

hgame week4 writeup by v0id

Reverse

vm

用 switch 实现的一个小 vm，使用基于 ida python 和 unicorn engine 的模拟器 flare-emu，写一个小的 trace 脚本进行分析。

```
from __future__ import print_function
import flare_emu

vm_code = 0
taintedMem = {}
#taintedReg

regTable = [{'eax':0, 'rax':0, 'al':0}, \
             {'ebx':0, 'rbx':0, 'bl':0}, \
             {'ecx':0, 'rcx':0, 'cl':0}, \
             {'edx':0, 'rdx':0, 'dl':0}, \
             {'r8b':0, 'r8d':0, 'r8':0}, \
             {'r9b':0, 'r9d':0, 'r9':0}, \
             {'rdi':0, 'edi':0}, \
             {'rsi':0, 'esi':0}, \
             {'r10':0}, \
             {'r11':0}]

count = 0
wantId = 1
def printLog(address,id,addition):
    if(wantId==0):
        print('[%d] 0x%x : %s %s'%(count,address,GetDisasm(address),addition))
    elif(id==wantId):
        print('[%d] 0x%x : %s %s'%(count,address,GetDisasm(address),addition))

def insHook(uc, address, instructionSize, userData):
    global myEH,vm_code
    if(address==0x140001A84):
        vm_code = myEH.allocEmuMem(0x280)
        uc.reg_write(myEH.regs['rax'],vm_code)
        print('vm_code is stored @ '+hex(myEH.getRegVal('rax')))

def accessHook(uc, accessType, memAccessAddress, memAccessSize, memValue,
userData):
    pc = myEH.getRegVal('pc')
    if(accessType==0x10 and GetOpType(pc,0)==o_reg):
        op1 = GetOpnd(pc,0)
        if(memAccessAddress in taintedMem and taintedMem[memAccessAddress]!=0):
            for i in range(10):
                if(op1 in regTable[i]):
                    for j in regTable[i].keys():
                        regTable[i][j] = taintedMem[memAccessAddress]
```

```

        break
    printLog(pc,taintedMem[memAccessAddress], ' read = \'
        + hex(struct.unpack('B',myEH.getEmuBytes(memAccessAddress,1))[0]\
        if (memAccessSize==1) \
        else myEH.getEmuPtr(memAccessAddress)))
else:
    for i in range(10):
        if(op1 in regTable[i]):
            for j in regTable[i].keys():
                regTable[i][j] = 0
            break
elif(accessType==0x11):
    if(GetOpType(pc,1)==o_reg):
        if('mov' in GetMnem(pc)):
            tainted = 0
            op2 = GetOpnd(pc,1)
            for i in range(10):
                if(op2 in regTable[i] and regTable[i][op2]!=0):
                    tainted = regTable[i][op2]
            if(tainted==0):
                taintedMem[memAccessAddress] = 0
                return
            taintedMem[memAccessAddress] = tainted
            printLog(pc,taintedMem[memAccessAddress], ' write =
'+hex(memValue))
            elif(memAccessAddress in taintedMem and
taintedMem[memAccessAddress]!=0):
                printLog(pc,taintedMem[memAccessAddress], ' write =
'+hex(memValue))
        else:
            pass

def hookCode(uc, address, instructionSize, userData):
    global count
    count+=1
    if(GetMnem(address)=='inc'):
        op1 = GetOpnd(address,0)
        for i in range(10):
            if(op1 in regTable[i] and regTable[i][op1]!=0):
                printLog(pc,regTable[i][op1], '')
                return
    for i in range(2):
        if(GetOpType(address,i)!=o_reg):
            return
    op1 = GetOpnd(address,0)
    op2 = GetOpnd(address,1)
    tainted = 0
    for i in range(10):
        if(op2 in regTable[i] and regTable[i][op2]!=0):
            tainted = regTable[i][op2]
            break
    if(tainted==0):
        for i in range(10):
            if(op1 in regTable[i]):

```

```

        for j in regTable[i].keys():
            regTable[i][j] = 0

    return
    for i in range(10):
        if(op1 in regTable[i]):
            for j in regTable[i].keys():
                regTable[i][j]=tainted
    printLog(pc,tainted, '')

myEH = flare_emu.EmuHelper()
myEH.emulateRange(0x140001640,instructionHook=insHook)
myEH.emulateRange(0x1400039C0)

ptr_vm_code = myEH.allocEmuMem(0x8)
flag = myEH.allocEmuMem(0x100)
vm_stack = myEH.allocEmuMem(0x100)

print('vm_stack is stored @ '+hex(vm_stack))
myEH.writeEmuPtr(0x14000E398,vm_stack)
myEH.writeEmuPtr(ptr_vm_code,vm_code)
myEH.writeEmuPtr(ptr_flag,flag)
myEH.writeEmuMem(flag,b'1234567890abcdefghijklmnopqrstuvwx')

for i in range(1,35):
    taintedMem[flag+i-1]=i

myEH.emulateRange(0x140003CC0,registers =
{"arg1":ptr_vm_code,"arg2":flag},memAccessHook=accessHook,instructionHook=hookCode
)

```

nllvm

ida 的字符串窗口中找不到 exe 打印出的提示，动态调试，在输入的地方暂停，栈回溯找到 main 函数。

```

__int64 sub_1400105F0()
{
    HANDLE v0; // rax
    __int64 v1; // rax
    __int64 v2; // rax
    __int64 v3; // rax
    HANDLE v4; // rax
    __int64 v5; // rax
    __int64 v6; // rax
    __int64 v7; // rax
    int v8; // ecx
    HANDLE v9; // rax
    __int64 v10; // rax
    HANDLE v11; // rax
    char v13; // [rsp+50h] [rbp-1B8h]

```

```

char v14; // [rsp+150h] [rbp-B8h]
__int64 v15; // [rsp+190h] [rbp-78h]
__int64 v16; // [rsp+198h] [rbp-70h]
char v17; // [rsp+1A0h] [rbp-68h]
int v18; // [rsp+204h] [rbp-4h]

v18 = 0;
v0 = GetStdHandle(0xFFFFFFFF5);
SetConsoleTextAttribute(v0, 0xFu);
byte_140051281 ^= 0xCEu;
byte_140051282 ^= 0xD1u;
byte_140051283 ^= 0xC1u;
byte_140051284 ^= 0xB6u;
byte_140051285 ^= 0x78u;
byte_140051286 ^= 0x82u;
byte_140051287 ^= 0xE5u;
byte_140051288 ^= 0xE6u;
byte_140051289 ^= 0x5Bu;
byte_14005128A ^= 0xC4u;
byte_14005128B ^= 0xA1u;
byte_14005128C ^= 0x6Fu;
.....//引用前解密字符串
v1 = sub_140017AE0(&off_140051730, &byte_140051281);
byte_140051281 ^= 0xCEu;
byte_140051282 ^= 0xD1u;
byte_140051283 ^= 0xC1u;
byte_140051284 ^= 0xB6u;
byte_140051285 ^= 0x78u;
byte_140051286 ^= 0x82u;
.....//引用后加密回去

```

原来程序对一些字符串和数组进行了加密，运行的时候再动态解密，解密过程就是简单的异或，写个小脚本还原一下。

```

def match(address):
    ins2 = NextHead(address)
    ins3 = NextHead(ins2)
    #每条指令为一组进行判断
    #如果符合这些特征，就提取出加密字节的地址和异或的 key
    cond1 = GetMnem(address)=='mov' and GetOpType(address,0)==o_reg
    cond2 = GetMnem(ins2)=='xor' and GetOpType(ins2,0)==o_reg
    cond3 = GetMnem(ins3)=='mov' and GetOpType(ins3,1)==o_reg
    if cond1 and cond2 and cond3:
        encrypted_addr = GetOperandValue(address,1)
        key = GetOperandValue(ins2,1)
        for i in range(address,NextHead(ins3)):
            PatchByte(i,0x90)
            #把原来的指令 patch 掉，写脚本的时候要注意，exe 引用字符串以后，会把他重新
            加密回去，记得也要 patch 掉。
        return (1,encrypted_addr,key)
    else:

```

```

        return (0,0,0)

visited = {}#这个字典保存了已经解密的字节的内存地址
for func in Functions():
    #遍历所有函数
    flags = idc.GetFunctionFlags(func)
    if flags & FUNC_LIB or flags & FUNC_THUNK:
        continue
    #忽略库函数
    dism_addr = list(idutils.FuncItems(func))
    start_addr = 0
    for line in dism_addr:
        #对函数中每条指令进行判断, 如果形式符合: xor reg,0xCE, 就开始解密
        cond = GetMnem(line)=='xor' and GetOpType(line,0)==o_reg and
        GetOperandValue(line,1)==0xCE
        if line > start_addr and cond:
            print 'found encrypt block @ '+hex(line)
            start_addr = line - 6
            #从mov reg,byte_00000000开始匹配
            while(1):
                isMatch,encrypted_addr,key = match(start_addr)
                if(isMatch):
                    if(encrypted_addr not in visited):
                        #如果得到匹配, 并且该地址还没解密
                        PatchByte(encrypted_addr,Byte(encrypted_addr)^key)
                        visited[encrypted_addr]=1
                    for i in range(3):
                        start_addr = NextHead(start_addr)
                else:
                    #不匹配说明解密结束了
                    break

```

patch 后的 main 函数, 好看多了。



调试猜测出几个函数的意义。

```

if ( sub_140025FC0(&v15) == 64 )//strlen(input)==64
.....
0 == sub_140021530(&byte_140051240, &v12, 64i64)//strncmp(s1,s2,64)

```

最后发现是 AES , 解密一下就ok了。

Last build: 4 years ago - Dark theme now available in 'Options' [Options](#) [About / Support](#)

Recipe

AES Decrypt

Passphrase/Key **UTF8** CryptoFAILUREforRSA2048Key!!

IV **Hex** 000102030405060708090a0b0c0d0e0f

Salt **Hex**

Mode **CBC** Padding **NoPadding**

Input format **Hex**

Output format **UTF8**

Bake!

☒ Auto Bake

Save recipe

Load recipe

Clear recipe

Step

Clear breakpoints

Input

length: 128
lines: 1

Clear I/O

Reset layout

91b3c1eb145dd5ce3a1d30e4706c6bd769787902a3a5df1bfd1c028914207afd2452f8a9f9f16b1c0f5d505bec42d18cb812cf2ca9693146fd9beadec8bf9469

Output

start: 64
end: 64
length: 0

time: 8ms
length: 64
lines: 1

Save to file

Move output to input

Undo

Max

hgame{cosmos_is_still_fighting_and_NEVER_GIVE_UP_000000000000}

A 5 Second Challenge

使用 IL2CppDumper 分析,写 frida 脚本对 checkBombAt 函数进行主动调用。

```
var mbase = Module.getBaseAddress('GameAssembly.dll');
console.log(mbase)
var checkBombFunc = mbase.add(0x5752A0);
var patch = mbase.add(0x570D05);
var patch2 = mbase.add(0x5752F9);
var checkTimeFunc = mbase.add(0x575560);
const maxPatchSize = 64;
Memory.patchCode(patch, maxPatchSize,function(code){
    const cw = new X86Writer(code, { pc: patch });
    cw.putJmpAddress(mbase.add(0x570CFF));
    cw.flush();
});
const maxPatchSize = 64;
Memory.patchCode(patch2, maxPatchSize,function(code){
    const cw = new X86Writer(code, { pc: patch2 });
    cw.putJmpAddress(mbase.add(0x5753B8));
    cw.flush();
});
Interceptor.attach(checkTimeFunc,{
    onLeave:function(retval){
        retval.replace(0);
    }
});
Interceptor.attach(checkBombFunc,{
    onEnter:function(args){
```

```

        if(Memory.readPointer(this.context.rsp)==Number(mbase.add(0x570C86))){
            console.log(this.context.rcx);
        }
    },
    onLeave:function(retval){
        if(Memory.readPointer(this.context.rsp)==Number(mbase.add(0x570C86))){
            console.log(retval);
        }
    }
});
/*
function hex2float(num) {
    var sign = (num & 0x80000000) ? -1 : 1;
    var exponent = ((num >> 23) & 0xff) - 127;
    var mantissa = 1 + ((num & 0x7fffff) / 0x7fffff);
    return sign * mantissa * Math.pow(2, exponent);
}*/
function DecToBinTail(dec, pad)
{
    var bin = "";
    var i;
    for (i = 0; i < pad; i++)
    {
        dec *= 2;
        if (dec >= 1)
        {
            dec -= 1;
            bin += "1";
        }
        else
        {
            bin += "0";
        }
    }
    return bin;
}
function DecToBinHead(dec,pad)
{
    var bin="";
    var i;
    for (i = 0; i < pad; i++)
    {
        bin = (parseInt(dec % 2).toString()) + bin;
        dec /= 2;
    }
    return bin;
}
function get_float_hex(decString)
{
    var dec = decString;
    var sign;
    var signString;
    var decValue = parseFloat(Math.abs(decString));
    if (decString.toString().charAt(0) == '-')

```

```
{
    sign = 1;
    signString = "1";
}
else
{
    sign = 0;
    signString = "0";
}
if (decValue==0)
{
    var fraction = 0;
    var exponent = 0;
}
else
{
    var exponent = 127;
    if (decValue>=2)
    {
        while (decValue>=2)
        {
            exponent++;
            decValue /= 2;
        }
    }
    else if (decValue<1)
    {
        while (decValue < 1)
        {
            exponent--;
            decValue *= 2;
            if (exponent ==0)
                break;
        }
    }
    if (exponent!=0) decValue-=1; else decValue /= 2;

}
var fractionString = DecToBinTail(decValue, 23);
var exponentString = DecToBinHead(exponent, 8);
return parseInt(signString + exponentString + fractionString, 2).toString(16);
}
var func = new NativeFunction(checkBombFunc,'int',['uint64','int']);
for(var i=0;i<=44;i++){
    console.log(func(Number('0x'+get_float_hex(i)+get_float_hex(44.0)),0));
}
```

找到所有雷的位置，就可以生成一张二维码拿 flag 了。