
# 渗透防火墙的Shellcode技术

作者：san@xfocus.org

# 内容摘要

- 1. 远程shellcode的几种实现方式
- 2. 复用当前连接技术的一些问题及优势
- 3. Win32平台的具体实现
- 4. Linux x86平台的具体实现
- 5. AIX PowerPC平台的具体实现
- 6. 特别感谢
- 7. 参考文档

# 实现方式简介

- 1. 远程shellcode的几种实现方式
  - 1.1 监听端口
    - 1.1.1 监听新的端口
    - 1.1.2 重新使用原端口
    - 1.1.2.1 端口复用
    - 1.1.2.2 重新绑定
  - 1.2 反向连接
  - 1.3 复用当前连接的SOCKET
    - 1.3.1 IIS的ECB结构
    - 1.3.2 getpeername
    - 1.3.3 fcntl设置socket状态
    - 1.3.4 ioctl(Linux/Unix)和ioctlsocket(Win32)
    - 1.3.5 使用OOB特性
    - 1.3.6 Hook系统的recv调用

- 2. 复用当前连接技术的一些问题及优势
  - 2.1 绑定shell
    - Unix下可以直接把SOCKET作为“/bin/sh”的输入输出句柄。
    - 在Win32下，socket()函数隐式指定了重叠标志，它创建的SOCKET是重叠套接字(overlapped socket)，不能直接将cmd.exe的stdin、stdout、stderr转向到套接字上，只能用管道(pipe)来与cmd.exe进程传输数据。winsock推荐使用重叠套接字，所以实际使用尽可能用管道。
    - WSASocket()创建的SOCKET默认是非重叠套接字，可以直接将cmd.exe的stdin、stdout、stderr转向到套接字上。

## – 2.2 多线程环境搜索SOCKET

```
-=-=-=-=-=- start sample code =-=-=-=-=-=-=
s = WSASocket(2,1,...)
bind(s,..)
listen(s,...)
s2 = accept(s,....)
-=-=-=-=-=- end sample code =-=-=-=-=-=-
```

- 当s处于accept状态时，任何对s操作的网络函数都会处于等待状态，直到有连接建立。
- 先用WaitForSingleObjectEx处理句柄，当s处于accept状态时会返回WAIT_TIMEOUT，可用的句柄返回WAIT_OBJECT_0。然后再用ioctlsocket/recv处理判断是否当前连接的socket。

## – 2.3 优势

- 复用当前连接的技术相对较隐蔽，而且对于Win32的用管道绑定cmd.exe后，和服务器交互的数据可以用xor的办法进行编码，进一步躲避IDS的检测。
- 单单查找SOCKET的shellcode可以写的相对比较短，在找到SOCKET后，可以再继续接收一段功能更复杂的shellcode到缓冲区，然后跳入执行。对于该后续shellcode就没有任何字符和长度的限制。
- 甚至接收一个dll文件，实现更复杂的功能。

- 3. Win32平台的具体实现
  - 3.1 端口复用具体实现
    - 端口复用shellcode要求服务重新绑定在0.0.0.0地址，而且不能使用SO_EXCLUSIVEADDRUSE选项。
    - 客户端攻击的时候需要把服务端的具体IP和端口写入shellcode里面。
    - Shellcode里执行：
    - setsockopt(s, 0xFFFF, 4, &d, 4);
    - bind(s, &sockaddr, 0x10);
    - 用netstat -na在服务端查看可以看到同一个端口有0.0.0.0和具体IP两个绑定着。
    - 如果服务端在NAT环境里，可能会存在问题。

# Win32平台的具体实现

## – 3.2 重新绑定原端口的实现

- CreateProcess()创建一个suspend模式的进程。
- GetThreadContext()来获得该线程的上下文结构和寄存器信息。
- 用VirtualAllocEx()在该进程里分配内存。
- 把shellcode指令用WriteProcessMemory()来写入该进程刚才分配的空间。
- SetThreadContext()把GetThreadContext()获得的EIP修改指向VirtualAllocEx()分配的内存地址。
- ResumeThread()恢复suspend模式的进程。
- TerminateProcess(-1, 0)终止当前进程。
- 循环绑定原来端口。

# Win32平台的具体实现

- 3.3 getpeername查找socket
  - 客户端在发送攻击串之前用getsockname函数获得套接字本地信息，把相应信息写入shellcode。
  - 服务端shellcode从1开始递增查找socket，并且用getpeername函数获得套接字远程信息。
  - 如果两个信息相符就认为找到socket，跳出递增循环，并且把shell绑定在这个socket上。
  - 有很大局限性，如果客户端在NAT网络环境里，客户端getsockname取得的套接字信息和服务端getpeername取得的套接字信息不一定相符，导致查找socket失败。

# Win32平台的具体实现

## – 3.4 字串匹配查找socket

- 客户端在发送完攻击数据包后，再发送几个字节的字符串，在服务端的shellcode对一个递增的socket值接收相应字节的字符串，然后匹配是否是当前连接的socket。这种方法避免了getpeername在NAT网络环境里的限制。
- 多线程环境下容易处理到处于accept下的socket，一般的网络函数都会进入等待状态，直到有连接建立。
- flier提到WaitForSingleObjectEx处理这种处于accept的socket会返回WAIT_TIMEOUT，可用的句柄返回WAIT_OBJECT_0。

– 3.4 字串匹配查找socket

- 查找流程如下：

```
while (1)
{
    i++;
    ret = WaitForSingleObjectEx(i, 10, 1);
    if (ret != 0) continue;
    ret = ioctlsocket(i, FIONREAD, &ul);
    if (ul != 4) continue;
    recv(i, buff, 4, 0);
    if( *(DWORD *)buff == 'Xc0n') goto shell;
}
```

- bkbll测试发现socket()函数创建的句柄在accept用户后有getsockname操作，那么后续WaitForSingleObjectEx返回WAIT_TIMEOUT (0x102)。

# Win32平台的具体实现

– 3.5 Hook系统的recv调用

- 用VirtualProtect设置真实recv函数地址开始的5个字节为可写。
- 把真实recv开始的指令改为跳转到新recv函数。
- 在新的recv函数里先把这5个字节指令改回去。
- 调用真实recv来执行系统本来的操作。
- 再把真实recv函数地址开始的5个字节改为跳转新recv的指令。
- 比较接收的数据是否是约定字串，如果是就绑定一个cmd.exe，否则跳到压栈的返回地址，继续系统原来的流程。
- 这种方法能绕过rpc之类机制，也能够搜索再次连接的socket。

# Win32平台的具体实现

## – 3.6 文件上传下载功能的实现

- 必须要客户端和shellcode做紧密配合。
- 上传文件需要客户端打开并读取文件发送给服务端，服务端的shellcode创建并写入该文件。
- 下载文件需要服务端的shellcode打开读取文件发送给客户端，客户端创建并写入该文件。
- 由于是非阻塞的连接，上传文件的时候，服务端的shellcode必须判断socket里是否还有数据可接收，如果没有就关闭句柄，执行后续流程。下载文件的时候，客户端必须判断socket里是否还有数据。
- select和ioctlsocket都可以实现这个功能。select的汇编实现相对复杂，ioctlsocket需要在发送缓冲块大于接收缓冲块的情况下使用。

## – 3.6 文件上传下载功能的实现

- 客户端和服务端shellcode可以使用一个约定的key对传输的数据做xor操作，由于用管道绑定cmd，那么交互的命令也是编码的，进一步增强隐蔽性，躲避IDS的检测。

- 4. Linux x86平台的具体实现
  - 4.1 fcntl设置socket状态
    - scz最早使用这种方法，基本思路如下：

```
while (1)
{
    i++;

    oldflags = fcntl(i, F_GETFL, 0);
    fcntl(i, F_SETFL, oldflags | O_NONBLOCK);
    read(i, buf, 4);
    fcntl( i, F_SETFL, oldflags );

    if (buf == 'Xc0n') goto shell;
}
```

– 4.2 利用OOB特性

- bkbll最先使用该技术。Berkeley套接口的实现OOB数据一般是不会被阻塞的，查找的流程大致如下：

```
while (1)
{
    i++;

    recv(i, buf, 1, 1);
    if (buf == 'I') goto shell;
}
```

- Unix/Linux该方法最是简单易行，而且有效。

– 4.3 利用ioctl函数的一些特性

- ioctl的FIONREAD可以判断句柄有多少数据可读，而且一般情况不会被阻塞。查找socket的流程大致如下：

```
while (1)
{
    i++;

    ioctl(i, FIONREAD, &ul);
    if (ul != 4) continue;
    read(i, buf, 4);
    if (buf == 'Xc0n') goto shell;
}
```

– 4.4 文件上传下载功能的实现

- 和Win32实现相似，只是Linux/Unix下似乎没有额外通过管道来绑定/bin/sh，所以shell里交互的数据无法编码处理。隐蔽性可能较差。

# AIX PowerPC下具体实现

- 5. AIX PowerPC平台的具体实现
  - 缓存机制
    - instruction cache
    - data cache

# AIX PowerPC下具体实现

– PowerPC自修改的代码按照如下的步骤：
  - 存储修改的指令。
  - 执行dcbst指令，强制包含有修改过的指令的高速缓存行进行存储。
  - 执行sync指令，确保dcbst完成。
  - 执行icbi指令，使将要存放修改后指令的指令高速缓存行无效。
  - 执行isync指令，清除所有指令的指令管道，那些指令在高速缓存行被设为无效之前可能早已被取走了。
  - 现在可以运行修改后的指令了。当取这个指令时会发生指令高速缓存失败，结果就会从存储器中取得修改后的指令。

# AIX PowerPC下具体实现

- 有些AIX是没有高速缓存管理指令。
  - 简单的解决方法是做完自修改后执行一个系统中断，那么后面就能正确执行自修改后的指令。
- 实现解码shellcode，为实现复杂shellcode做好基础。

# AIX PowerPC下具体实现

- 5.1 利用OOB特性的实现
  - 和Linux x86的实现类似
- 难点：
  - 各版本AIX的系统调用号都是不相同的，导致exploit不通用。
  - 本地exploit先用oslevel -r判断系统版本，然后把各不同系统调用号写入shellcode。
  - 远程exploit更加困难。如果服务端开放dtscpd服务，给该服务发送验证数据可以得到系统版本信息。

- 6. 特别感谢
  - eyas, scz, flier, tombkeeper, watercloud, Emmanuel, H D Moore, KF...

# 参考文档

- 7. 参考文档

  - [1] Win32 Assembly Components
  - http://www.lsd-pl.net/documents/winasm-1.0.1.pdf
  - http://www.lsd-pl.net/documents/winasm.ppt
  - http://lsd-pl.net/projects/winasm-1.1.tar.gz
  - [2] Win32 One-Way Shellcode
  - http://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-chong.pdf
  - [3] Understanding Windows Shellcode
  - http://www.hick.org/code/skape/papers/win32-shellcode.pdf
  - [4] http://www.0x557.org/release/ex_servu.c
  - [5] Serv-U FTPD 2.x/3.x/4.x/5.x "MDTM" Command Remote Exploit
  - http://www.cnhonker.com/index.php?module=releases&act=view&type=3&id=54
  - [6] 一段远程shellcode
  - http://bbs.nsfocus.net/index.php?act=ST&f=2&t=144419
  - [7] 利用OOB查找socket(更正by bkbll,sorry to scz)
  - http://www.cnhonker.com/index.php?module=articles&act=view&type=6&id=28
  - [8] The GNU assembler
  - ftp://ftp.gnu.org/gnu/Manuals/gas/text/as.txt
  - [9] Linux System Call Table
  - http://www.system-calls.com/sys_call_table.php
  - [10] UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes
  - http://www.lsd-pl.net/documents/asmcodes-1.0.2.pdf
  - http://www.lsd-pl.net/documents/asmcodes.ppt
  - http://www.lsd-pl.net/projects/asmcodes-1.0.2.tar.gz
  - [11] PowerPC 体系结构开发者指南
  - http://www-900.ibm.com/developerWorks/cn/linux/l-powarch/index.shtml
  - [12] UNIX网络编程（第一卷） W.Richard Stevens

# 结束

- 谢谢大家！