unicorn学习（2）-》》》 unicorn的简单应用

原创

pipixia233333 于 2020-01-31 15:04:00 发布 1121 收藏
分类专栏： web python开发
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/qq_41071646/article/details/104113731
版权


web python开发 专栏收录该内容
15 篇文章 0 订阅
订阅专栏

这个参考了一下ctf题

看了一篇国外的文章 感觉很不错 自己也拿来做一下看看 ，

hxp CTF 2017 Fibonacci

嘻嘻 这个名字 斐波那契

看到了这个程序的反汇编代码

```
  unsigned int v11; // edi
  int v13; // [rsp+Ch] [rbp-1Ch]

  v3 = &unk_4007E1;
  v4 = 0;
  setbuf(stdout, 0LL);
  printf("The flag is: ", 0LL);
  for ( i = 73LL; ; i = *(v3 - 1) )
  {
    v8 = 0LL;
    while ( 1 )
    {
      v13 = 0;
      sub_400670(v4 + v8, &v13, v5, v6, v8, i);
      v6 = v10;
      v8 = v10 + 1;
      v11 = v9 ^ (v13 << v6);
      if ( v8 == 8 )
        break;
      i = v11;
    }
    v4 += 8;
    if ( (v13 << v6) == v9 )
      break;
    ++v3;
    _IO_putc(v11, stdout);
  }
  _IO_putc(10, stdout);
  return 0LL;
}
```

https://blog.csdn.net/qq_41071646

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports | Exports

```
__int64 v11; // r9
__int64 result; // rax
unsigned int v13; // esi
unsigned int v14; // edx

v6 = a2;
if ( a1 )
{
  if ( a1 == 1 )
  {
    result = sub_400670(0LL, a2, a3, a4, a5, a6);
  }
  else
  {
    v7 = sub_400670((a1 - 2), a2, a3, a4, a5, a6);
    result = v7 + sub_400670((a1 - 1), a2, v8, v9, v10, v11);
  }
  v13 = ((result - ((result >> 1) & 0x55555555)) >> 2) & 0x33333333;
  v14 = v13
      + ((result - ((result >> 1) & 0x55555555)) & 0x33333333)
      + ((v13 + ((result - ((result >> 1) & 0x55555555)) & 0x33333333)) >> 4);
  *v6 ^= ((BYTE1(v14) & 0xF) + (v14 & 0xF) + ((((v14 >> 8) & 0xF0F0F) + (v14 & 0xF0F0F0F)) >> 16)) & 1;
}
else
{
  *a2 ^= 1u;
  result = 1LL;
}
return result;
}
```

https://blog.csdn.net/qq_41071646

用的是递归来斐波那契

但是斐波那契 数字越大 他的值 就越不好找，

其实常规做法可以把这个算法还原 然后打一个斐波那契的表 map一下就ok

但是我们要用unicron，

unicron 需要的步骤 代码与程序段加载、栈配置、特殊寄存器、外部调用patch

然后我先试了一下 。

我自己看着别的例子 自己写了一遍

发现了，

```
from unicorn import *
from unicorn.x86_const import *
from pwn import*

def hook_code(mu, address, size, user_data):
    if address == 0x400560:
        c = mu.reg_read(UC_X86_REG_RDI)
        print(chr(c))
        mu.reg_write(UC_X86_REG_RIP, address+size)


if __name__ == '__main__':
    try:
        mu = Uc(UC_ARCH_X86, UC_MODE_64)
        ADDRESS = 0x400000
        mmap_size=0x100000
        mu.mem_map(ADDRESS, mmap_size)#code
        mu.mem_map(0,mmap_size)#stack
        mu.mem_write(ADDRESS, read("./fibonacci"))#read process_file
        mu.reg_write(UC_X86_REG_RSP,mmap_size - 1)#set esp
        mu.hook_add(UC_HOOK_CODE, hook_code)
        mu.emu_start(0x00000000004004E0, 0x0000000000400575)
    except UcError as e:
        print("UcError %s") %(e)
```

报了下面的错误

UcError Invalid memory read (UC_ERR_READ_UNMAPPED)

读取了无效的内存，，，

看了一下 帖子 发现了 。

```
>>> Tracing instruction at 0x4004ef, instruction size = 0x7
UcError Invalid memory read (UC_ERR_READ_UNMAPPED)
pipixia@pipixias-MacBook-Pro /Volumes/Lenovo/unicorn/pc
```

然后在ida里面看一下发现了

```
xor     edx, edx
sub     rsp, 18h
mov     rdi, cs:stdout  ; stream
call    _setbuf
mov     edi, offset format ; "The flag is: "
xor     eax, eax
call    _printf
```

我们并没有加载bss段 只是要进行bss段访问的 我们只要跳过就好 还有调用printf 函数等

、python3

```
from unicorn import *
from unicorn.x86_const import *




def hook_code(mu, address, size, user_data):
    #print('>>> Tracing instruction at 0x%x, instruction size = 0x%x' %(address, size))
    #print hex(UC_X86_REG_RIP)
    if address in user_data:
            #print(hex(UC_X86_REG_RIP))
        mu.reg_write(UC_X86_REG_RIP, address+size)
    elif address == 0x400560:
        #print hex(address)
        c = mu.reg_read(UC_X86_REG_RDI)
        print(chr(c))
        mu.reg_write(UC_X86_REG_RIP, address+size)

if __name__ == '__main__':
    try:
        got_list=[0x00000000004004EF, 0x00000000004004F6, 0x0000000000400502, 0x000000000040054F]
        mu = Uc(UC_ARCH_X86, UC_MODE_64)
        ADDRESS = 0x400000
        mmap_size=0x100000
        mu.mem_map(ADDRESS, mmap_size)#code
        mu.mem_map(0,mmap_size)#stack
        mu.mem_write(ADDRESS, open('./fibonacci','rb').read())#read process_file
        mu.reg_write(UC_X86_REG_RSP,mmap_size - 1)#set esp
        mu.hook_add(UC_HOOK_CODE, hook_code,got_list)
        mu.emu_start(0x00000000004004E0, 0x0000000000400575)
    except UcError as e:
        print("UcError %s") %(e)
```

然后还是很慢（当然很慢 因为我们并没优化这个算法）

然后重点来了，

我们可以自己优化 ，， 这里给出的方案就是

每一次进入的时候 我们搞一个键对 参数 对等于函数的返回值

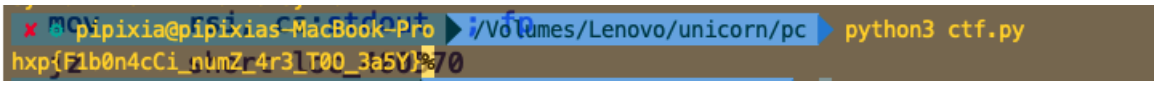
如果参数找到了 我们直接把这个返回值返回过去 不需要然后直接ret 不需要在计算一次了

会减杀很多很多的递归层数 然后调试了一下发现了 两个参数 其中一个参数只会是0 或者是1

```
from unicorn import *
from unicorn.x86_const import *

import struct


key_list={}
stack=[]
```

```
def u32(data):
    return struct.unpack("I", data)[0]

def p32(num):
    return struct.pack("I", num)


def hook_code(mu, address, size, user_data):
    #print('>>> Tracing instruction at 0x%x, instruction size = 0x%x' %(address, size))
    #print hex(UC_X86_REG_RIP)
    if address in user_data:
            #print(hex(UC_X86_REG_RIP))
        mu.reg_write(UC_X86_REG_RIP, address+size)
    elif address == 0x400560:
        #print hex(address)
        c = mu.reg_read(UC_X86_REG_RDI)
        print(chr(c),end="")
        mu.reg_write(UC_X86_REG_RIP, address+size)
    elif address==0x400670: #in f got
        arg_0 = mu.reg_read(UC_X86_REG_RDI)
        esi_r = mu.reg_read(UC_X86_REG_RSI)
        arg_1 = u32(mu.mem_read(esi_r,4))

        if (arg_0,arg_1) in key_list:
            (ret_rax,ret_bool)=key_list[(arg_0,arg_1)]
            mu.reg_write(UC_X86_REG_RAX, ret_rax)
            mu.mem_write(esi_r,p32(ret_bool))
            mu.reg_write(UC_X86_REG_RIP,0x400582)
        else:
            stack.append((arg_0,arg_1,esi_r))

    elif address in [0x00000000004006F1, 0x0000000000400709]: #in fun_end
        ret_rax=mu.reg_read(UC_X86_REG_RAX)
        (arg0, arg1, r_rsi) = stack.pop()
        ret_bool=u32(mu.mem_read(r_rsi,4))
        key_list[(arg0,arg1)]=(ret_rax,ret_bool)

if __name__ == '__main__':
    try:
        mu = Uc(UC_ARCH_X86, UC_MODE_64)
        ADDRESS = 0x400000
        mmap_size=0x100000
        user_data=[0x00000000004004EF, 0x00000000004004F6, 0x0000000000400502, 0x000000000040054F]
        mu.mem_map(ADDRESS, mmap_size)#code
        mu.mem_map(0,mmap_size)#stack
        mu.mem_write(ADDRESS, open('./fibonacci','rb').read())#read process_file
        mu.reg_write(UC_X86_REG_RSP,mmap_size - 1)#set esp
        mu.hook_add(UC_HOOK_CODE, hook_code,user_data)
        mu.emu_start(0x00000000004004E0, 0x0000000000400575)
    except UcError as e:
        print("UcError %s") %(e)
```

正确get flag！！！



然后还给了两个例子

我们接着那个arm的例子 搞定就ok了

arm的传参的方式 是

如果不超过4个参数 就 r0 r1 r2 r3 传参

这个例子其实和上面的很像

都是递归函数

```
from unicorn import *
from unicorn.arm_const import *
import struct




def u32(data):
    return struct.unpack("I", data)[0]

def p32(num):
    return struct.pack("I", num)

d={}
stack=[]



def hook_code(mu, address, size, user_data):
    #print('>>> Tracing instruction at 0x%x, instruction size = 0x%x' %(address, size))
    if address==user_data[0]:  #in fun start
        r0_v=mu.reg_read(UC_ARM_REG_R0)
        if r0_v in d:
            ret=d[r0_v]
            mu.reg_write(UC_ARM_REG_R0,ret)
            mu.reg_write(UC_ARM_REG_PC,0x105BC)
        else:
            stack.append(r0_v)
    elif address==user_data[1]:
        #print(hex(address),stack)
        ret=mu.reg_read(UC_ARM_REG_R0)
        r0_v=stack.pop()
        d[r0_v]=ret




if __name__ == '__main__':
    base_addr=0x10000
    size=0x100000
```

```
    stack_addr=0x200000
    mu=Uc(UC_ARCH_ARM,UC_MODE_LITTLE_ENDIAN)

    mu.mem_map(base_addr, size)
    mu.mem_map(stack_addr,size)

    mu.mem_write(base_addr, open("./task4","rb").read())
    mu.reg_write(UC_ARM_REG_SP,stack_addr + int(size/2))
    list_s_e=[0x000104D0,0x00010580]
    mu.hook_add(UC_HOOK_CODE, hook_code,list_s_e)
    mu.emu_start(0x00010584, 0x000105A8)
    ret=mu.reg_read(UC_ARM_REG_R1)
    print("[*]get_number:",ret)
```

unicorn 里面对andorid 的用处被挖掘了很多

发现很多帖子写的也很经典

unicorn 调用so库

其实一直对android so库一直很麻烦

有时候并不能很好的复现他的算法

但是 有些时候动态调试的时候又发现很多反调试 等等很多东西。

unicorn 就很好的解决了这个问题

如果单纯的想复现so里面的某个算法 其实是和上面差不多的

如果要是想把整个so的某个段 或者函数都搞定的话

还是需要pyelftools这个库 可以分析elf的某些区段

然后可以搞定防止模拟的时候访问到无效内存

这里的用法参考了无名侠大佬的看雪的帖子 参考链接放在了下面

直接把代码copy了出来，， ==

嘿嘿嘿 这里就直接偷懒了 代码来源于

https://mp.weixin.qq.com/s?__biz=MjM5NTc2MDYxMw==&mid=2458298594&idx=1&sn=4feaaf7959a89c3299fdd569535a60f6&chksm=b1

ELF 文件有两种视图，链接视图和执行视图。elftools 是基于链接视图解析ELF格式的，然而现在有一些ELF文件的section信息是被抹掉的，elftools就无法正常工作，我也没时间重写一个elf loader，就只能凑合用一下elftools。

我已经在前面一篇文章介绍了内存分配方面的东西，加载ELF文件第一步需要将ELF文件映射到内存。如何映射呢？只需要找到类型为PT_LOAD的segment，按照segment的信息映射即可。

代码如下：

```
# - LOAD (determinate what parts of the ELF file get mapped into memory)
load_segments = [x for x in elf.iter_segments() if x.header.p_type == 'PT_LOAD']

for segment in load_segments:
    prot = UC_PROT_ALL
    self.emu.memory.mem_map(load_base + segment.header.p_vaddr, segment.header.p_memsz, prot)
    self.emu.memory.mem_write(load_base + segment.header.p_vaddr, segment.data())
```

解析 init_array

ELF 的有一个列表，用于存储初始化函数地址，在动态链接的时候，linker会依次调用init_array中的每一个函数。init_array 中的函数一般用于初始化程序，偶尔也有ELF外壳程序在init_array中添加自解密代码，另外有一些字符串解密也是在init_array中完成的。想要模拟native程序，必然需要调用init_array 中的函数。

init_array 是一个数组， 一般情况下，每一项都是函数入口偏移， 然而也有为0的情况。因为init_array实际解析时机在重定位完成之后， init_array 也可能被重定位。 所以要解析init_array的时候还需要判断重定位表。

我的策略是，当读出init_array中为0的条目的时候就去重定位表中查找重定位值。

```
for _ in range(int(init_array_size / 4)):
    # covert va to file offset
    for seg in load_segments:
        if seg.header.p_vaddr <= init_array_offset < seg.header.p_vaddr + seg.header.p_memsz:
            init_array_foffset = init_array_offset - seg.header.p_vaddr + seg.header.p_offset
    fstream.seek(init_array_foffset)
    data = fstream.read(4)
    fun_ptr = struct.unpack('I', data)[0]
    if fun_ptr != 0:
        # fun_ptr += load_base
        init_array.append(fun_ptr + load_base)
        print ("find init array for :%s %x" % (filename, fun_ptr))
    else:
        # search in reloc
        for rel in rel_section.iter_relocations():
            rel_info_type = rel['r_info_type']
            rel_addr = rel['r_offset']
            if rel_info_type == arm.R_ARM_ABS32 and rel_addr == init_array_offset:
                sym = dynsym.get_symbol(rel['r_info_sym'])
                sym_value = sym['st_value']
                init_array.append(load_base + sym_value)
                print ("find init array for :%s %x" % (filename, sym_value))
                break
    init_array_offset += 4
```

解析符号

32位ELF文件 symbol table entry 的定义如下。

```
typedef struct {
        Elf32_Word      st_name;
        Elf32_Addr      st_value;
        Elf32_Word      st_size;
        unsigned char   st_info;
        unsigned char   st_other;
        Elf32_Half      st_shndx;
} Elf32_Sym;
```

当st_shndx字段的值为SHN_UNDEF时，表明该符号在当前模块没有定义，是一个导入符号，要去其它模块查找。为了便于管理已经加载模块的符号地址，应该用一个map，将name和address映射起来。
其它情况，简单起见，均看成导出符号，将地址重定位后加入到管理符号map。

```
# Resolve all symbols.
symbols_resolved = dict()

for section in elf.iter_sections():
    if not isinstance(section, SymbolTableSection):
        continue
    itersymbols = section.iter_symbols()
    next(itersymbols)  # Skip first symbol which is always NULL.
    for symbol in itersymbols:
        symbol_address = self._elf_get_symval(elf, load_base, symbol)
        if symbol_address is not None:
            symbols_resolved[symbol.name] = SymbolResolved(symbol_address, symbol)

def _elf_get_symval(self, elf, elf_base, symbol):
    if symbol.name in self.symbol_hooks:
        return self.symbol_hooks[symbol.name]

    if symbol['st_shndx'] == 'SHN_UNDEF': # 外部符号
        # External symbol, lookup value.
        target = self._elf_lookup_symbol(symbol.name)
        if target is None:
            # Extern symbol not found
            if symbol['st_info']['bind'] == 'STB_WEAK':
                # Weak symbol initialized as 0
                return 0
            else:
                logger.error('=> Undefined external symbol: %s' % symbol.name)
                return None
        else:
            return target
    elif symbol['st_shndx'] == 'SHN_ABS':
        # Absolute symbol.
        return elf_base + symbol['st_value']
    else:
        # Internally defined symbol.
        return elf_base + symbol['st_value']
```

重定位

```
# Relocate.
for section in elf.iter_sections():
    if not isinstance(section, RelocationSection):
        continue
    #for relsection in elf.get_dynmic_rel():
    for rel in section.iter_relocations():
        sym = dynsym.get_symbol(rel['r_info_sym'])
        sym_value = sym['st_value']

        rel_addr = load_base + rel['r_offset']  # Location where relocation should happen
        rel_info_type = rel['r_info_type']

        # Relocation table for ARM
        if rel_info_type == arm.R_ARM_ABS32:
            # Create the new value.
            value = load_base + sym_value
            # Write the new value
            self.emu.mu.mem_write(rel_addr, value.to_bytes(4, byteorder='little'))

        elif rel_info_type == arm.R_ARM_GLOB_DAT or \
                rel_info_type == arm.R_ARM_JUMP_SLOT or \
                rel_info_type == arm.R_AARCH64_GLOB_DAT or \
                rel_info_type == arm.R_AARCH64_JUMP_SLOT:
            # Resolve the symbol.
            if sym.name in symbols_resolved:
                value = symbols_resolved[sym.name].address

                # Write the new value
                self.emu.mu.mem_write(rel_addr, value.to_bytes(4, byteorder='little'))
        elif rel_info_type == arm.R_ARM_RELATIVE or \
                rel_info_type == arm.R_AARCH64_RELATIVE:
            if sym_value == 0:
                # Load address at which it was linked originally.
                value_orig_bytes = self.emu.mu.mem_read(rel_addr, 4)
                value_orig = int.from_bytes(value_orig_bytes, byteorder='little')

                # Create the new value
                value = load_base + value_orig

                # Write the new value
                self.emu.mu.mem_write(rel_addr, value.to_bytes(4, byteorder='little'))
            else:
                raise NotImplementedError()
        else:
            logger.error("Unhandled relocation type %i." % rel_info_type)
```

不过如果就简单的搞一些算法 就没有必要搞得那么认真

这里找一些demo 然后搞定就ok

上面差不多 就不多说了

然后重点就是 怎么与jni 优雅的交互

这里有一个 看雪链接的介绍 什么时候有大把的时间空隙了 可以研究一下

AndroidNativeEmu 这个项目

https://bbs.pediy.com/thread-254799.htm

参考链接

https://mp.weixin.qq.com/s?__biz=MjM5NTc2MDYxMw==&mid=2458298594&idx=1&sn=4feaaf7959a89c3299fdd569535a60f6&chksm=b1

参考链接

原帖

http://eternal.red/2018/unicorn-engine-tutorial/

看雪翻译贴

https://bbs.pediy.com/thread-224330.htm