## 一、漏洞信息

### 1. 漏洞简述

* 漏洞名称：MS08-067
* 漏洞编号：CVE-2008-4250
* 漏洞类型：栈溢出漏洞
* 漏洞影响：远程代码执行
* CVSS评分：（不知道）
* 利用难度：Medium
* 基础权限：不需要

### 2. 组件概述

Netsvcs是svchost.exe使用的子进程，属于Windows的基础组件，负责网络相关的工作，此例中负责进行远程过程的参数的接收和过程的执行，以及执行后结果的回送。

### 3. 漏洞利用

此漏洞属于远程代码执行漏洞，在使用RPC调用靶机的函数的时候通过构造一段特殊的参数可以达到执行隐藏在参数中恶意代码的目的。

### 4. 漏洞影响

此漏洞在Windows 2000；XP；Server2003上严重影响，在Windows Vista；Server 2008；7 Beta上为重要影响。

### 5. 解决方案

下载安装官方给出的补丁（KB958644）。

## 二、漏洞复现（远程和本地）

### 1. 环境搭建

（1）本地

* 靶机环境

操作系统:Windows xp sp2 eng。

编译器：VC6.0。

* 靶机配置

（由于从Windows xp sp2开始有了数据执行保护DEP靶机配置分为开启DEP和关闭DEP两种）

DEP的开启和关闭（分为两种一种是全局关闭，一种是为指定程序关闭）

全局关闭：（全局打开针对的是系统服务进程）

通过按WIN+R运行C:boot.int打开boot.ini

在boot.ini中初始为如下所示的字符串，为全局打开

[operating systems]

multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect /NoExecute=OptIn

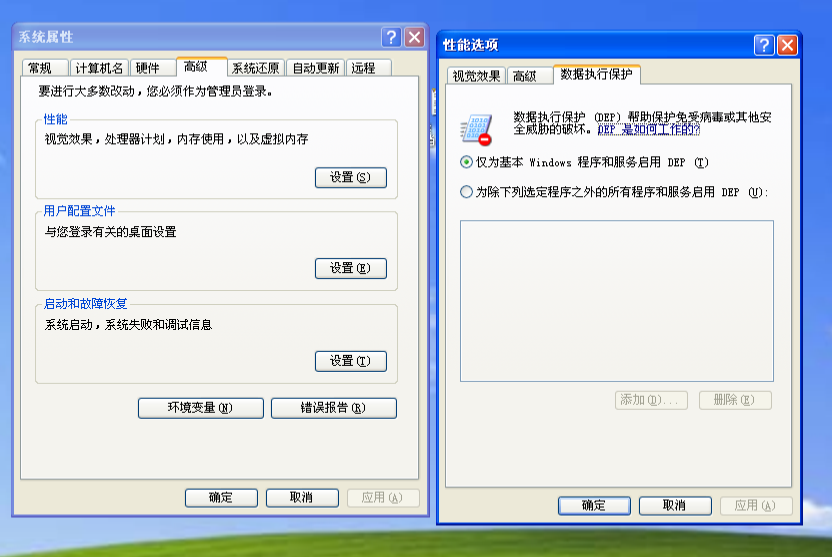
修改为：

[operating systems]

multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect /Execute

为全局关闭。

指定程序关闭：我的电脑->属性->高级->性能->设置->数据执行保护中指定程序关闭。



**图2.1.1 为指定的程序关闭DEP**

（2）远程

* 靶机环境

操作系统：windows xp sp2 eng。

* 靶机配置

DEP开启或关闭同本地。

开启共享管道：通过在CMD上输入net share开启。

* 攻击机环境

操作系统：Windows xp sp3(简体中文版)。

编译器: VS2008。

* 攻击机配置

匿名连接靶机共享管道：

在cmd上输入net use \\靶机IP\IPC$ “” /user:“”连接。

### 2. 复现过程

（1）本地（关闭DEP）

将Windows\local\closedep\exploit.c复制到靶机VC6.0新建的项目中编译运行即可显示弹窗kingtest。(编译为release版本并关闭代码优化，详见ms06-040中操作)。



**图2.2.1 运行效果**

（2）本地(开启DEP)

将Windows\local\opendep\exploit复制到靶机VC6.0新建的项目中编译运行即可显示弹窗kingtest。(编译为release版本并关闭代码优化，详见ms06-040中操作)。



**图2.2.2 运行效果**

假如依旧运行关闭DEP使用的代码将出现如图所示的界面。

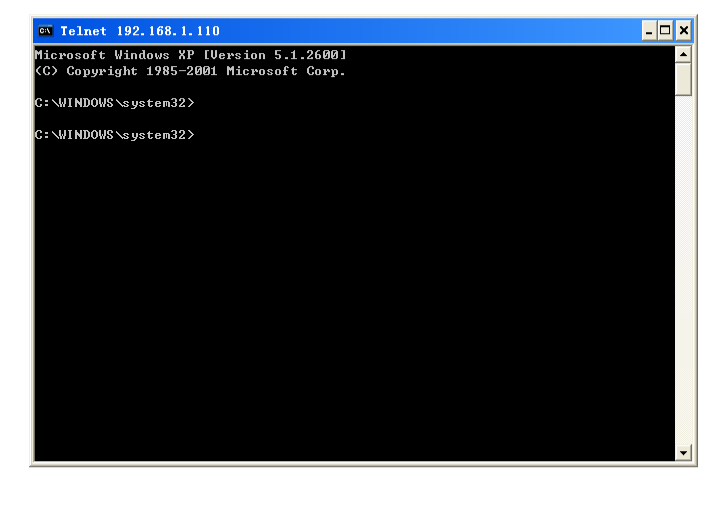


**图2.2.3 在开启DEP的条件下运行未开启的利用代码**

（3）远程

远程的利用在DEP开启的条件下进行，利用代码处于Windows\rpc下有三个文件exploit.idl、exploit.acf、exploit.c在攻击机的VS2008编译器下编译运行(代码的构建过程详见MS06-040中的分析)

运行编译后的程序后打开cmd输入:telnet 靶机IP 6666进行正向连接，会得到靶机shell。



**图2.2.4 远程执行效果**

## 三、漏洞分析

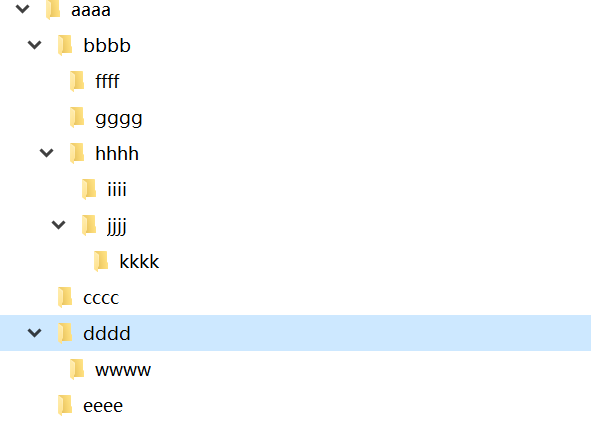
### 1. 基本信息

* 漏洞文件：netapi32.dll
* 漏洞函数：NetpwPathCanonicalize
* 漏洞对象：/xxxx…xxxxx/../../xxx…

### 2. 背景知识

(移经操作)

熟悉命令行的都知道cd （文件夹名）是进入该文件夹，cd .不执行任何操作，cd ..为返回上一文件夹，类似的cd ../name为进入上一目录下的name文件夹。如图3.2.1所示的文件夹结构：



**图3.2.1 文件结构示意图**

在执行变更文件夹的操作中cd /aaaa/../bbbb/gggg/../../dddd/wwww和cd dddd/wwww的操作结果一致，在进行参数传递的时候后者优于前者，所以就要构造一套规则（函数）将前边的字符串转化为后边的字符串，这就是移经操作的核心。

转到编程层面要实现前边字符串到后边字符串的转化可以借助如图所示的规则：



**图3.2.2 操作结构图**

开始的时候p1处于1的位置处然后向后检索假如为字符接着向后检索，假如是\ p1指向此p2指向刚刚p1的位置接着向后当p1指向3时,p2指向2,p3指向1,接着向后检索就会出现如图所示的情况，此时就要移除p1和p3之间的内容,并把p2更新为p3的位置,而p3向前检索直到检索到\让p3指向此处。让p1和p2指向相同位置，接着让p1向后检索重复以上步骤直到把字符串中的/../全部剔除，这就是移经操作的编程思路，这里的情况恰好是正常的情况，假如遇到如图所示的情况：



**图3.2.3 特殊情况**

在这种情况下正确的处理是P2更新为p3指向的位置后P3判断是否是边界是边界P3的位置就不能改变了，但此漏洞中边界的检测有问题，致使P3依旧会向前检索‘\’使p3指向前边越界的‘\’，这个时候P3指向前边越界的‘\’P2指向1处，1后的字符串被3后的字符串替代变为如图所示的结构：



**图3.2.4 变更后**

这是p1接着从1向后检索当p1指向4时将4后的字符串复制到5后这时就造成了溢出，假如1前有返回地址，这次溢出就会将返回地址覆盖进而执行恶意代码。

（2）DEP绕过(这一部分的内容单独讲解，位于同目录下的DEP分析中)。

### 3. 详细分析

#### 1. 基础分析

本例中使用的是window xp sp2 eng版本的windows/system32/netapi32.dll文件，大小为3332288字节，产生漏洞的函数为NetpwPathCanonicalize，此函数有6个参数：

{

char path[];

char can\_path[];

int maxbuf;

char prefix[];

int \* pathtype;

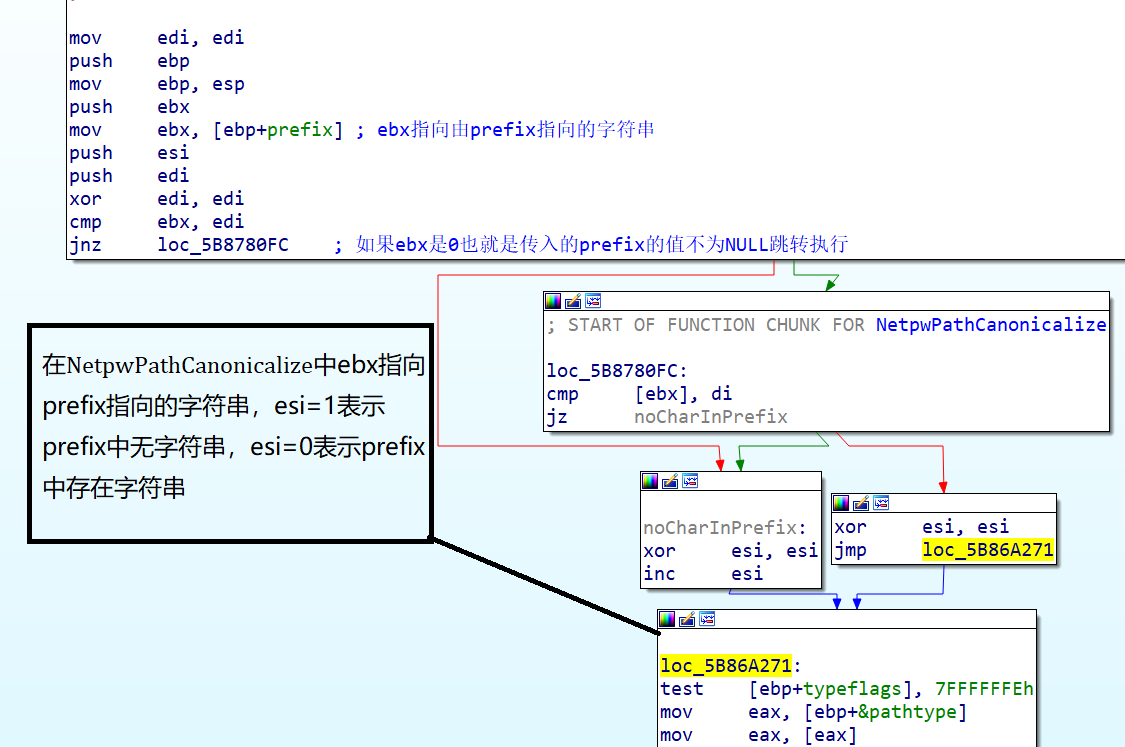
int pathflags;

}

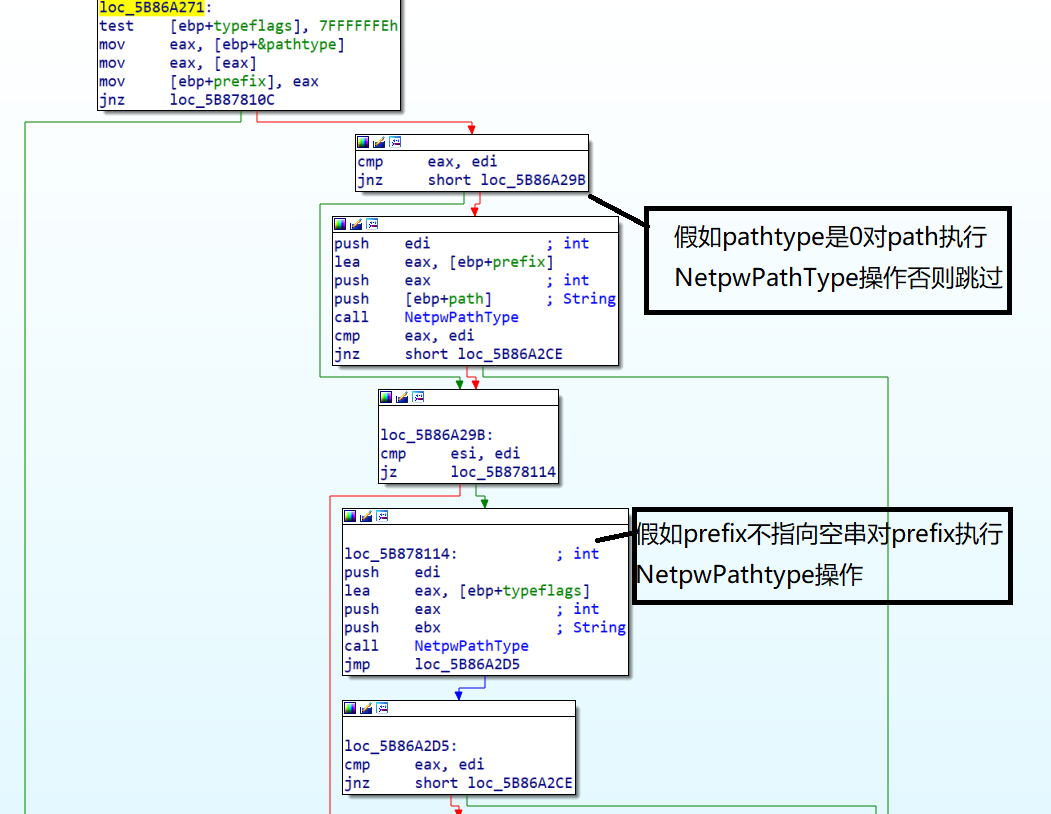
功能简单的讲是将prefix[] + ’\’ + path[] -> can\_path[];maxbuf用来限制can\_path的长度，完成的功能就是两个字符串的连接，不过这两个字符串是用来表示路径的。这样连接后的字符串就会存在上边移经操作中去除\..\的过程，最后将格式化后的连接串复制到can\_path中。此漏洞就在于将连接好的字符串进行格式化的过程中并未进行有效的边界检测使得在向前（低地址处）查找’\’的过程中越过了边界，造成了栈溢出。

#### 2. 静态分析

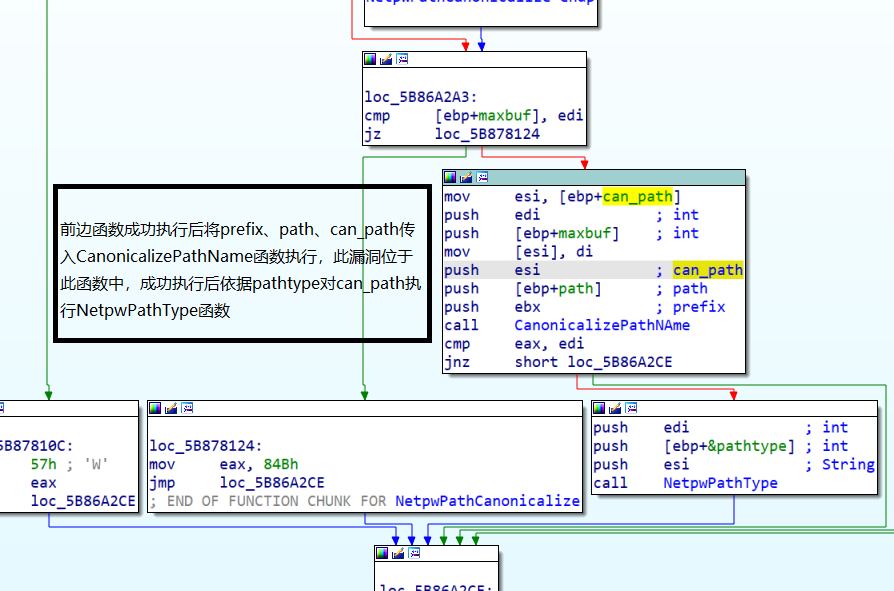
将存在漏洞的netapi32.dll文件拖入IDA pro 32位版本中，定位到漏洞函数NetpwPathCanonicalize的位置，并跳转到此函数的汇编代码中。



**图3.3.1 分析1**

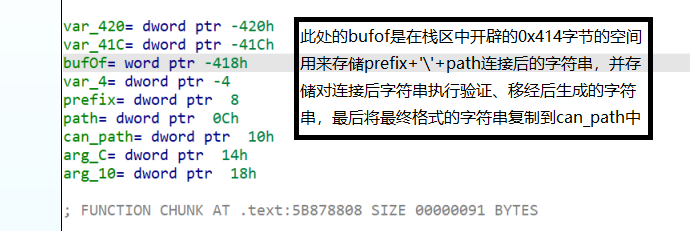


**图3.3.2 分析2**



**图3.3.3 分析4**

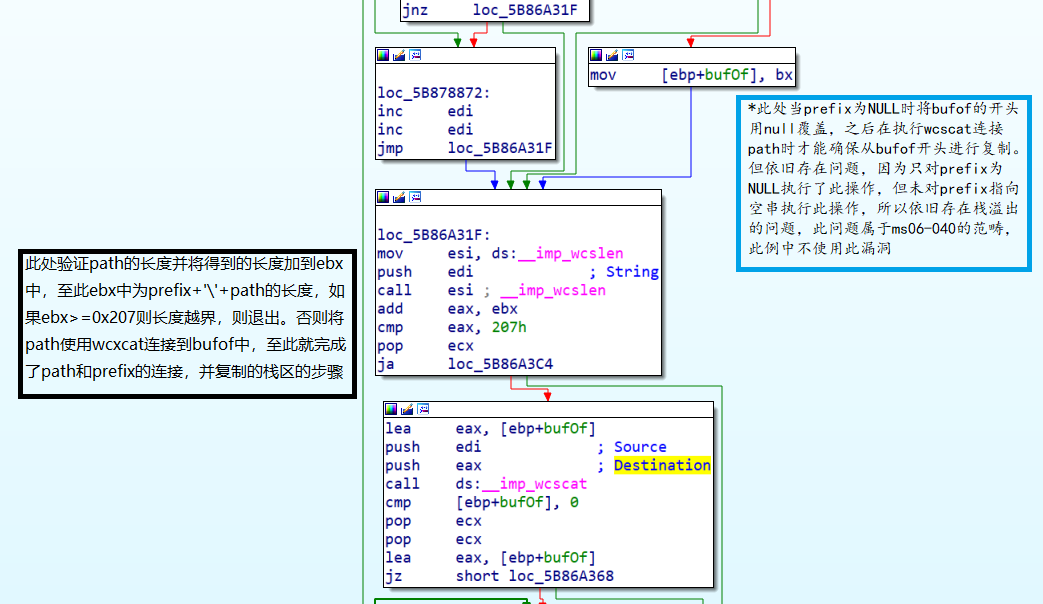
在上述分析中可以得知NetpwPathCanonicalize主体中大致是对prefix和path执行NetpwPathtype操作（由于漏洞与此函数无关因此不深入分析），执行操作后的path和Prefix以及can\_path传入Canonicalize执行连结、格式验证、移经并复制到can\_path的操作，接下来深入分析Canonicalize，跳转到其汇编中。



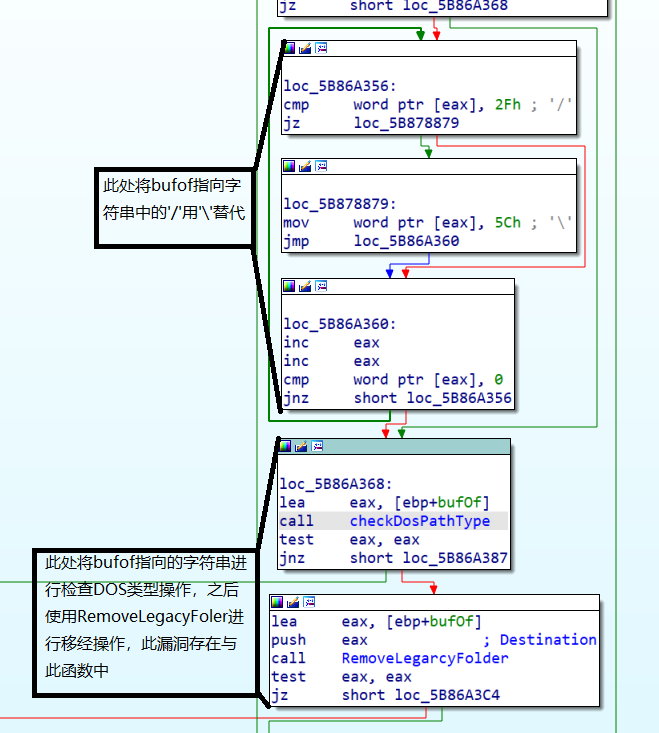
**图3.3.4 分析5**



**图3.3.5 分析5**

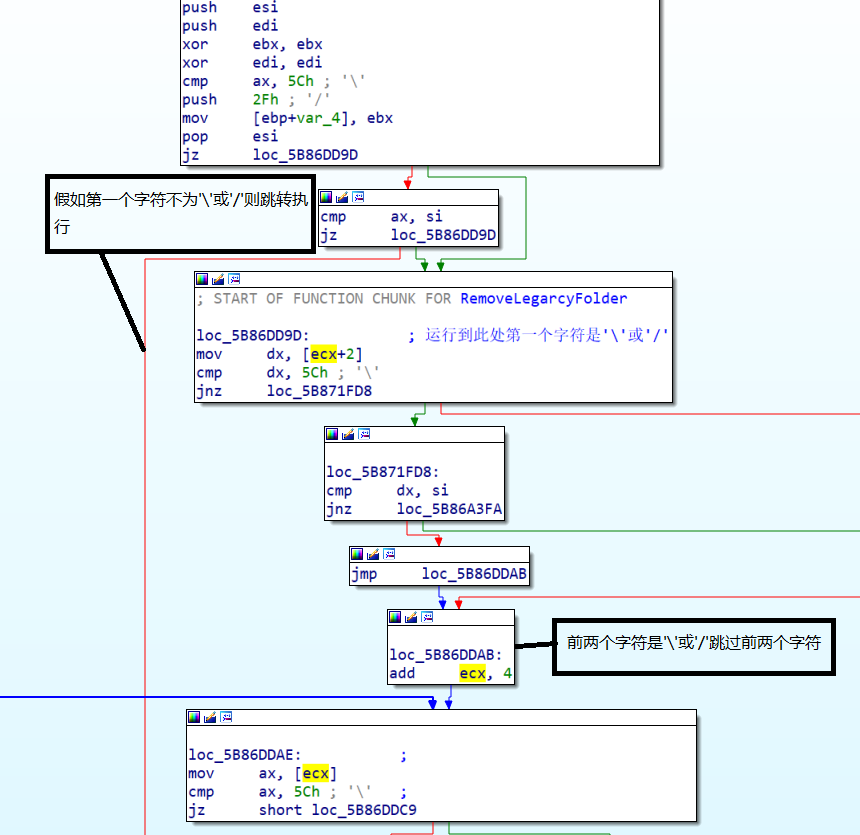
****

**图3.3.6 分析6**

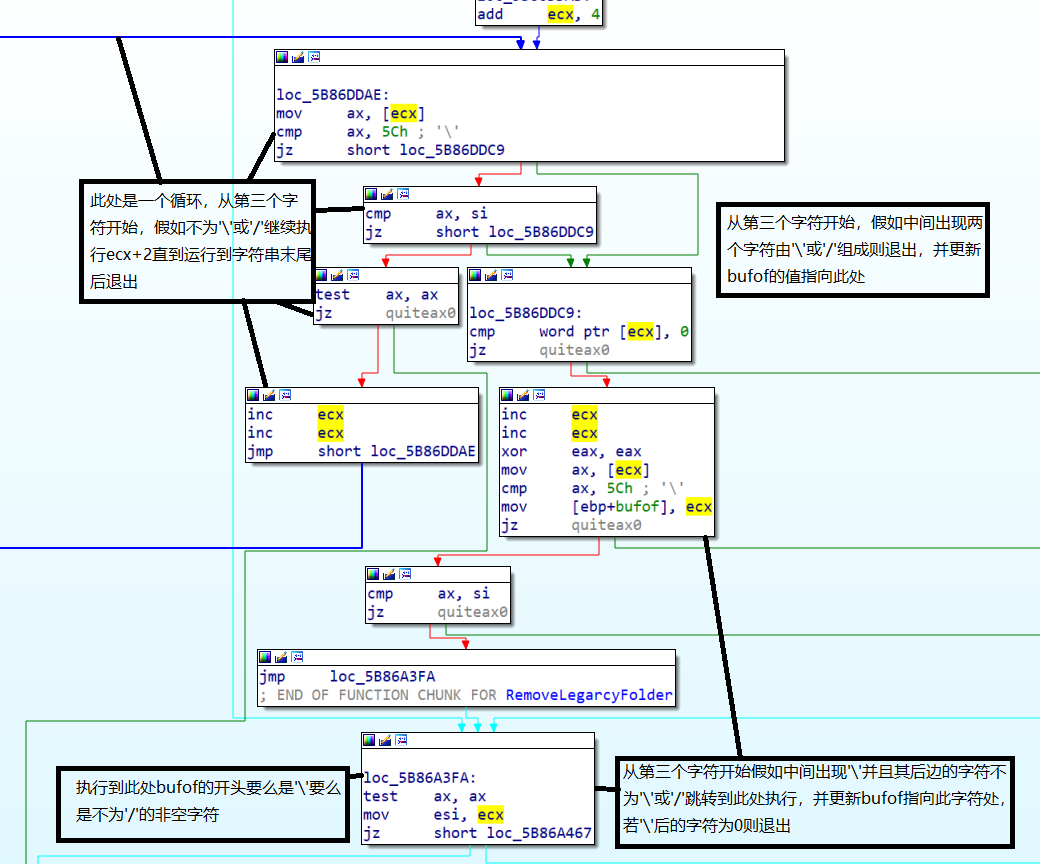
****

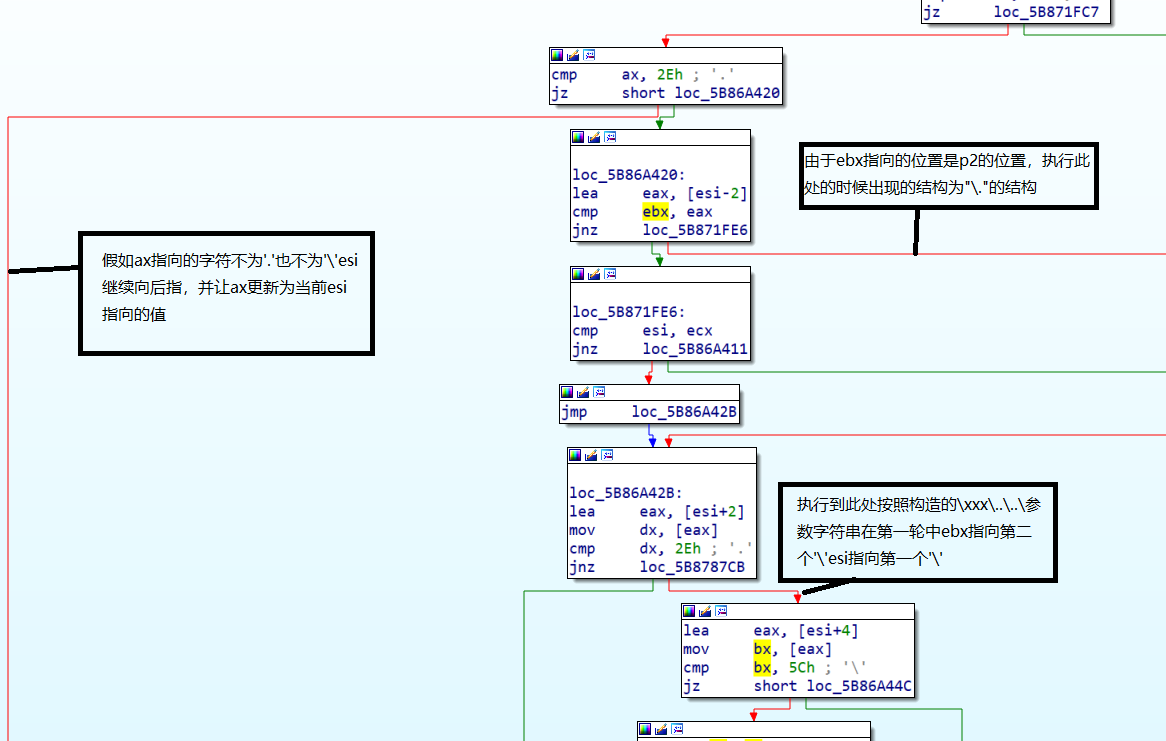
**图3.3.7 分析7**

成功执行到此以后就把bufof中保存的字符串复制到can\_path中并退出。此漏洞位于RemoveLegacyFolder中，在checkDosPathType中存在着漏洞利用的关键一步，这里暂且不表，先对漏洞函数进行分析。

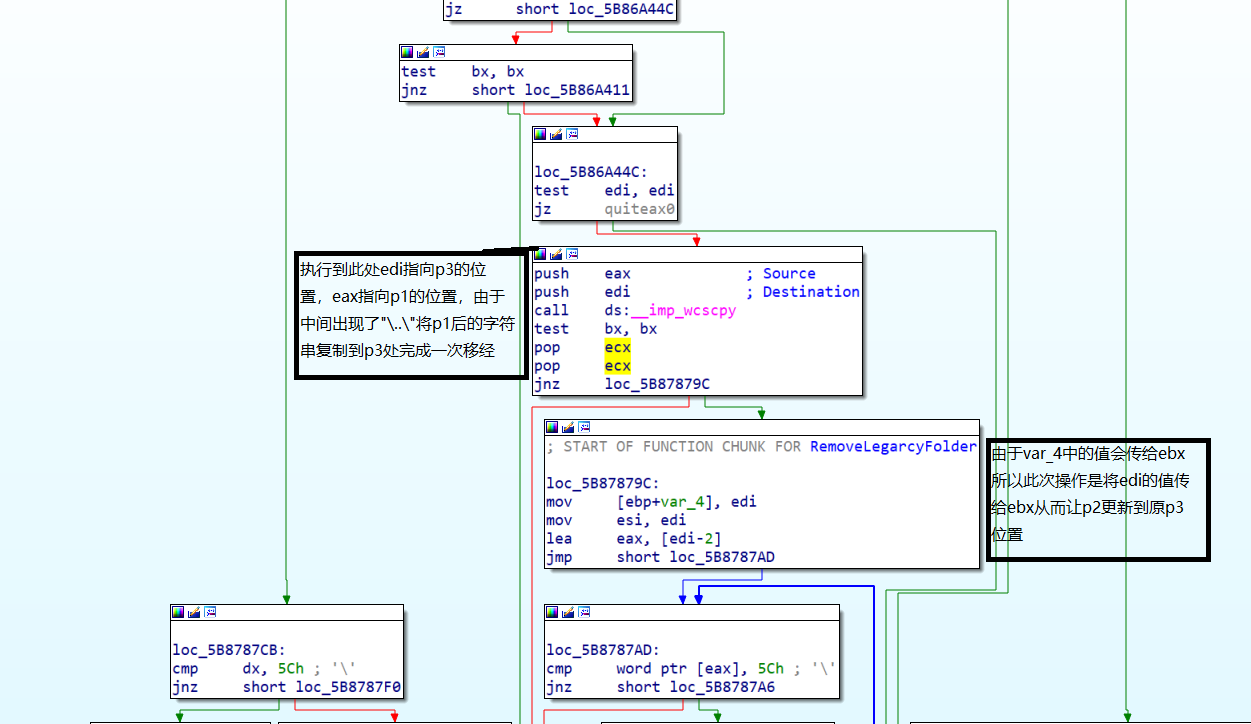
****

**图3.3.8 分析8**

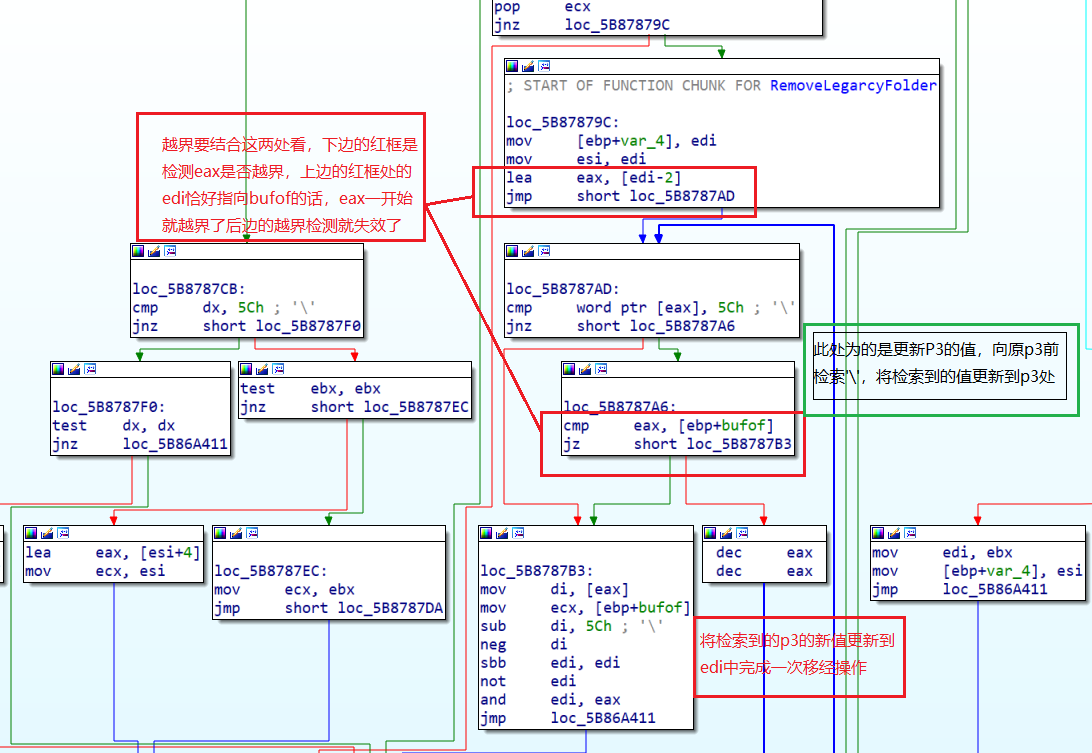
****

****

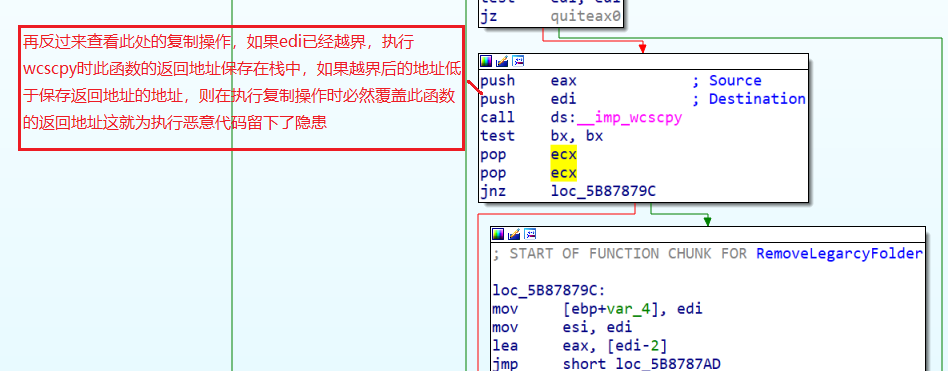
**图3.3.10 分析10**

****

**图3.3.11 分析11**

****

**图3.3.12 分析12**

****

**图3.3.13 分析13**

至此整个漏洞的静态分析关于漏洞方面的就结束了。但现在要考虑一个问题，如何确保越界后的空间内一定存在’\’呢？即便存在如何确保此’\’的位置到返回地址的界限不超过恶意代码的最大长度呢？这看起来好像无解，此漏洞看起来很难利用，或利用起来有较大随机性。

反观此漏洞函数的执行过程可以发现在执行RemoveLegacyFolder前还执行了checkDosPathType函数，执行checkDosPathType函数的时候一般情况下会用到栈空间用来保存运算的中间结果，假如传入的参数也就是bufof中的字符串，使得其中的运算恰好产生0x005c（‘\’）就可以很好的利用这个产生的参数。这时候就要查看checkDosPathType使得经过修饰后的字符串可以产生0x005c。

（注：此处的思路，我首先用kali的msf框架下的ms08-067漏洞的exploit攻击靶机，然后在靶机中用OD attach到被攻击的服务上，在此漏洞函数处下断点单步执行，当执行checkDosPathType后产生了0x005c）

接下来分析checkDosPathType（由于只是为了产生0x005c就舍弃分析的细节）

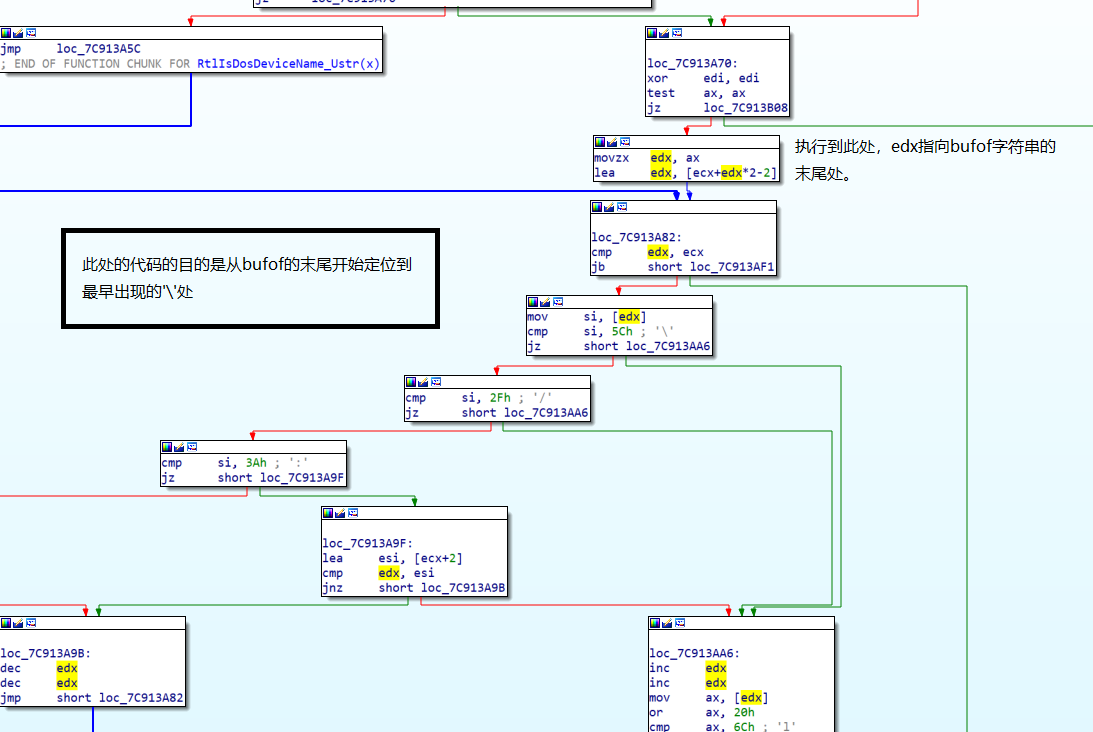
首先先给出一个结论，当bufof最后一个’\’后的字符恰好’a’、’l’、’p’中的任意一个，并且’\’后的字节数位90字节时会产生0x005c在栈中。

结构如图所示:



**图3.3.14 产生0x005c的bufof的结构图**

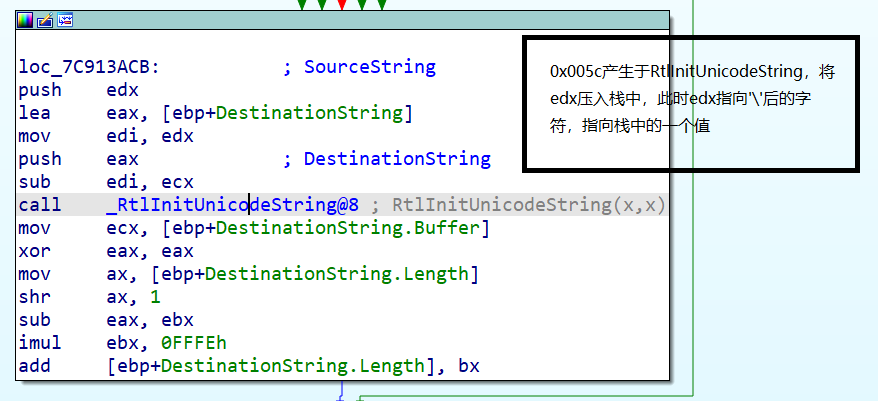
0x005c的产生位于checkDosPathType中的RtlIsDosDeviceName\_U的外部函数中，传入其的参数为bufof，此外部函数位于ntdll.dll中位于system32下，用IDA pro打开此Dll文件，并定位到RtlIsDosDeviceName处。0x005c的产生于RtlIsDosDeviceName\_Ustr中。结合动态分析定位到此函数中。

****

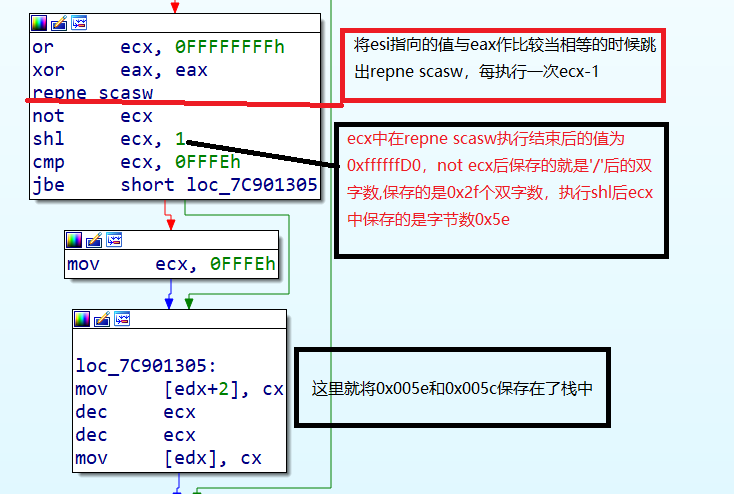
**图3.3.15 从bufof的末尾开始定位到最早出现的’\’**

****

**图3.3.16 可以产生0x005c的‘\’后的字符**

****

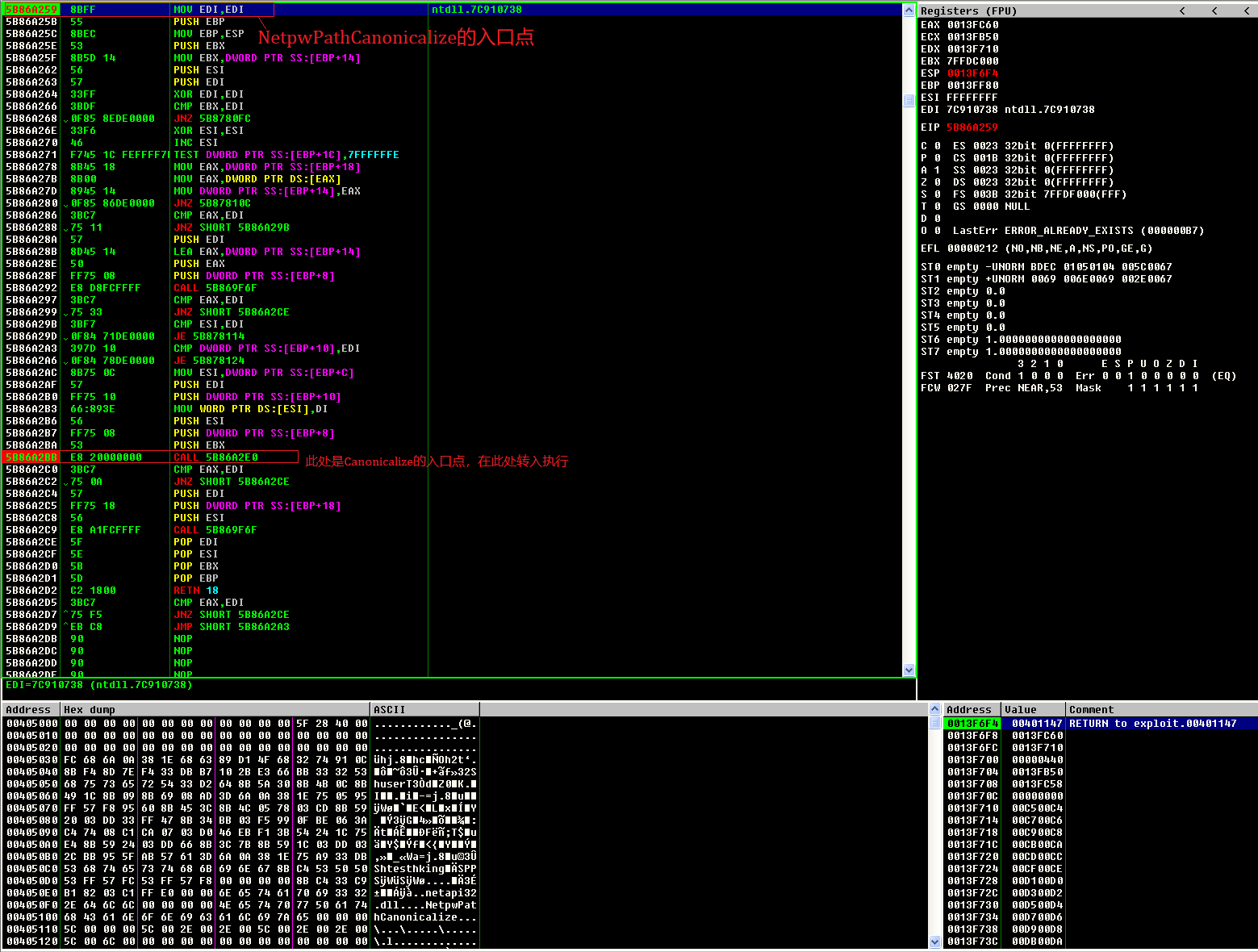
**图3.3.17 产生0x005c的函数**

****

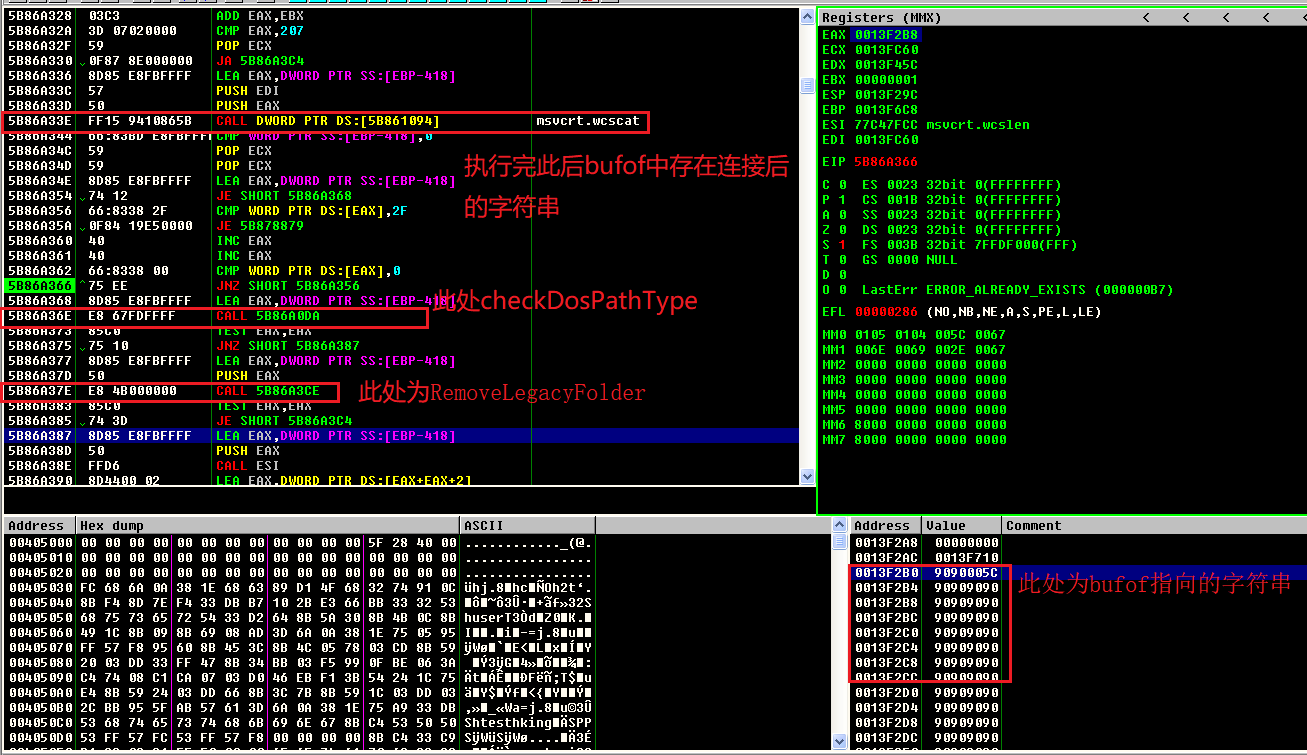
**图3.3.18 0x005c的诞生**

#### 3. 动态分析

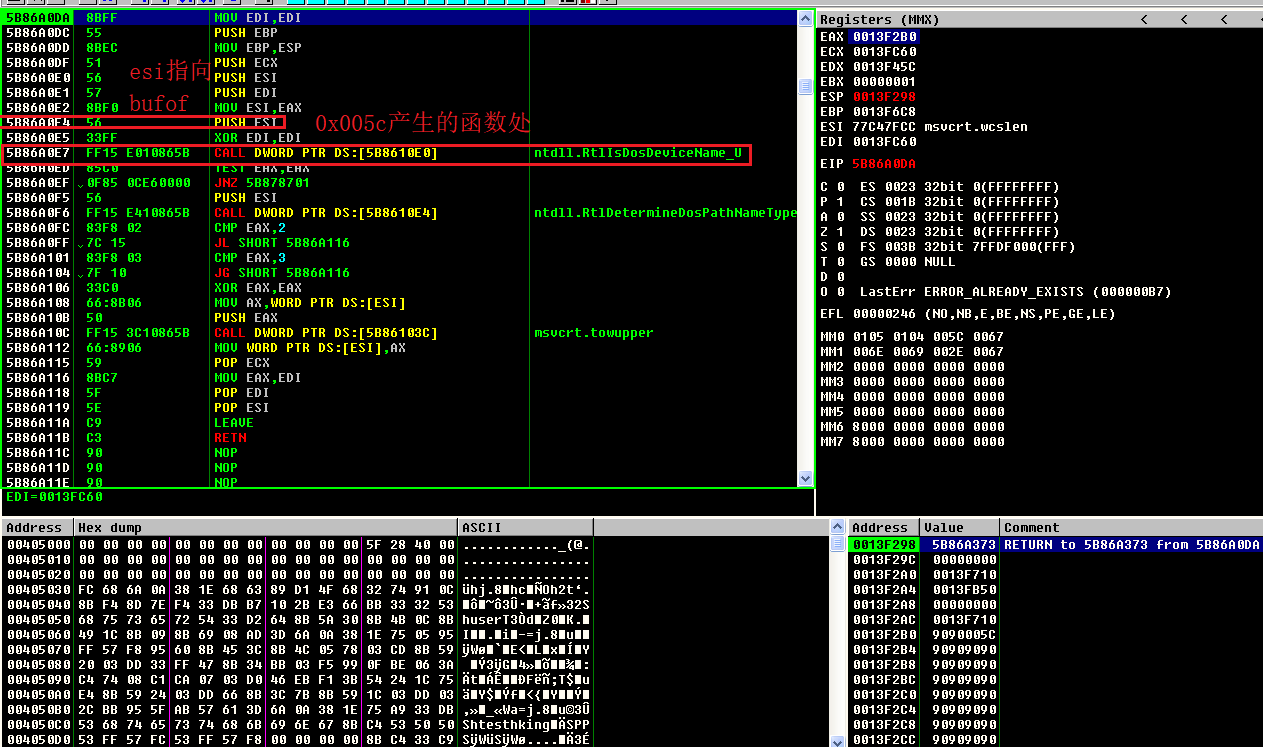
（此次的动态分析着重漏洞本身，因此使用关闭DEP，并且本地运行的测试代码，代码保存在windows\local\opendep\exploit.c）



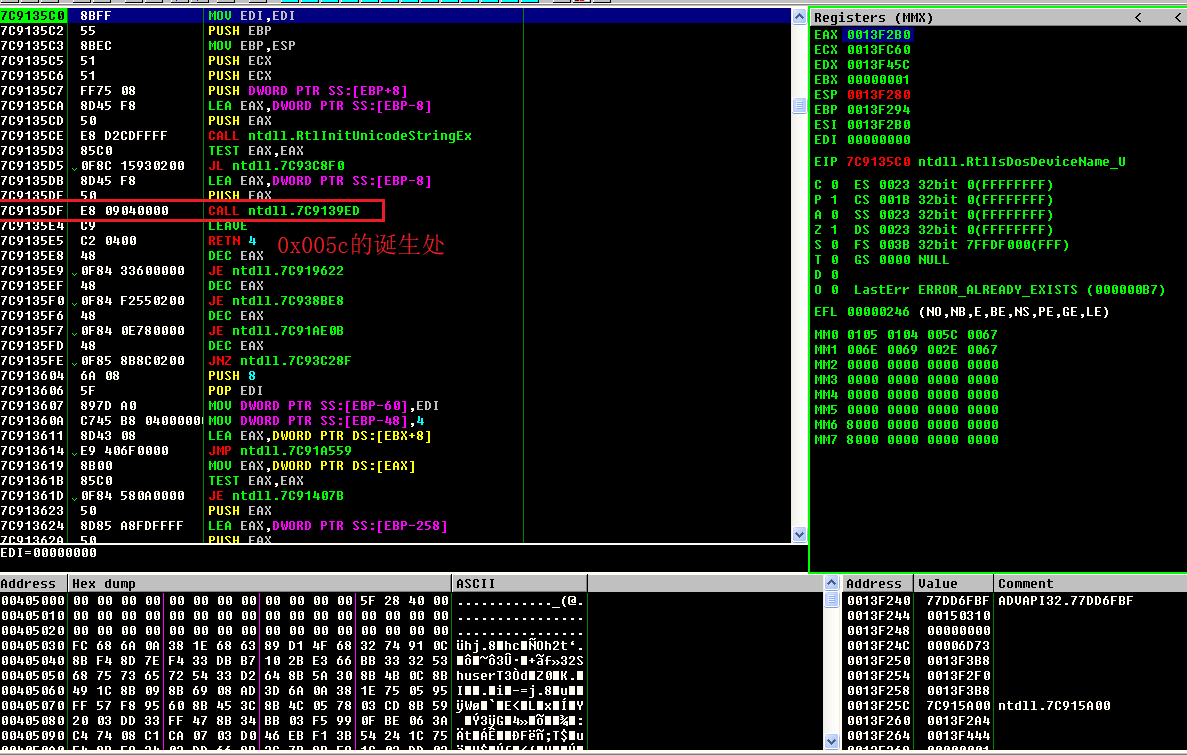
**图3.4.1 函数入口点**



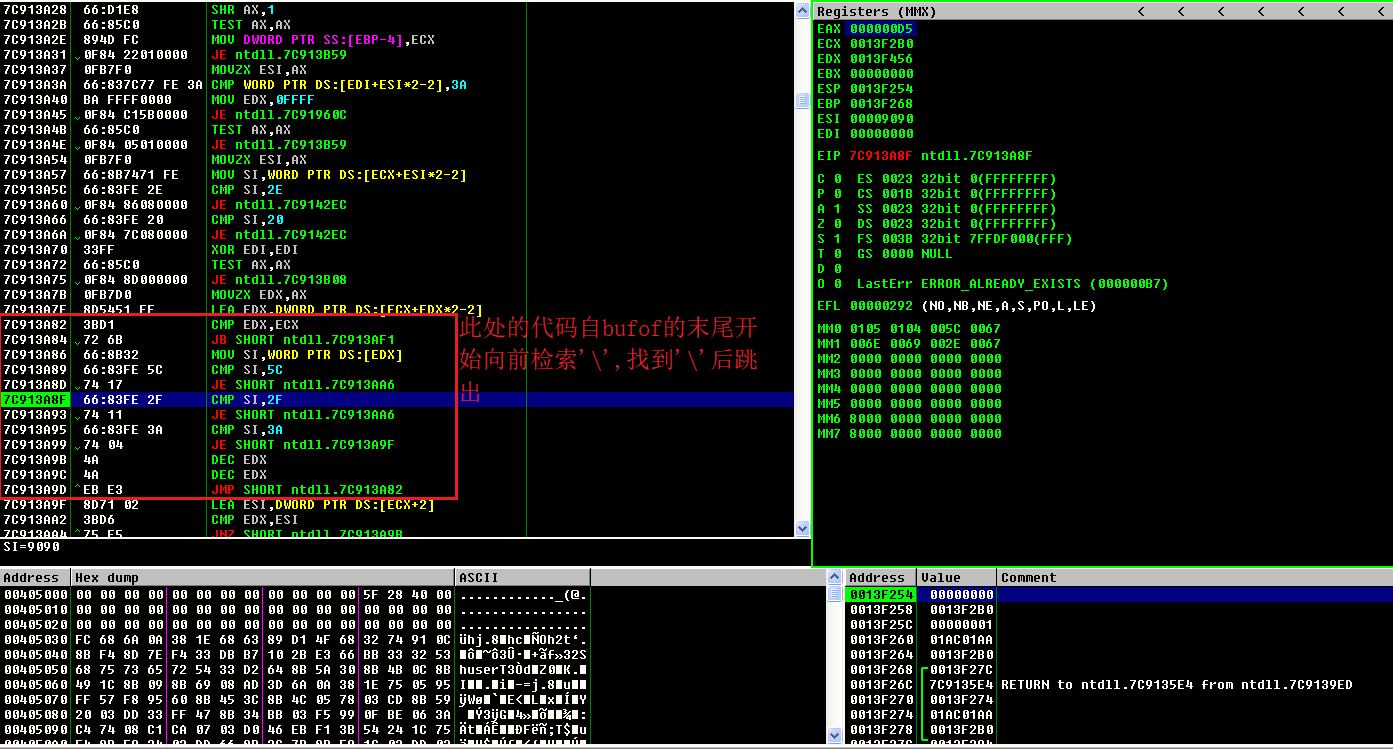
**图3.4.2 0x005c产生和移经操作**



**图3.4.3 产生0x005c的函数**



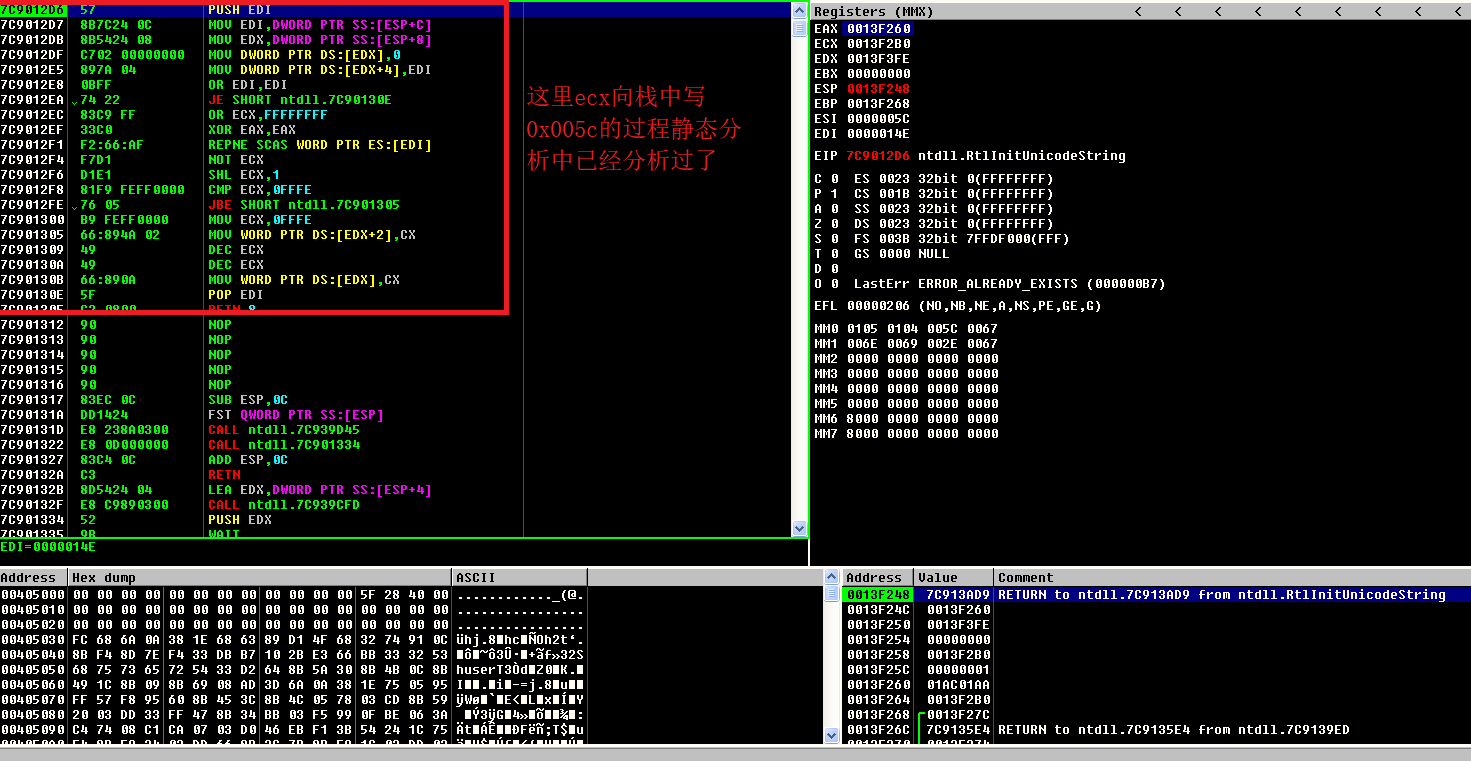
**图3.4.4 产生0x005c的函数**



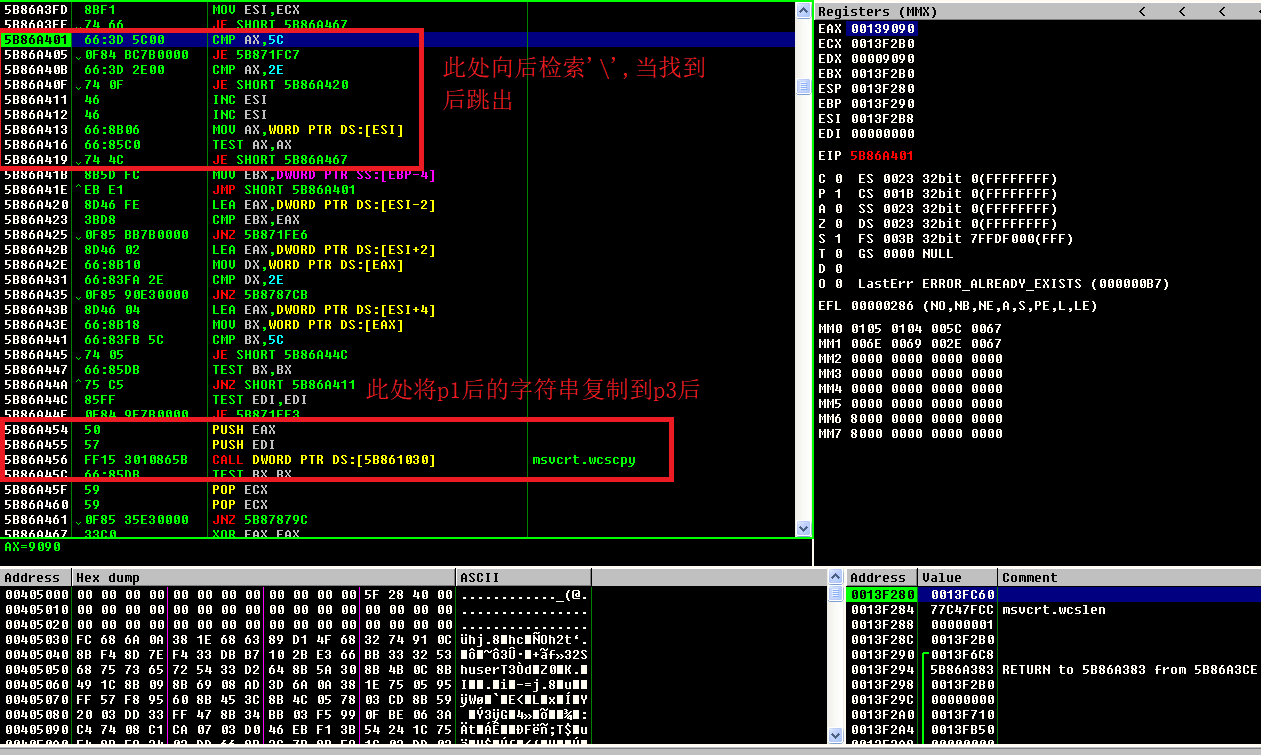
**图3.4.5 自bufof末尾向前检索’\’**



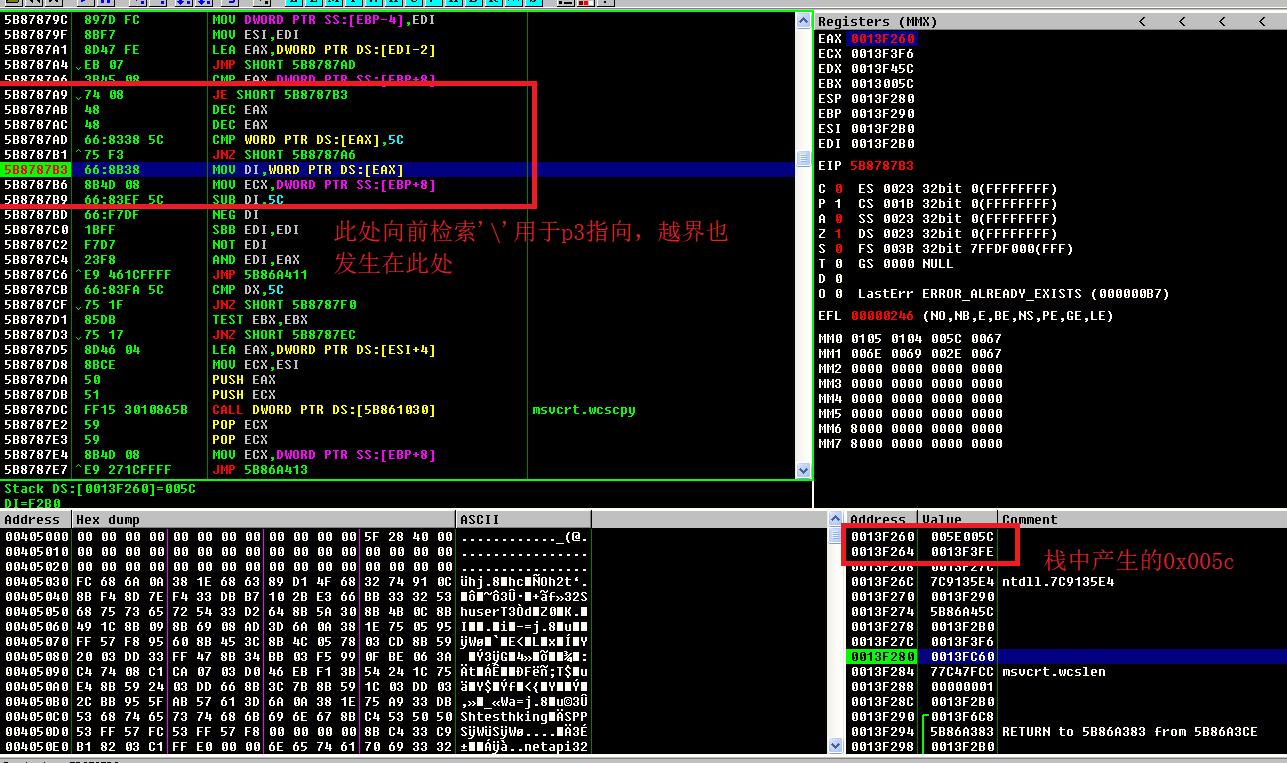
**图3.4.6 在’\’后写a\l\p\n的原因**



**图3.4.7 0x005c写入栈的过程**



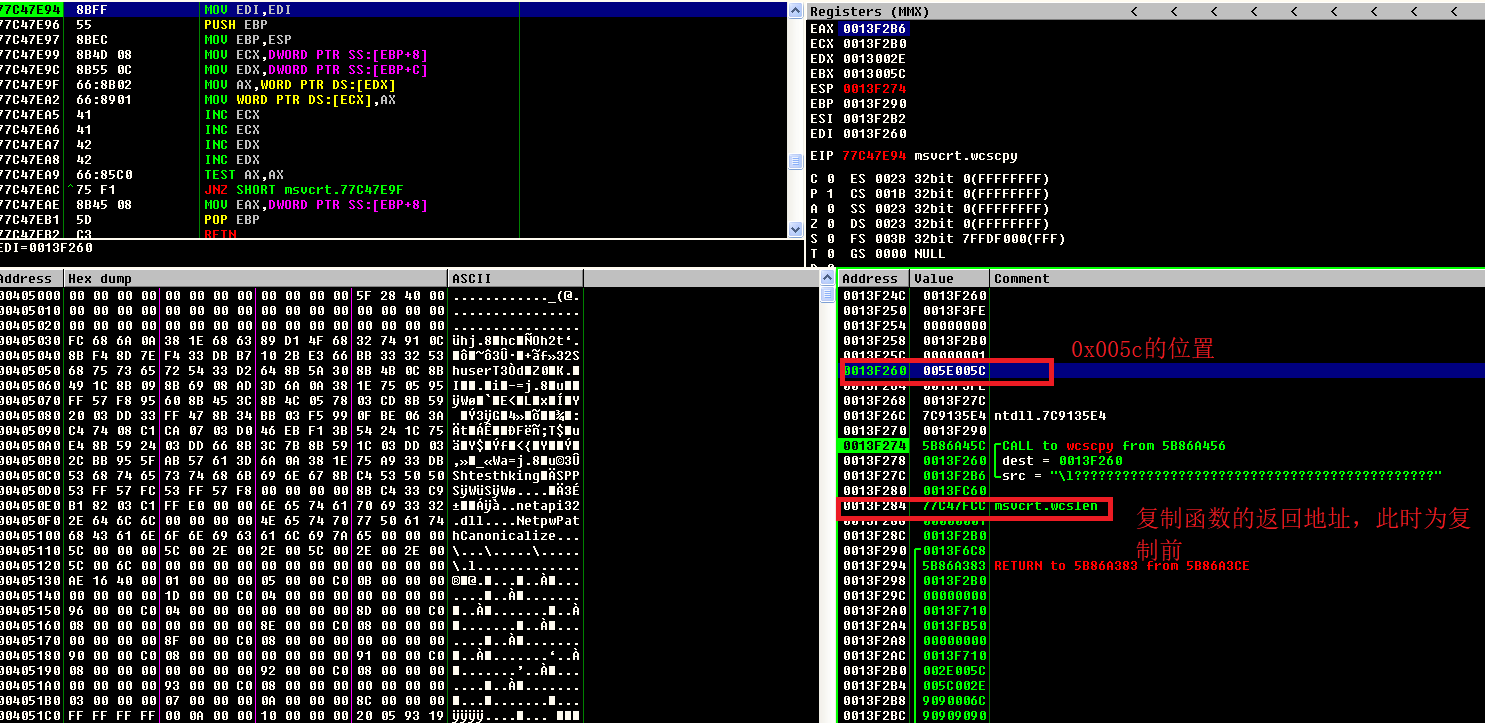
**图3.4.8 产生0x005c的函数**



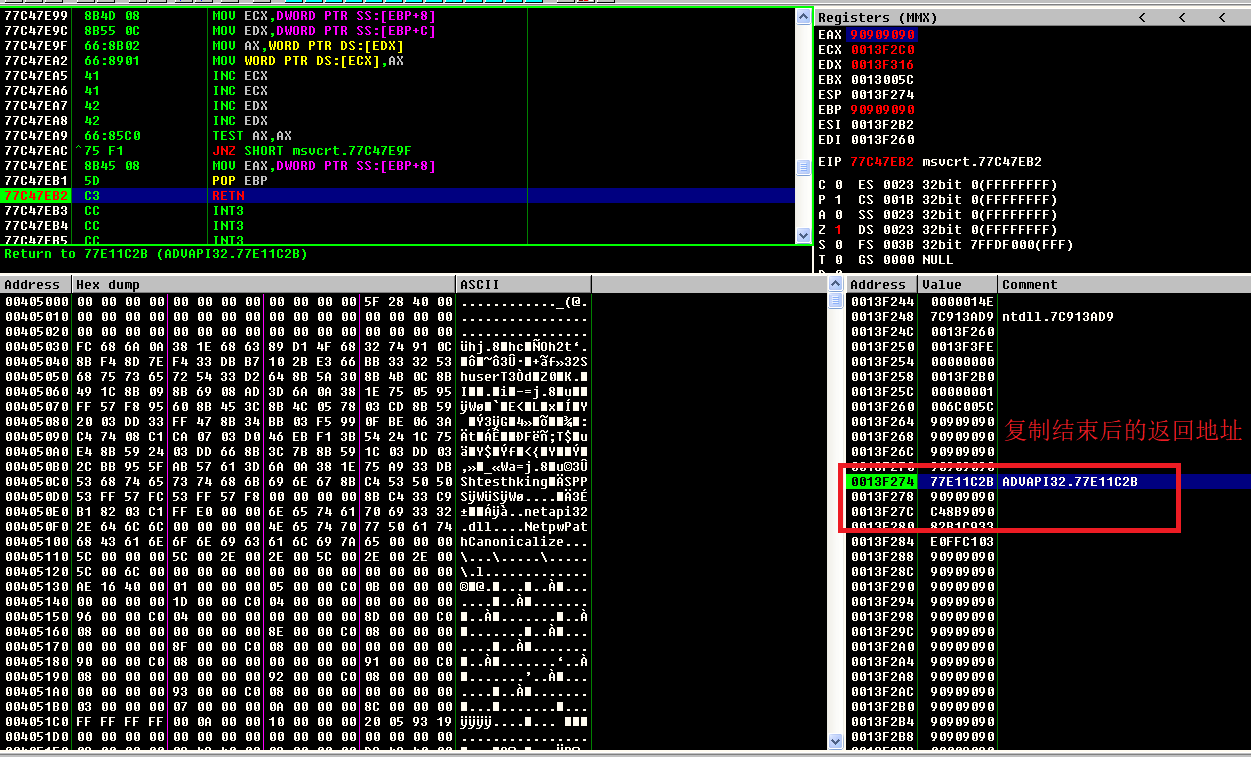
**图3.4.9 p3越界处**



**图3.4.10 返回地址覆盖处**

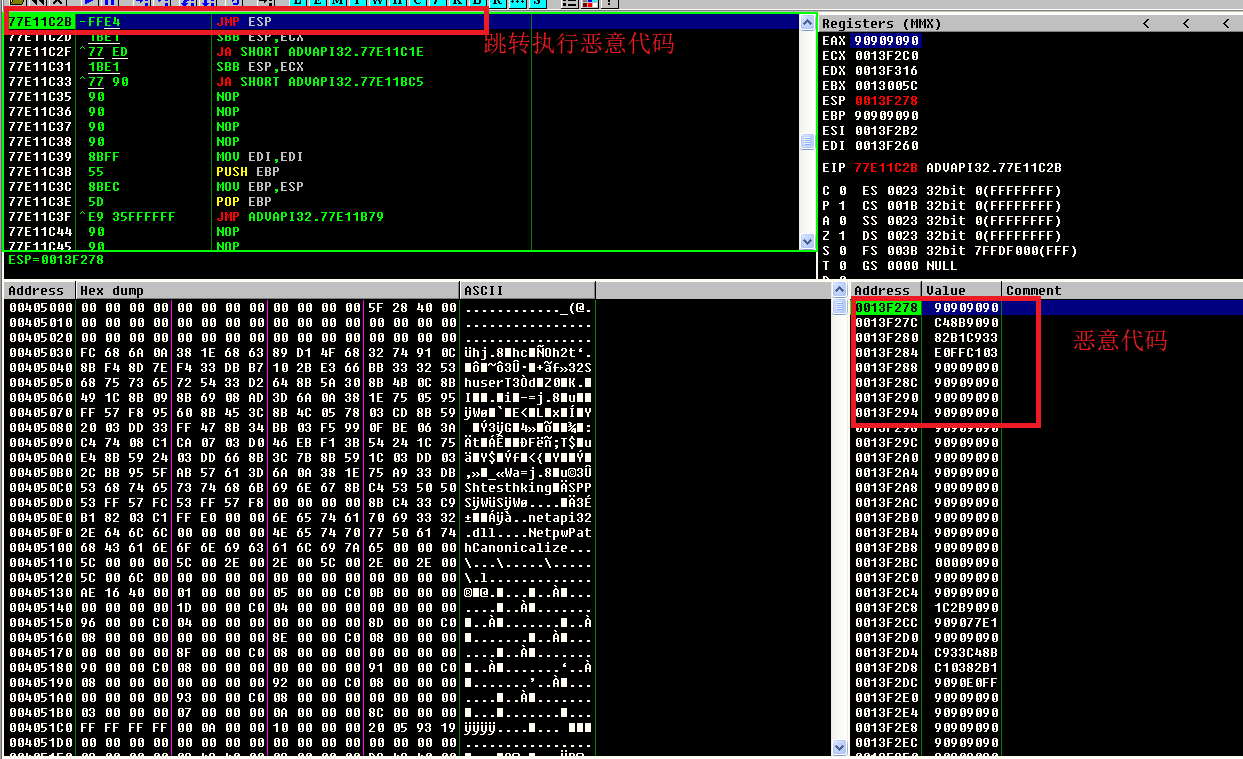


**图3.4.11 复制前的返回地址**



**图3.4.12 复制后的返回地址指向恶意代码的跳转地址**

此时esp指向恶意代码在内存中的位置，寻找一条指令 jmp esp即可完成恶意代码的执行。

****

**图3.4.13 跳转恶意代码执行**

至此动态分析结束

## 四、参考文献

0day安全：软件漏洞分析