**当Client发送SYN包之后崩溃(SYN FLOOD攻击)**

Client发送SYN包给Server后崩溃，Server回给Client的SYN-ACK一直没收到Client的ACK确认，因此Server需要一个超时时间将这个连接断开，否则这个连接会一直占用Server的SYN队列中的一个位置，大量这样的连接就会将Server的SYN连接队列耗尽，让正常的连接无法得到处理。

目前，Linux下**默认会进行5次重发SYN-ACK包**，重试的间隔时间从1s开始，下次的重试间隔时间是前一次的两倍，5次重试时间间隔分别为1s, 2s, 4s, 8s, 16s，总共31s，第5次发出后还要再等32s才知道第5次也超时，所以**总共需要 1s + 2s + 4s+ 8s+ 16s + 32s = 63s，TCP才会断开这个连接**。

由于SYN超时需要63秒，那么就给攻击者一个攻击服务器的机会，攻击者在短时间内发送大量的SYN包给Server**(SYN Flood 攻击)**，耗尽Server的SYN队列，对于这种情况，linux提供了如下几个TCP参数：

**net.ipv4.tcp\_syncookies = 1**  #开启 syncookies 功能

**net.ipv4.tcp\_max\_syn\_backlog** = 262144 # 最大SYN队列大小

**net.ipv4.tcp\_synack\_retries** = 2 # SYN/ACK 重试次数

**net.ipv4.tcp\_syn\_retries** = 2 # SYN重试次数

**tcp\_abort\_on\_overflow = 1** #Accept队列满**，**0则直接丢弃ACK，1则首先丢弃ACK，然后发送RST通知Client

**net.ipv4.tcp\_synack\_retries** #内核放弃连接之前发送SYN+ACK包的数量

**net.ipv4.tcp\_syn\_retries** #内核放弃建立连接之前发送SYN包的数量

为了应对SYN Flooding，Linux实现了一种称为SYNcookie的机制，通过参数**net.ipv4.tcp\_syncookies**控制，1表示开启。简单地说**SYNcookie就是将连接信息编码在ISN(InitialSequenceNumber)中返回给客户端**，**这时server不需要将半连接保存在队列中，而是利用客户端随后发来的ACK带回的ISN还原连接信息，以完成连接的建立，避免了半连接队列被攻击SYN包填满。**

**syn队列满的情况(tcp\_abort\_on\_overflow)**

SYN半连接队列的大小是由（/proc/sys/net/ipv4/tcp\_max\_syn\_backlog）内核参数控制。

1. 当**SYN队列满，不开启syncookies**，**Server会丢弃新来的SYN包**，而Client端在多次重发SYN包得不到响应而返回**（connection time out）**错误；

2. 当**SYN队列满，开启syncookies=1**，**Server端SYN半连接队列就没有逻辑上的最大值了**，并且/proc/sys/net/ipv4/tcp\_max\_syn\_backlog设置的值也会被忽略。

**查看SYN Queue溢出：**

**netstat -s | grep LISTEN**

4375 SYNs to LISTEN sockets dropped

**accept队列满的情况**

当accept队列满了之后，server通过**/proc/sys/net/ipv4/tcp\_abort\_on\_overflow**来决定如何返回**：**

**1. tcp\_abort\_on\_overflow=0，TCP协议栈将此连接标记为acked，但仍保留在SYNQueue中，并启动Timer以便重发SYN/ACK包，当SYN/ACK重传次数超过net.ipv4.tcp\_synack\_retries时，再将该连接从SYNQueue中移除；**

**2. tcp\_abort\_on\_overflow=1，TCP协议栈回复RST包，并直接从SYN queue中删除该连接信息；**

**相应地，client则会分别返回read timeout 或者 connection reset by peer。**

**查看Accept Queue溢出：**

**netstat -s | grep listen**

22438 times the listen queue of a socket overflowed

**accept队列满了，对 syn队列也有影响，**在代码 net/ipv4/tcp\_ipv4.c ：

int tcp\_v4\_conn\_request(struct sock \*sk, struct sk\_buff \*skb)

{

/\*tcp\_syncookies为2 进行syn cookie

tcp\_syncookies为1 且request队列满了 进行syn cookie处理

tcp\_syncookies为0 且request队列满了 将该syn报文drop掉

\*/

if ((sysctl\_tcp\_syncookies == 2 ||

inet\_csk\_reqsk\_queue\_is\_full(sk)) && !isn) {

want\_cookie = tcp\_syn\_flood\_action(sk, skb, "TCP");

if (!want\_cookie)

goto drop;

}

/\* Accept backlog is full. If we have already queued enough

\* of warm entries in syn queue, drop request.

\*/

if (sk\_acceptq\_is\_full(sk) && inet\_csk\_reqsk\_queue\_young(sk) > 1) {

NET\_INC\_STATS\_BH(sock\_net(sk), LINUX\_MIB\_LISTENOVERFLOWS);

goto drop;

}

accept队列大多数情况下会比较小，所以会出现SYN 队列没有满，而ACCEPT 队列满了的情况，此时**会按照tcp\_aborton\_overflow来决定直接丢弃，还是返回拒绝RST。 如果启用了syncookies，那么会限制SYN包进入的速度**。

当系统丢弃最后的ACK，Linux会根据**net.ipv4.tcp\_synack\_retries**重新发送 SYN/ACK 包，客户端收到多个SYN/ACK 包，则会认为之前的ACK丢包了，于是客户端再次发送ACK，在accept队列有空闲的时候最终完成连接，若accept队列始终满，则最终客户端收到RST包。