# MS06-040漏洞分析(Windows XP sp0)

## 一、漏洞信息

### 1. 漏洞简述

* 漏洞名称：MS06-040
* 漏洞编号：CVE-2006-3439
* 漏洞类型：栈溢出
* 漏洞影响：远程代码执行
* CVSS评分：（以CVSS 3.0为准（一般分数较高），或者2者都备注）
* 利用难度：Medium
* 基础权限：不需要

### 2. 组件概述

Svchost是从动态链接库（DLL）中运行服务的通用主机进程名称。使用svchost -k netsvcs创建服务会处理由neiapi32.dll导出的NetpwPathCanonicalize函数的远程过程调用，此栈溢出漏洞由此函数引起。

### 3. 漏洞利用

漏洞的利用分为本地和远程两种，本地利用用于分析此导出函数存在的漏洞，在远程靠此漏洞进行恶意代码执行，进而控制存在此漏洞的主机。

### 4. 漏洞影响

此例中给出的漏洞利用代码影响的是Windows xp sp0。

### 5. 解决方案

下载安装官方给出的安全补丁KB921883。

## 二、漏洞复现

### 1. 环境搭建（分为本地利用和远程攻击）

（1）本地利用

* 测试平台：Windows xp professional （chinese）
* 编译器：VC6.0

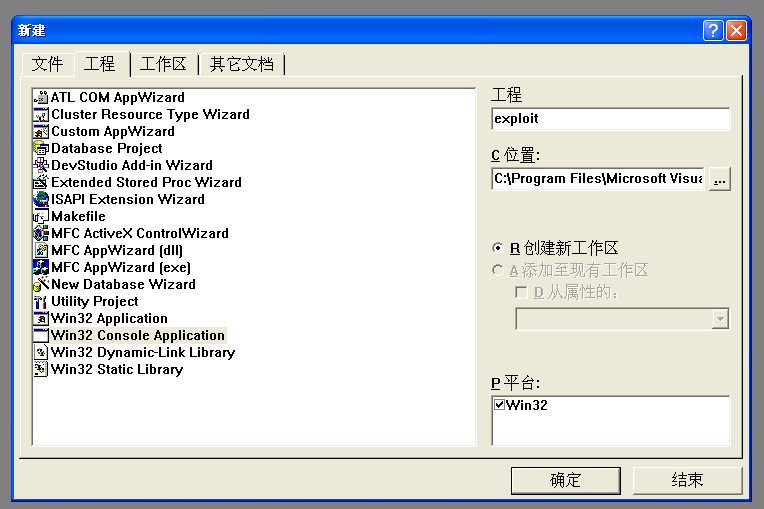
（远程攻击）

* 靶机平台：Windows xp professional（chinese）
* 攻击机平台：Windows xp professional sp3（chinese）
* 攻击机编译器：visual studio 2008

### 2. 复现过程

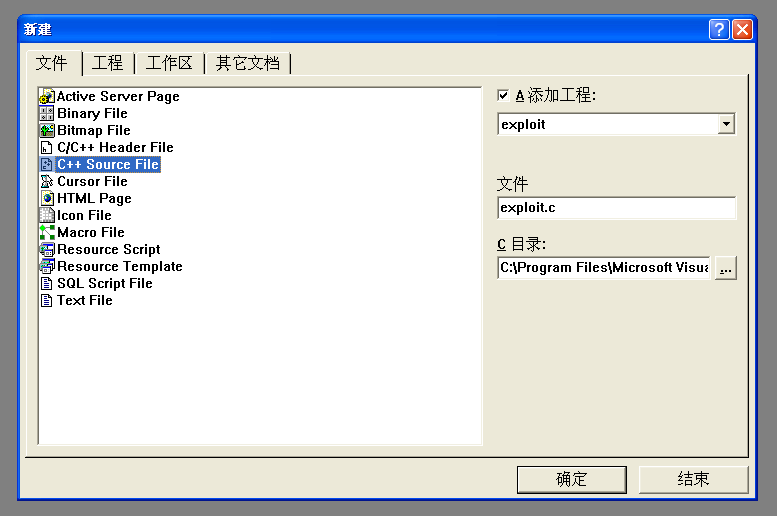
（本地利用）

1. 在测试平台上安装并启动VC6.0，新建项目win32控制台项目，并命名（可自行选择exploit）



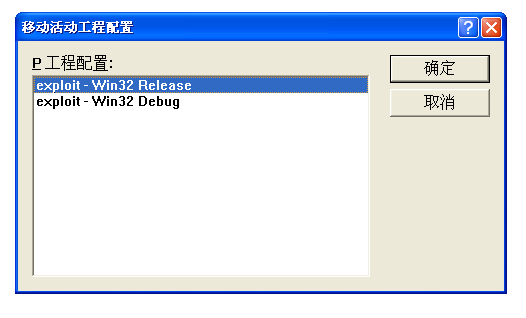
**图2.2.1 新建win32控制台项目**

1. 在新建的项目中添加.c文件并命名（exploit.c）萨乌丁大王的那位对外接口五年的我为了扩大那我

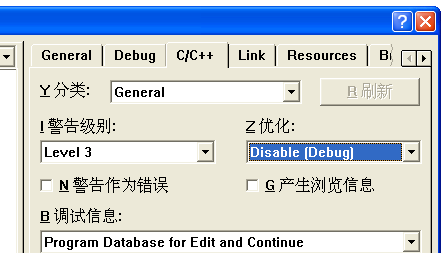


**图2.2.2 新建exploit.c文件**

1. 将windows/local/exploit.c复制到新建的源文件中，并配置VC6.0（关闭优化，输出编译文件为release版本）



**图2.2.3 输出编译版本为release版本**

****

**图2.2.4 在工程->设置->C/C++中关闭优化**

编译并运行所复制的代码，即可运行shellcode（一个弹窗）

****

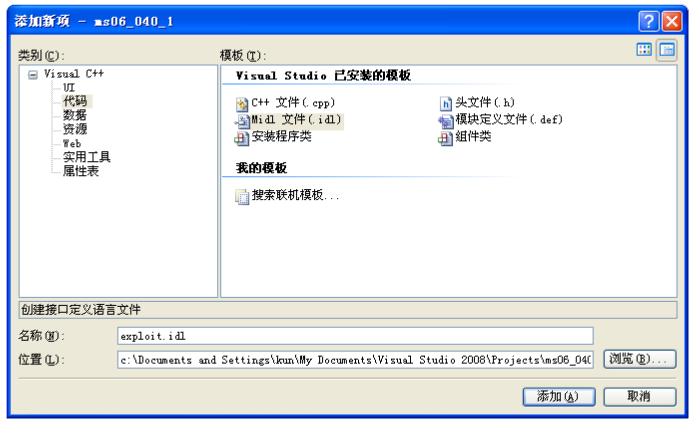
**图2.2.5 运行shellcode效果**

至此本地利用结束。

（远程攻击）

1. 在攻击机上安装并运行vs2008，新建项目（项目名称自定）在新建的额

项目中添加.idl文件（名称自定）将windows/rpcexploit/exploit.idl复制到新建的文件中。然后新建.acf文件（.acf文件并没有模板可以修改.cpp文件为.acf文件，名称自定）将windows/rpcexploit/exploit.acf文件复制到新建的文件中编译运行。

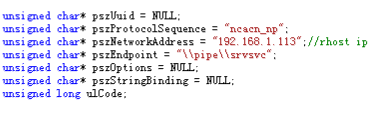


**图2.2.6 新建exploit.idl文件（acf文件类似）**

编译后在创建的idl文件相同的目录下会生成三个文件（名称依赖于项目名称）：exploit\_c.c、exploit\_s.c、exploit\_h.h；在此例中只用到了exploit\_c.c和exploit\_h.h将exploit\_c.c文件添加到源文件并将exploit\_h.h添加到头文件，之后在源文件中新建exploit.c文件并将windows/rpcexploit/exploit.c文件复制到新建文件中。

1. 查看靶机地址（注意务必使攻击机与靶机处于同一局域网下，并且攻击

机能够ping通靶机）修改exploit.c中的靶机IP地址为查到的IP地址。



**图2.2.7 修改靶机地址**

1. 在靶机的cmd下运行net share查看共享的管道，如果未开启输入Y开启，在攻击机的cmd下运行net use \\靶机IP\IPC$ “” /user:””将攻击机与靶机之间建立空连接（这一步主要目的是攻击机与靶机之间可以进行SMB通信）。



**图2.2.8 开启共享管道**

1. 编译运行攻击代码之后调出cmd在cmd上输入telnet 靶机IP 6666进行远程控制台连接，即可得到靶机的shell。



**图2.2.9 得到靶机shell**

## 三、漏洞分析

### 1. 基本信息

* 漏洞文件：netapi32.dll
* 漏洞函数：NetpwPathCanonicalize

### 2. 背景知识

（1）未开启GS守护下栈溢出的利用。

（2）Window上rpc编程。

（3）shellcode的编写。

（4）ollydbg以及ida pro的使用。

### 3. 详细分析

#### 1. 基础分析

NetpwPathCanonicalize函数有6个参数分别为

{

U16 \* path;

U16 \* can\_path;

int maxbuf;

U16 \* prfix;

int \*\* pathtype;

int pathflags;

}

在连续两次进行NetpwPathCanonicalize调用的时候，在第一次中prefix长度0x100字节、path长度为0x300字节，can\_path长度0x440字节，其中prefix与path均用0x90填充。在第二次调用时长度不变，但是prefix为空串（prefix中存储的均为0）path还是0x90，在第二次函数调用时出现崩溃通过动态分析可以得知返回地址后的一大片区域被0x90覆盖，推测得知发生了栈溢出。

#### 2. 静态分析

由LoadPE确定NetpwPathCanonicalize的入口地址为0x71ba4244（LoadPE可以从网上下载），在进行静态分析的时候可以用IDA pro（32位版本）打开存在漏洞的动态链接库netapi32.dll在导出函数中确定NetpwPathCanonicalize的位置。

##### (1). NetpwPathCanonicalize执行流程

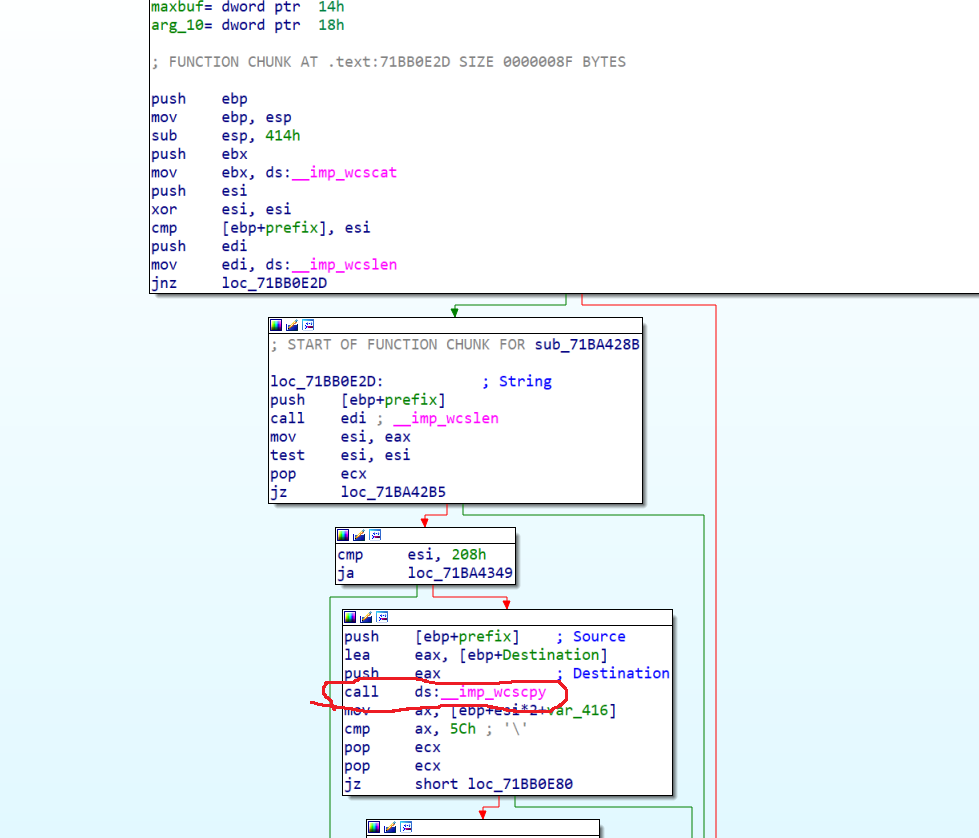


**图3.3.1 NetpwPathCanonicalize执行流程图**

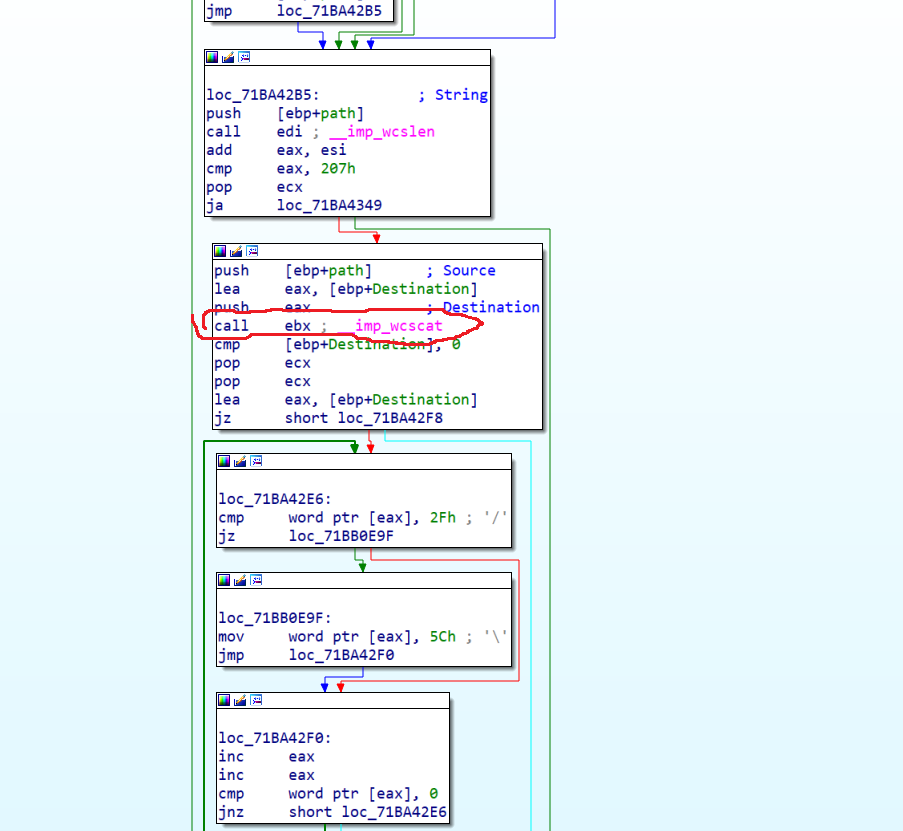
NetpwPathCanonicalize的大致流程为对prefix和path分别进行格式修正，通过检验后连同can\_path输入到sub\_71BA428B中进行prefix与path的合并，合并后的值复制到can\_path中，最后对can\_path进行格式修正并退出。函数sub\_71BA428B是漏洞点，其参数为prefix、path、can\_path、以及maxbuf。

##### (2). 漏洞函数分析（sub\_71BA428B）（流程图不想画了）

首先开辟了0x414字节的栈区BUF用于保存连接的字符串，得到prefix指向的字符串的长度，如果长度不为0并且Unicode编码长度不大于0x207，那么执行wcscpy将prefix指向的字符串以Unicode的方式保存在BUF中，并将prefix指向的字符串Unicode编码长度保存在esi中，然后在BUF字符串尾添加’\’,并计算path字符串的Unicode编码长度将此长度与prefix的长度相加不大于0x207将path指向的字符串用wcscat连接到BUF的末尾。如果prefix的长度为0，那么计算path的长度，若长度小于0x207那么将path指向的字符串用wcscat添加到BUF所指向区域的末尾。问题就出在这里，假如BUF中原本就有一定长度的字符串，则在连接path的时候依旧按照0x207进行连接，这样必定导致栈溢出。



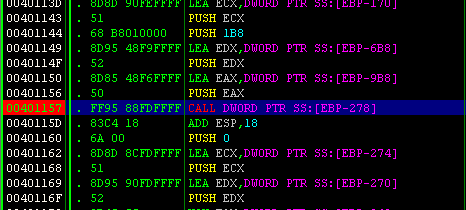
**图3.3.2 标红为prefix到BUF的复制**

****

**图3.3.3 标红为path到BUf的连接（prefix为0和不为0均执行此处代码，为0则直接执行，此处是栈溢出的关键点）**

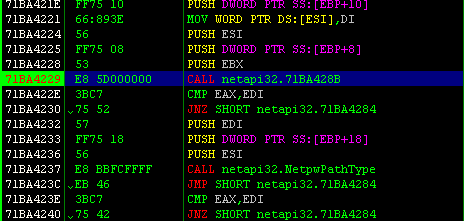
#### 3. 动态分析（分为本地和远程两种）

（1）本地利用动态分析，将windows/local/exploit.c的文件编译链接后生成的可执行文件用ollydbg打开。在第一次调用NetpwPathCanonicalize的位置下断点。



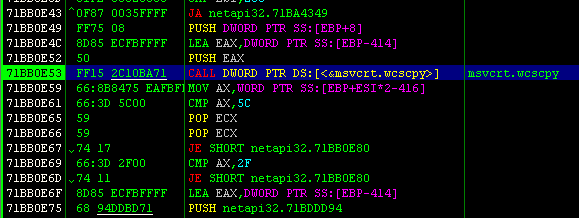
**图3.3.4 下断点处**

直接执行到此处然后进入，在静态分析中可以得知在NetpwPathCanonicalize开始只是做了一些格式矫正。真正的漏洞点在末尾的sub\_71BA428B，则在此函数处下断点执行至此进入。

****

**图3.3.5 下断点处**

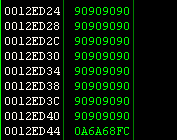
在第一次调用当中给prefix和path指向的字符串的长度均符合要求所以可以正常执行首先是prefix复制到栈区（prefix中也包含了shellcode）在prefix复制到BUF的函数处下断点。查看运行前后栈区的内容

****

**图3.3.6 下断点处**

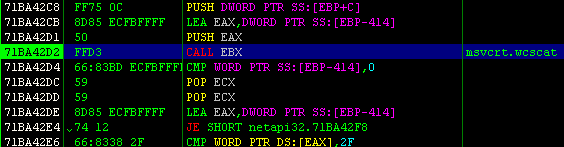
****

**图3.3.7 执行前栈区**

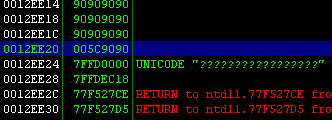
****

**图3.3.8 执行后栈区**

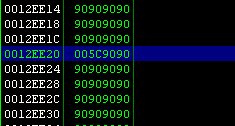
至此prefix复制到了BUF指向的栈区中，之后在将path连接到BUF末尾的函数处下断点查看栈区的内容。

****

**图3.3.10 下断点处**

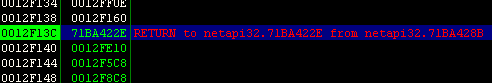
****

**图3.3.11 执行前的栈区**

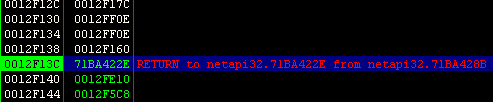
****

**图3.3.12 执行后的栈区**

至此查看一下返回地址，依旧指向调用sub\_71BA428B的下一指令的地址处。

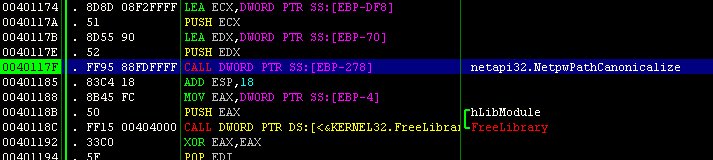
****

**图3.3.13 刚进入sub\_71BA428B时的返回地址**

****

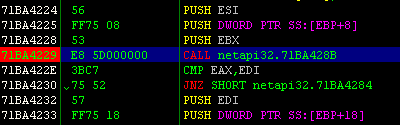
**图3.3.14 退出sub\_71BA428B时的返回地址**

可以看到返回地址并未改变返回到主函数中进行第二次NetpwPathCanonicalize调用。

****

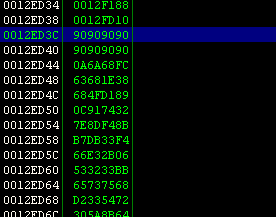
**图3.3.15 第二次执行NetpwPathCanonicalize的位置**

进入此函数，同第一次一样，在NetpwPathCanonicalize该开始做的只是格式矫正，还在最后的漏洞函数处下断点。

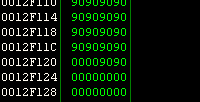
****

**图3.3.16 下断点处**

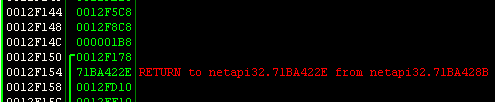
在下断点处进入执行，在此次执行中由于prefix指向的为空串所以会直接执行wcscat将path连接到BUF后，在执行连接函数前先查看栈区内容，并查看返回地址。

****

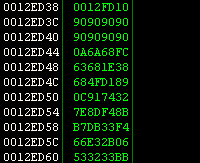
**图3.3.17 执行前的栈区开头**

****

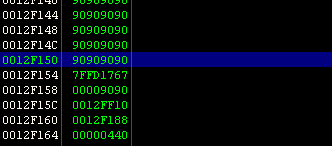
**图3.3.18 执行前的栈区尾**

****

**图3.3.19 执行前的返回地址**

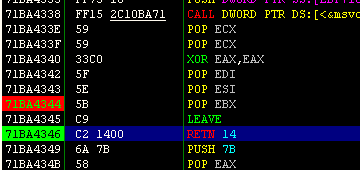
****

**图3.3.20 执行后的栈区头**

****

**图3.3.21 执行后的栈区尾，以及函数返回地址**

可以看到在执行连接函数前后返回地址发生了变化（如何确定返回地址呢？ebp+4的位置即为函数返回地址）执行到函数返回处RET 14处。

****

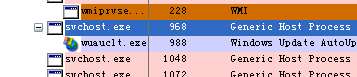
**图3.3.22 函数返回点**

至此esp指向的位置即为要返回的地址，此地址已经被覆盖为构造的一个地址，通过查看寄存器可以看到ecx指向的时BUF开头的位置，由第一次的prefix中保存的是shellcode，可以得知ecx指向的位置保存的是shellcode，所以只要在内存中找到一条jmp ecx的指令的位置来替代返回地址就可以跳转到shellcode处执行。这里借助ollydbg中的ollyfindaddr插件进行jmp ecx的检索，得到的地址用来替代返回地址。

继续执行就可以跳转到shellcode中执行。至此本地分析结束

（2）远程攻击动态分析

此次分析同本地分析类似，不同处在于需要在靶机中用ollydbg attach到被攻击点（也就是被攻击的svchost服务）。在进行attach时会发现有许多svchost，这里可以使用ProcessExplorer确定存在漏洞svchost服务的pid。



**图3.3.23 存在漏洞点的svchost服务**

## 四、参考文献

（太多了不想写了）

附：Windows/flowchart.vsdx是由visio画的NetpwPathCanonicalize的流程图。