiling Framework Documentation

rand()是库函数，不是系统调用，所以不能用set_syscall，应该用set_api。

```
def my_rand(ql:Qiling):
    # ql.reg.write("w0", 0)
    ql.reg.write("eax", 1)
    return;

def challenge5(ql:Qiling):
    ql.set_api("rand", my_rand)
```

不过从第一题开始，我的程序就会在第5关这里崩溃，换成x8664不用qiling直接执行也一样，所以后续的题解验证需要自己逆向出源代码，安装aarch64编译工具来验证。

```
sudo apt install gccgo-5-aarch64-linux-gnu
aarch64-linux-gnu-gcc-5 test.c -o aarch64
```

逆向出来的源码：

```
#include <sys/types.h>
#include <stdio.h>
int main()
{
    int32_t dwArr1[5] = {0};
    int32_t dwArr2[5] = {};
    int nFlag = 1;
    int s = time(0);
    srand(s);
    for (int i = 0; i <= 4; ++i)
    {
        dwArr2[i] = rand();
        printf("%d\n", dwArr2[i]);
    }
    for (int i = 0; i <= 4; ++i)
    {
        if(dwArr1[i] != dwArr2[i])
        {
            nFlag = dwArr2[i];
            break;
        }

    }
    if(1 == nFlag)
    {
        printf("congratulation\n");
    }
    return 0;
}
```

rand返回值仍然保存在w0里，所以qiling脚本不用变。

challenge 6

伪代码：

```
int n1 = 1;
int n2;
while( (n1 & 0xff) != 0)
{
    n2 = (n1 & 0xff);
}
```

和第4关差不多

```
def hook_cmp_w0_6(ql:Qiling):
    ql.reg.write("w0", 0x0)
    return;
def challenge6(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.hook_address(hook_cmp_w0_6, pBase + 0xFE0)
```

challenge 7 - 3种思路

伪代码：

```
*pFlag = 1;
sleep(0xFFFFFFFF);
```

需要做的是劫持sleep函数，修改睡眠时间。

解1

逆向实现：

```
#include <unistd.h>

int main()
{
    sleep(0xFFFFFFFF);
    return 0;
}
// .text:00000000000007C0                MOV             W0, #0xFFFFFFFF ; seconds
// .text:00000000000007C4                BL              .sleep
```

Qiling脚本：

```
def my_sleep_call(ql:Qiling):
    ql.reg.write("w0", 0x0)
    return;

def challenge7(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.hook_address(my_sleep_call, pBase + 0x7C4)
```

只要程序马上退出，就说明成功了。

解2

也可以hook api，脚本实现：

```
def my_sleep(ql:Qiling):
    return;
def challenge7_hook_api(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.set_api("sleep", my_sleep)
```

解3

也可以劫持系统调用，根据DEBUG信息，sleep其实是调用了nanosleep():

```
[+]     syscall hooked 0x7fffb7e7862c: ql_syscall_nanosleep()
```

官网文档里也有说明：

```
https://man7.org/linux/man-pages/man3/sleep.3.html
https://man7.org/linux/man-pages/man2/nanosleep.2.html
 On Linux, sleep() is implemented via nanosleep(2).
```

Qiling脚本如下：

```
# int nanosleep(const struct timespec *req, struct timespec *rem);
def my_nanosleep(ql:Qiling, req, rem, *args,**kwargs):
    return;

def challenge7_hook_syscall(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.set_syscall("nanosleep", my_nanosleep)
```

challenge 8

调用了两次malloc，根据反汇编，结构如下：

```
struct Heap1 // 0x18 bytes
{
    struct *pHeap2; // 8 bytes
    int32_t n1 = 0x00000539; // 4 bytes
    int32_t n1 = 0x3DFCD6EA; // 4 bytes
    int *pFlag; // 8 bytes
}

struct Heap2 // 0x1e bytes
{
    char buf[] = {"Random data"}
}
```

创建IDA结构体：

```
// 00000000 Heap1           struc ; (sizeof=0x18, align=0x8, mappedto_40)
// 00000000 pHeap2          DCQ ?
// 00000008 MAGIC           DCQ ?
// 00000010 NUM             DCQ ?
// 00000018 Heap1           ends
// 00000018
// 00000000 ; ---------------------------------------------------------------------------
// 00000000
// 00000000 Heap2           struc ; (sizeof=0x20, align=0x8, mappedto_42)
// 00000000 buf             DCB 30 dup(?)
// 0000001E                 DCB ? ; undefined
// 0000001F                 DCB ? ; undefined
// 00000020 Heap2           ends
// .text:00000000000011D0                 LDR             X0, [SP,#0x30+var_8__pHeap1]
// .text:00000000000011D4                 LDR             X1, [SP,#0x30+var_18___pFlag]
// .text:00000000000011D8                 STR             X1, [X0,#0x10]
// .text:00000000000011DC                 NOP
Heap1 *__fastcall challenge8(__int64 a1)
{
  Heap1 *result; // x0
  Heap1 *v3; // [xsp+28h] [xbp+28h]

  v3 = (Heap1 *)malloc(0x18uLL);
  v3->pHeap2 = (__int64)malloc(0x1EuLL);
  LODWORD(v3->MAGIC) = 1337;
  HIDWORD(v3->MAGIC) = 1039980266;
  strcpy((char *)v3->pHeap2, "Random data");
  result = v3;
  v3->NUM = a1;
  return result;
}
```

逆向

```c
// aarch64-linux-gnu-gcc-5 test.c -g -o aarch64

#include <stdint.h>
#include <string.h>
struct Heap2 // 0x1e bytes
{
   char buf[0x1e];
};
struct Heap1 // 0x18 bytes
{
    struct Heap2* pHeap2; // 8 bytes
    int64_t nMagic;
    int *pFlag; // 8 bytes
};


int main()
{
    struct Heap1 heap1;
    struct Heap2 heap2;
    char arrBuf[] = "Random data";
    int nFlag = 0;
    heap1.pHeap2 = &heap2;
    heap1.nMagic = 0x3DFCD6EA00000539;
    heap1.pFlag = &nFlag;
    memcpy(heap2.buf, arrBuf, sizeof(arrBuf));
    printf("heap1.pHeap2: 0x%p\n", heap1.pHeap2);
    printf("heap1.nMagic: 0x%16x\n", heap1.nMagic);
    printf("heap1.pFlag: 0x%p\n", heap1.pFlag);
    __asm__("nop");
    if(1 == nFlag)
    {
        printf("congratulations\n");
    }
    return 0;
}
```

HOOK目标地址就是nop指令的地址，目的是改写heap1.pFlag。

解1-读栈

```
.text:0000000000000878 heap1           = -0x50
.text:00000000000008B8                 STR             X0, [X29,#0x70+heap1]
```

脚本（注意我自己写的代码其实没有调用malloc，结构体是直接存在栈上的）：

```
def read_stack_heap1(ql:Qiling):
    # pHeap1 = ql.mem.read(ql.reg.read("sp") + 0x20, ql.archbit // 8);
    # pHeap1 = ql.unpack64(pHeap1);
    # heap1 = ql.mem.read(pHeap1, 0x18)

    # out struct is in stack actually
    heap1 = ql.mem.read(ql.reg.read("sp") + 0x20, 0x18);

    pHeap2, nMagic, pFlag = struct.unpack("QQQ", heap1);
    # ql.log.info("pHeap1: %x" % pHeap1)
    ql.log.info("heap1.pHeap2: %x" % pHeap2)
    ql.log.info("heap1.nMagic: %x" % nMagic)
    ql.log.info("heap1.pFlag: %x" % pFlag)
    ql.mem.write(pFlag, b"\x01");


def challenge8_read_stack(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.hook_address(read_stack_heap1, pBase + 0x96C) # 0x96C : nop
```

解2-搜索内存

搜索内存里的magic，找到heap1, 修改`*pFlag`。

```
import struct
def search_heap1(ql:Qiling):
    nMagic = 0x3DFCD6EA00000539;
    pMagics = ql.mem.search(ql.pack64(nMagic))
    for pMagic in pMagics:
        pHeap1 = pMagic - ql.archbit//8;
        heap1 = ql.mem.read(pHeap1, 0x18);
        pHeap2, _, pFlag = struct.unpack("QQQ", heap1)
        if ql.mem.string(pHeap2) == "Random data":
            ql.mem.write(pFlag, b"\x01")
            break;
    return;
def challenge8_search_mem(ql:Qiling):
    pBase = ql.mem.get_lib_base(os.path.split(ql.path)[-1])
    ql.hook_address(search_heap1, pBase + 0x96C) # 0x96C : nop
```

challenge 9

伪代码：

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main()
{
    char src[0x20] = {0};
    char dst[0x20] = {0};
    int nFlag = 0;
    int i = 0;

    strcpy(src, "aBcdeFghiJKlMnopqRstuVWxYz");
    strcpy(dst, src);
    for(i = 0; i < 26; ++i)
    {
        dst[i] = tolower(dst[i]);
    }

    nFlag = (strcmp(src, dst) == 0) ? 1  : 0;

    if(1 == nFlag)
    {
        printf("congratulations\n");
    }
    return 0;
}
```

Hook掉tolower就行了：

```python
def my_tolower(ql: Qiling):
    return

def challenge9(ql: Qiling):
    ql.set_api('tolower', my_tolower)
```

challenge 10

伪代码：

```c
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main()
{
    int nFlag = 0;
    int fd = 0;
    char path[] = {"/proc/self/cmdline"};
    char buf[0x40] = {0};
    int nRet = 0;
    int i = 0;

    do{
        fd = open(path, O_RDONLY);
        if( -1 == fd) break;
        nRet = read(fd, buf, 0x3F);
        if (0 == nRet) break;

    }while(0);

    for(i = 0; i < nRet; ++i)
    {
        if(buf[i] == 0)
        {
            buf[i] = 0x20;
        }
    }

    if(strcmp(buf, "qilinglab") == 0 )
    {
        nFlag = 1;
    }

    if(1 == nFlag)
    {
        printf("congratulations\n");
    }
    return 0;
}
```

和第三关一样，劫持`/proc/self/cmdline`设备即可。

```python
class My_cmdline(QlFsMappedObject):
    def read(self, size):
        return b"qilinglab"
    def fstat(self):
        return -1
    def close(self):
        return 0

def challenge10(ql: Qiling):
    ql.add_fs_mapper("/proc/self/cmdline", My_cmdline())
```

challenge 11

aarch64

```
.text:00000000000013E4 FF 83 00 D1                SUB             SP, SP, #0x20
.text:00000000000013E8 E0 07 00 F9                STR             X0, [SP,#0x20+var_18__pFlag]
.text:00000000000013EC 00 00 38 D5                MRS             X0, #0, c0, c0, #0
.text:00000000000013F0 E0 0F 00 F9                STR             X0, [SP,#0x20+var_8]
.text:00000000000013F4 E0 0F 40 F9                LDR             X0, [SP,#0x20+var_8]
.text:00000000000013F8 01 FC 50 D3                LSR             X1, X0, #0x10
.text:00000000000013FC E0 66 82 D2                MOV             X0, #0x1337
.text:0000000000001400 3F 00 00 EB                CMP             X1, X0 ; if ((var_8 >> 0x10) == 0x1337) *pFla
g = 1
.text:0000000000001404 81 00 00 54                B.NE            loc_1414
.text:0000000000001408 E0 07 40 F9                LDR             X0, [SP,#0x20+var_18__pFlag]
.text:000000000000140C 21 00 80 52                MOV             W1, #1
.text:0000000000001410 01 00 00 39                STRB            W1, [X0]
.text:0000000000001414
.text:0000000000001414             loc_1414                               ; CODE XREF: challenge11+20↑j
.text:0000000000001414 1F 20 03 D5                NOP
.text:0000000000001418 FF 83 00 91                ADD             SP, SP, #0x20 ; ' '
.text:000000000000141C C0 03 5F D6                RET
```

看网上用ghidra逆出来好像效果好一些。

这个MRS指令比较复杂，大概就是用来收集CPU各种信息的，参考文档：Arm A64 Instruction Set Architecture

Arm的msr指令在Intel指令集里对应cpuid指令，

参考文档（很长）：CPUID — CPU Identification (felixcloutier.com)

本实验x8664版本指令：

```
...
.text:000000000000118A                mov     eax, 40000000h
.text:000000000000118F                cpuid
...
```

解

可以使用qiling提供的hook_code，直接把这个指令跳过，参考文档：Hook - Qiling Framework Documentation。

逆向一下：

```c
#include <unistd.h>
#include <stdio.h>
int main()
{
    int nFlag = 0;
    int nTmp = 0;
    __asm__("mrs x0, midr_el1");
    if( ( nTmp >> 0x10 ) == 0x1337)
    {
        nFlag = 1;
    }
    if(1 == nFlag)
    {
        printf("congratulations\n");
    }
    return 0;
}

// .text:00000000000007B8 FD 7B BE A9                STP             X29, X30, [SP,#var_20]!!
// .text:00000000000007BC FD 03 00 91                MOV             X29, SP
// .text:00000000000007C0 BF 1B 00 B9                STR             WZR, [X29,#0x20+nFlag]
// .text:00000000000007C4 BF 1F 00 B9                STR             WZR, [X29,#0x20+nTmp]
// .text:00000000000007C8 00 00 38 D5                MRS             X0, #0, c0, c0, #0
// .text:00000000000007CC A0 1F 40 B9                LDR             W0, [X29,#0x20+nTmp]
// .text:00000000000007D0 01 7C 10 13                ASR             W1, W0, #0x10
// .text:00000000000007D4 E0 66 82 52                MOV             W0, #0x1337
// .text:00000000000007D8 3F 00 00 6B                CMP             W1, W0
// .text:00000000000007DC 61 00 00 54                B.NE            loc_7E8
// .text:00000000000007E0 20 00 80 52                MOV             W0, #1
// .text:00000000000007E4 A0 1B 00 B9                STR             W0, [X29,#0x20+nFlag]
```

脚本：

```python
def my_mrs(ql: Qiling, address: int, size: int):
    opcode = ql.mem.read(address, size)
    if (0x7C8 == (address & 0xfff)) and (b"\x00\x00\x38\xD5" == opcode):
        ql.reg.write("w0", 0x1337<<0x10)
        ql.reg.arch_pc += 8;

def challenge11(ql: Qiling):
    ql.hook_code(my_mrs)
```

5. 参考资料

Qiling：一款功能强大的高级代码模拟框架 - FreeBuf网络安全行业门户

11个小挑战，Qiling Framework 入门上手跟练-软件逆向-看雪论坛-安全社区|安全招聘|bbs.pediy.com

Shielder - QilingLab – Release

zodf0055980/QilingLab_Writeup: QilingLab challenge writeup (github.com)