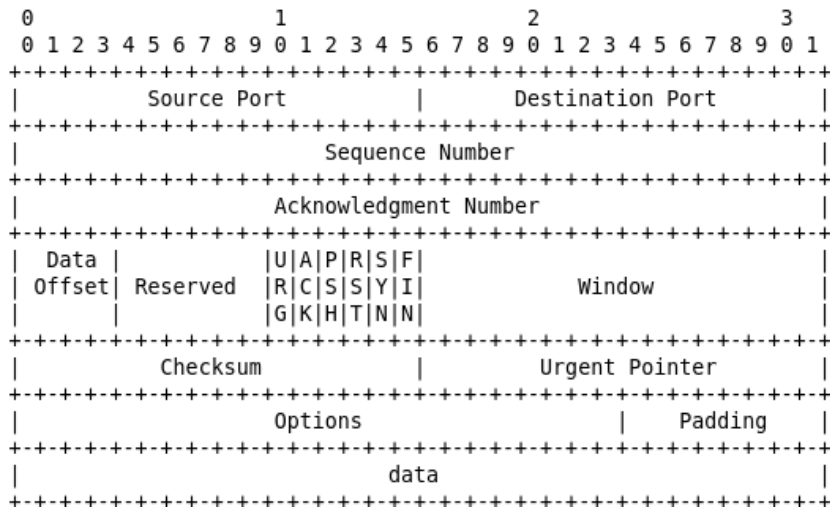


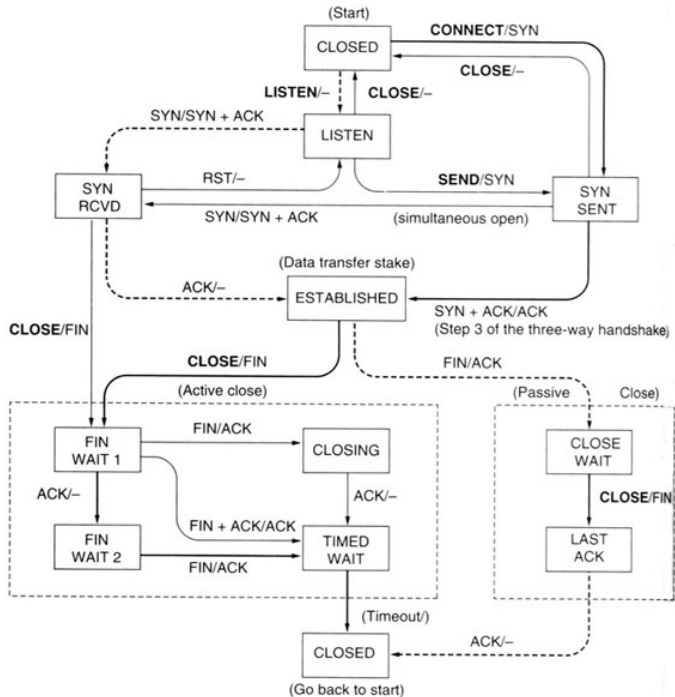
TCP: Transmission Control Protocol

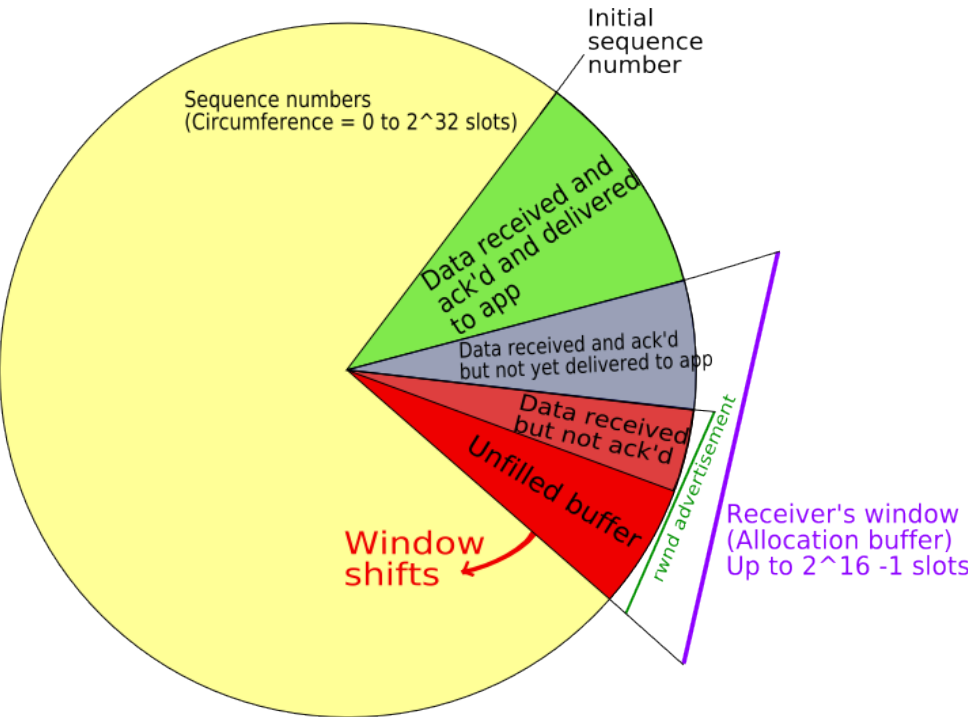
TCP Header Format



TCP Header Format

Note that one tick mark represents one bit position.





stream data



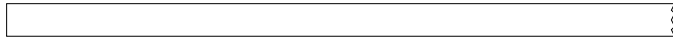
0

100

200

...

stream data



0

100

200

...

goal: send this stream
of data to a remote peer

stream data



0

100

200

...

**goal: send this stream
of data to a remote peer**

stream data



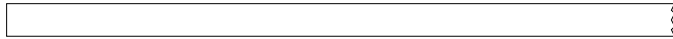
0

100

200

...

stream data



0

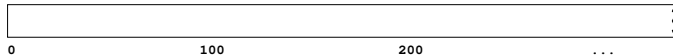
100

200

...

but over a network
with unreliable delivery
out of order delivery
repeated delivery
and max message size

stream data



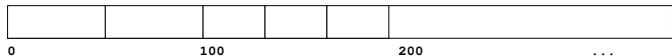
☐ **data present**

☒ **data missing**

stream data



stream data



☐ data present

☒ data missing

stream data



stream data



0

100

200

...



segments

☐ **data present**

☒ **data missing**

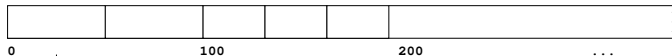
stream data



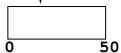
100

200

stream data



send()



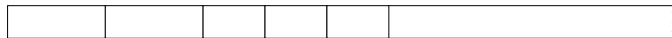
☐ data present

☒ data missing

stream data



stream data



0

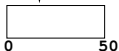
100

200

...

send()

send()



0

50

50

100

☐ data present

☒ data missing

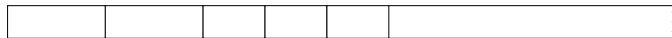
stream data



100

200

stream data



0

100

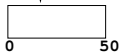
200

...

send()

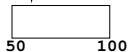
send()

...



0

50



50

100

☐ data present

☒ data missing

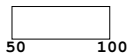
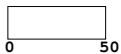
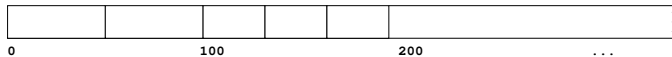
stream data



100

200

stream data



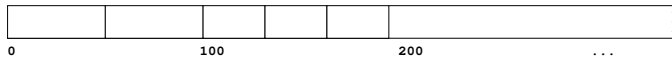
□ data present

■ data missing

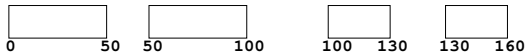
stream data



stream data



sent
data



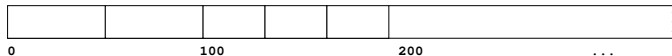
□ data present

■ data missing

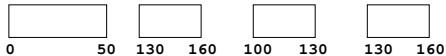
stream data



stream data



recv
data



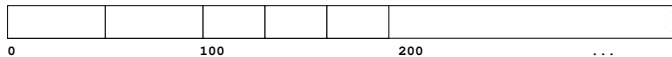
☐ data present

☒ data missing

stream data

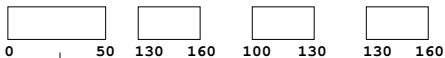


stream data



□ data present

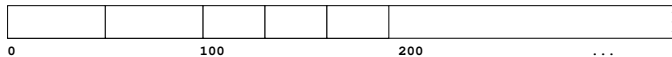
■ data missing



stream data

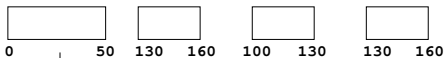


stream data



□ data present

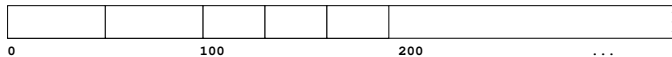
■ data missing



stream data



stream data



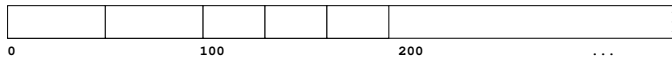
□ data present

■ data missing

stream data

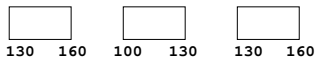


stream data



□ data present

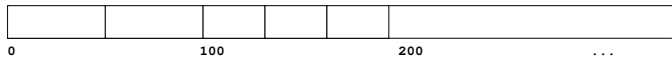
■ data missing



stream data

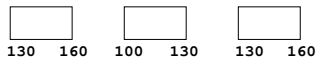


stream data



□ data present

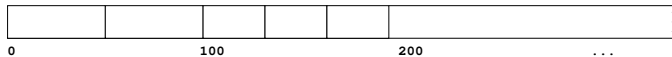
■ data missing



stream data

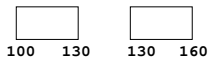


stream data



□ data present

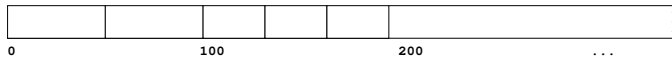
■ data missing



stream data



stream data



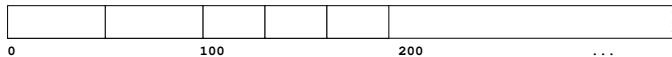
□ data present

■ data missing

stream data



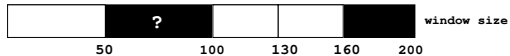
stream data



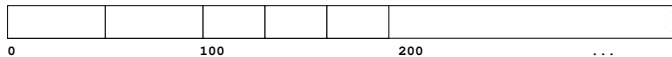
□ data present

■ data missing

stream data



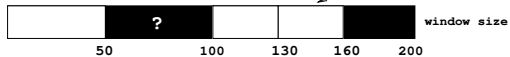
stream data



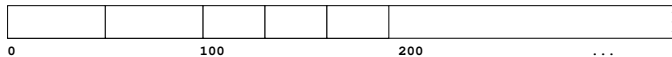
□ data present

■ data missing

stream data



stream data



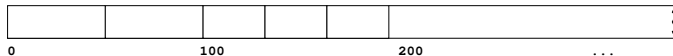
□ data present

■ data missing

stream data



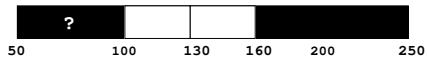
stream data



☐ data present

☒ data missing

stream data



Two ways to Close TCP

- FIN control message
 - reliably delivered, other side ACKs
 - signals “I have no more data to send”
 - other side can still send data
 - consumes an octet for ACK purposes
- RST control message
 - unilateral
 - takes effect immediately (no ack needed)
 - only accepted if it has “correct” sequence number
 - i.e., within reasonable window
 - “Connection reset by peer.”

When to send reset

- first, never in response to a RST
- any segment received on a CLOSED connection
 - if ACK is on, use ACK as the SEQ number in reply
 - if ACK is off, use 0 as SEQ in reply
- any ACK received in LISTEN, use ACK as SEQ in reply
- incorrect ACK in SYN-SENT or SYN-RECEIVED, use ACK as SEQ in reply
- if a SYN is set with SEQ in-window in all other states

Examples when not to send reset

- SYNs after ESTABLISHED with SEQ out of window
 - could be retransmission or delayed prior to connect
 - send an ACK back to the sender.
- ACKs for data not yet sent
 - not fatal, just ignore entire packet
- data sent out of the window
 - not fatal, handled with ACKs

Attacks on TCP

TCP Threat: Reset Attack

- a single packet to either A or B ends the TCP connection
- Eve can interfere with communication
 - low cost: single packet does it
 - need to know port and a reasonable sequence number
 - trivial for non-blind attacker
 - often one side is known (port 80, 443)
- any packet that is sent after is rejected
- what kind of attack is this?
- what is it an attack on?

This is not an existential threat:

This is not an existential threat:
this is one of the techniques used
by the People's Republic of China
to prevent their citizens from accessing
news from non government-approved sources.

This is not an existential threat:
this is one of the techniques used
by the People's Republic of China
to prevent their citizens from accessing
news from non government-approved sources.
its also how Comcast stopped bittorrent

Ignoring the Great Firewall of China

Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson

University of Cambridge, Computer Laboratory, William Gates Building,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom

`{richard.clayton, steven.murdoch, robert.watson}@cl.cam.ac.uk`

Abstract. The so-called “Great Firewall of China” operates, in part, by inspecting TCP packets for keywords that are to be blocked. If the keyword is present, TCP reset packets (viz: with the RST flag set) are sent to both endpoints of the connection, which then close. However, because the original packets are passed through the firewall unscathed, if the endpoints completely ignore the firewall’s resets, then the connection will proceed unhindered. Once one connection has been blocked, the firewall makes further easy-to-evade attempts to block further connections from the same machine. This latter behaviour can be leveraged into a denial-of-service attack on third-party machines.


cam(54190) → china(http) [SYN]
china(http) → cam(54190) [SYN, ACK] TTL=39
cam(54190) → china(http) [ACK]
cam(54190) → china(http) GET /?falun HTTP/1.0<cr><lf><cr><lf>
china(http) → cam(54190) [RST] TTL=47, seq=1, ack=1
china(http) → cam(54190) [RST] TTL=47, seq=1461, ack=1
china(http) → cam(54190) [RST] TTL=47, seq=4381, ack=1
china(http) → cam(54190) HTTP/1.1 200 OK (text/html)<cr><lf> etc...
cam(54190) → china(http) [RST] TTL=64, seq=25, ack zeroed
china(http) → cam(54190) ...more of the web page
cam(54190) → china(http) [RST] TTL=64, seq=25, ack zeroed
china(http) → cam(54190) [RST] TTL=47, seq=2921, ack=25

6 Denial-of-Service Attacks

As we have already noted, a single TCP packet containing a request such as `?falun` is sufficient to trigger blocking between the destination address and source address for periods of up to an hour. If the source of the packet is forged, this permits a (somewhat limited) denial-of-service attack which will prevent a particular pair of endpoints from communicating. However, depending upon their motives, this might be sufficient for some attackers. For example, it might be possible to identify the machines used by regional government offices and prevent them accessing “Windows Update”; or prevent a particular ministry accessing specific UN websites; or prevent access by Chinese embassies abroad to particular Chinese websites “back home”.

Comcast Throttles BitTorrent Traffic, Seeding Impossible

August 17, 2007 by **Ernesto Van der Sar**

 265 comments

[HOME](#) > [TECHNOLOGY](#) > [BITTORRENT](#) >

Over the past weeks more and more Comcast users started to notice that their BitTorrent transfers were cut off. Most users report a significant decrease in download speeds, and even worse, they are unable to seed their downloads. A nightmare for people who want to keep up a positive ratio at private trackers and for the speed of BitTorrent transfers in general.

ISPs have been throttling BitTorrent traffic for almost two years now.



Most ISPs simply limit the available bandwidth for BitTorrent traffic,

but [Comcast](#) takes it one step further, and prevents their customers from seeding. And

Comcast is not alone in this, Canadian ISPs Cogeco and Rogers use [similar methods on a smaller scale](#).

Comcast Defends Role As Internet Traffic Cop

By Cecilia Kang





Washington Post Staff Writer

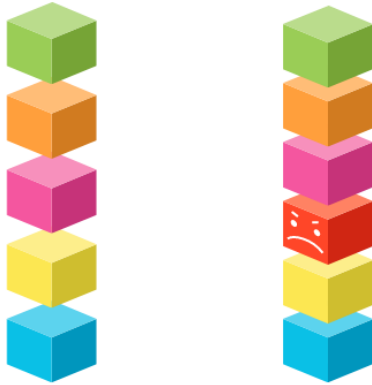
Wednesday, February 13, 2008

Comcast said yesterday that it purposely slows down some traffic on its network, including some music and movie downloads, an admission that sparked more controversy in the debate over how much control network operators should have over the Internet.

In a filing with the Federal Communications Commission, Comcast said such measures -- which can slow the transfer of music or video between subscribers sharing files, for example -- are necessary to ensure better flow of traffic over its network.

In defending its actions, Comcast stepped into one of the technology industry's most divisive battles. Comcast argues that it should be able to direct traffic so networks don't get clogged; consumer groups and some Internet companies argue that the networks should not be permitted to block or slow users' access to the Web.

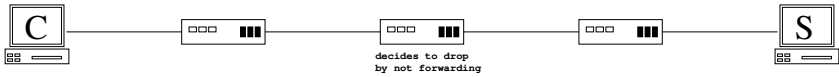
TOOLBOX	
 Resize	 Print
 E-mail	 Reprints

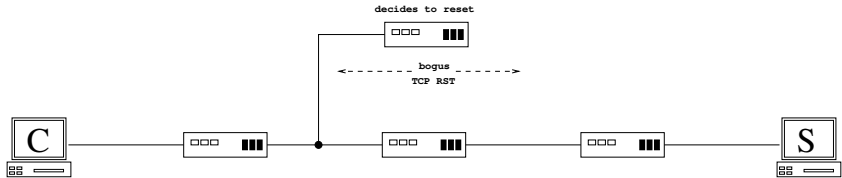


Packet Forgery By ISPs: A Report On The Comcast Affair

FCC formally rules Comcast's throttling of BitTorrent was illegal

Regulators may be on shaky legal ground when voting 3-2 to say that BitTorrent throttling violated Net neutrality rules, a first, which invites a legal challenge from Comcast.





TCP Reset Attack

- SEQ number should reflect ACK in segment to which the RST is in response
 - SEQ of zero used if no ACK present
- TCP requires that receivers accept RST liberally
 - any in-window sequence number should be accepted
 - e.g., previous TCP data segments may have been lost
 - RST should never trigger with a SEQ less than data already received

TCP SYN Reset Attack

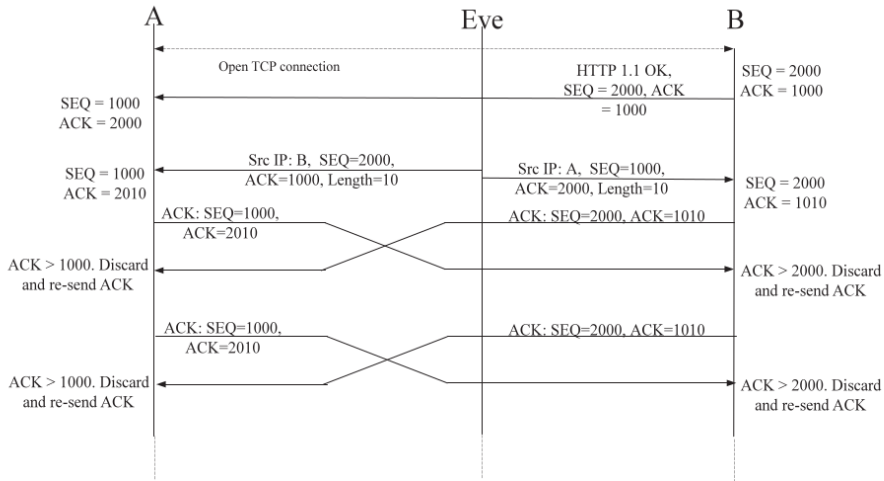
- TCP will send a RST if the SYN bit is set and the sequence number is in-window
 - attacker sends a SYN, victim sends the RST
- both cases require knowing valid window
 - window size is a function of available memory
 - has been increasing in the years
 - easier for blind attackers
 - need $\frac{2^{32}}{W}$ total guesses

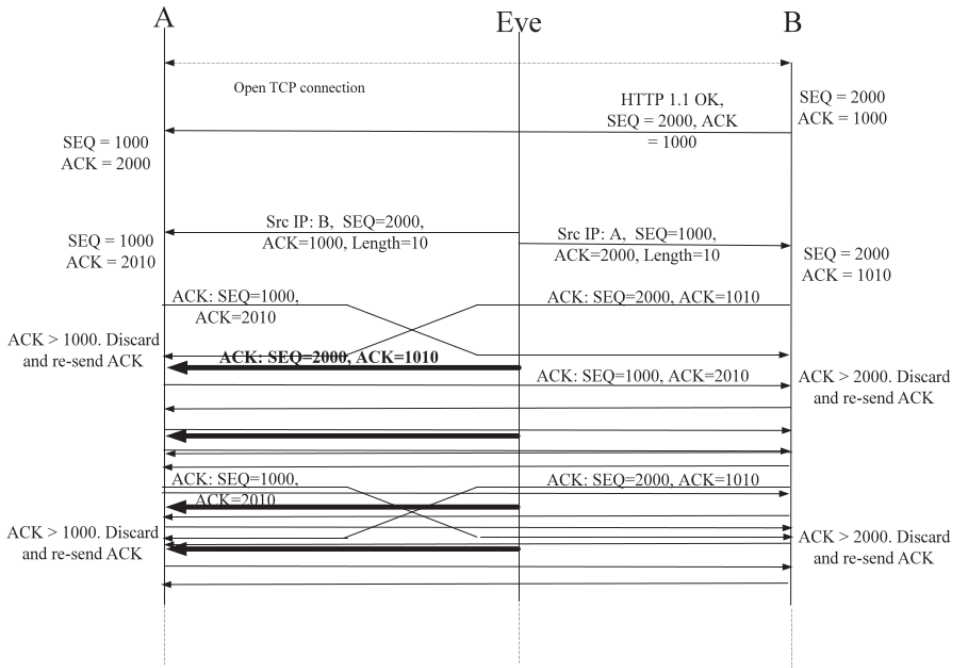
Why RST Attack Popular with Censors

- easy to cause
 - only need one packet that is accepted by one of the two endpoints
- easy to manage
 - no need to consult ACL regarding what flows to allow
 - gets the victims to do the work instead of the censor
- low risk
 - RST are accepted silently and generate no response
 - destabilizing the TCP connection other ways can result in ACK storm

ESTABLISHED STATE

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$ then, set $\text{SND.UNA} \leftarrow \text{SEG.ACK}$. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate ($\text{SEG.ACK} < \text{SND.UNA}$), it can be ignored. If the ACK acks something not yet sent ($\text{SEG.ACK} > \text{SND.NXT}$) then send an ACK, drop the segment, and return.





TCP Ack Storm

- stops only when congestion causes an ACK to get dropped
 - as ACK do not carry data, they are not retransmitted
- a easy to trigger DoS for others
- not good DoS attack for Comcast's illegal censorship (why?)

Detecting Reset Attack

- any reset attack by a third party has inherent issues with timing
 - e.g., SEQ number at the time decision is made to censor communication may change
- resulting RST packet will have features not representative of a real end host
- decisions for other TCP fields can be used to fingerprint the specific censorship system being used

Example Fraudulent Reset Features

- sending multiple RST at different SEQ numbers
 - e.g., +0, +1000, +2000, etc.
 - sending RST after current SEQ
 - sending RST before current SEQ
- having ACK as zero, or arbitrary value, or same as SEQ
 - accidentally incrementing the ACK instead of SEQ to do range
- how long from SYN to RST
- does real data arrive after RST with higher SEQ
- including data in the RST segment

Identified Source	Signature
Identified Injector	
Sandvine	Multipacket: First Packet IPID += 4, second packet SEQ + 12503, IPID += 5
Bezeqint	Multipacket: Constant sequence, RST_ACK_CHANGE, IPID = 16448
Yournet	SYN_RST: Only on SMTP, TTL usually +3 to +5, unrelated IPID
Victoria	Multipacket: Sequence Increment 1500, IPID = 305, TTL += 38
IPID 256	Single packet: Usually less TTL, IPID = 256
IPID 64	Multipacket: IPID = 64, often sequence increment of 1460
IPID -26	Multipacket: First IPID -= 26, often sequence increment of 1460
SEQ 1460	Multipacket: Sequence increment always 1460
RAE	Single packet: Sets RST, ACK and ECN nonce sum (control bit 8)
Go Away	Single packet: Payload on RST of "Go Away, We're Not Home"
Optonline	Multipacket: No fingerprint, all activity from a single ISP

- RFC for hardening TCP against blind injection attacks
- rewrites the rules for handling SYN and RST for established connections
- if the SYN bit is set in ESTABLISHED
 - ignore the sequence number
 - send a “challenge ACK”
 - current sequence and ack
 - absent attacker, honest peer would send a RST
 - this confirms loss of state
 - spoofed SYN will cause ACK to be ignored (duplicate)
 - ignore message


```

/* step 4: Check for a SYN
 * RFC 5961 4.2 : Send a challenge ack
 */
if (th->syn) {
    if (sk->sk_state == TCP_SYN_RECV && sk->sk_socket && th->ack &&
        TCP_SKB_CB(skb)->seq + 1 == TCP_SKB_CB(skb)->end_seq &&
        TCP_SKB_CB(skb)->seq + 1 == tp->rcv_nxt &&
        TCP_SKB_CB(skb)->ack_seq == tp->snd_nxt)
        goto pass;

syn_challenge:
    if (syn_inerr)
        TCP_INC_STATS(sock_net(sk), TCP_MIB_INERRS);
    NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPSYNCHALLENGE);
    tcp_send_challenge_ack(sk);
    SKB_DR_SET(reason, TCP_INVALID_SYN);
    goto discard;
}

pass:
    bpf_skops_parse_hdr(sk, skb);

    return true;

```

- if the RST bit is set in ESTABLISHED
 - if it is outside window, silently drop it
 - else if it **exactly** matches the expected sequence number, reset
 - else send a “challenge ACK”
 - a real RST now will cause another RST in response
 - will this cause an infinite RST/ACK cycle?

```

/* Step 2: check RST bit */
if (th->rst) {
    /* RFC 5961 3.2 (extend to match against (RCV.NXT - 1) after a
    * FIN and SACK too if available):
    * If seq num matches RCV.NXT or (RCV.NXT - 1) after a FIN, or
    * the right-most SACK block,
    * then
    *     RESET the connection
    * else
    *     Send a challenge ACK
    */
    if (TCP_SKB_CB(skb)->seq == tp->rcv_nxt ||
        tcp_reset_check(sk, skb))
        goto reset;

    // [ removed SACK handling ]

    /* Disable TFO if RST is out-of-order
    * and no data has been received
    * for current active TFO socket
    */
    if (tp->syn_fastopen && !tp->data_segs_in &&
        sk->sk_state == TCP_ESTABLISHED)
        tcp_fastopen_active_disable(sk);
    tcp_send_challenge_ack(sk);
    SKB_DR_SET(reason, TCP_RESET);
    goto discard;
}

```

TCP Threat: Data Injection

- insert data instead of disrupting a connection
 - attacker knows correct port and sequence number
 - receiver doesn't realize it is happening
 - source IP can be spoofed
- called TCP **connection hijacking** or **session hijacking**
 - take over an already established connection

Bob's TCP stream from Alice

stream start

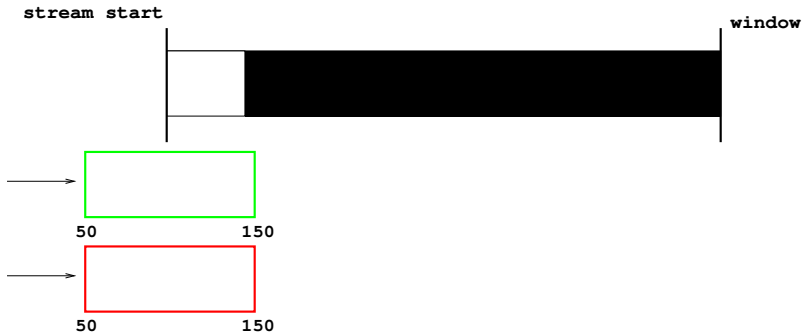


Bob's TCP stream from Alice

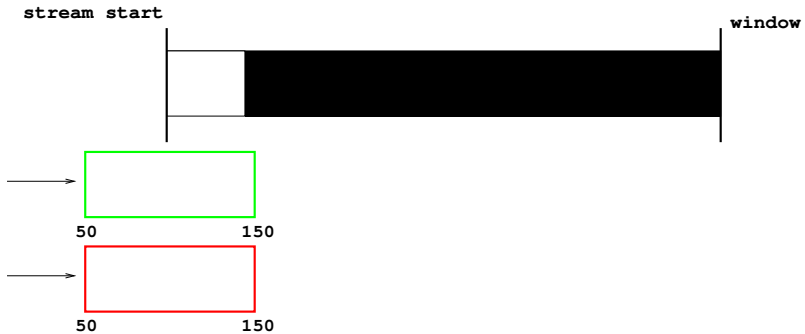
stream start



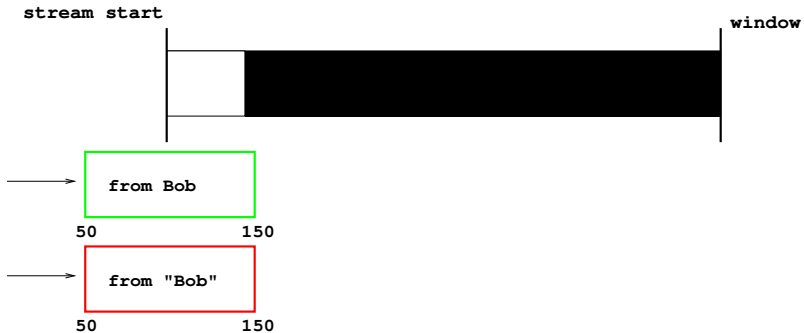
Bob's TCP stream from Alice



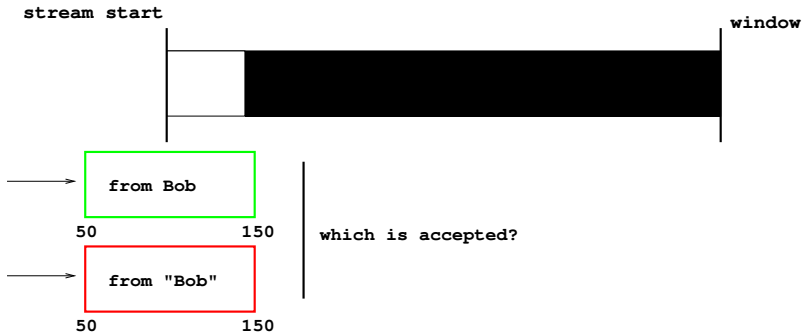
Bob's TCP stream from Alice



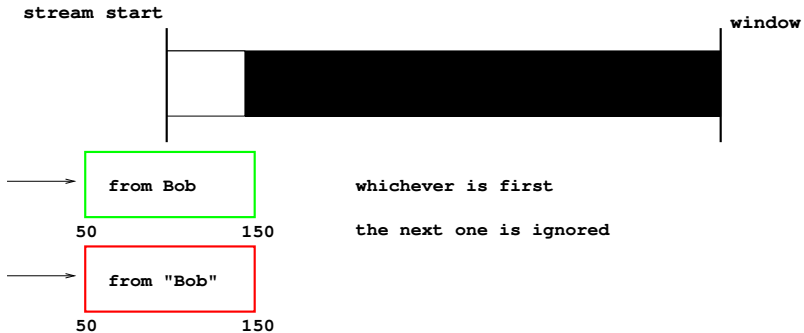
Bob's TCP stream from Alice



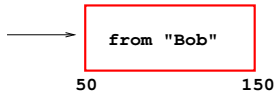
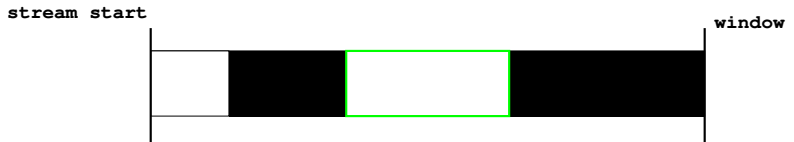
Bob's TCP stream from Alice



Bob's TCP stream from Alice

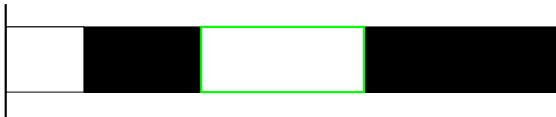


Bob's TCP stream from Alice



Bob's TCP stream from Alice

stream start



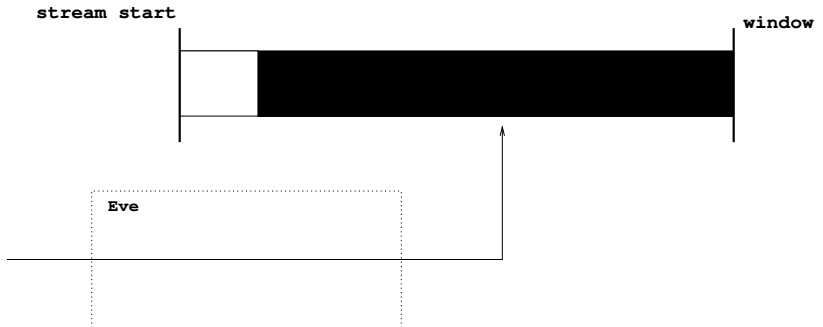
window

Bob's TCP stream from Alice

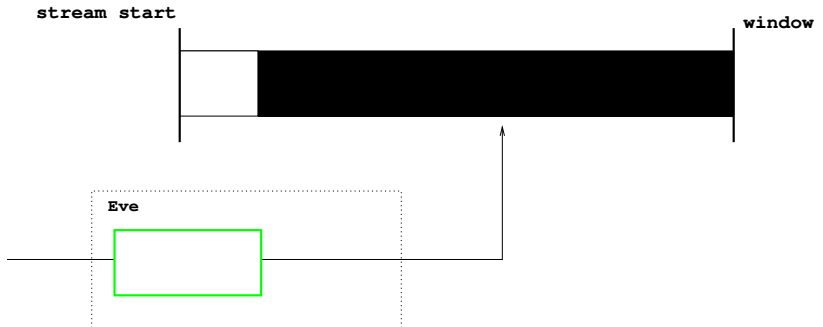
stream start



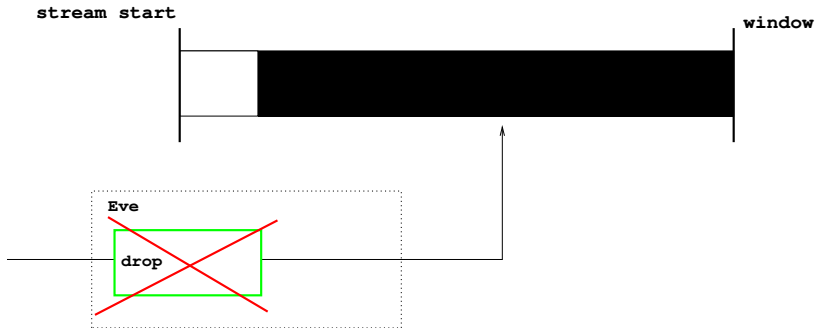
Bob's TCP stream from Alice



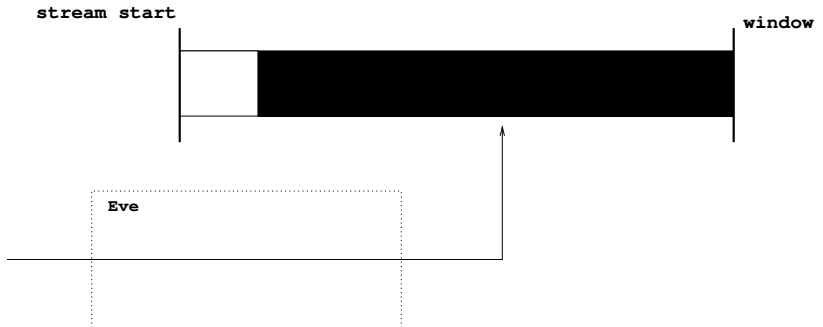
Bob's TCP stream from Alice



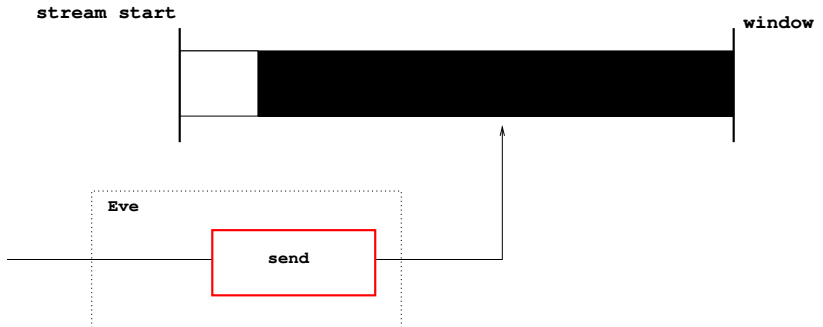
Bob's TCP stream from Alice



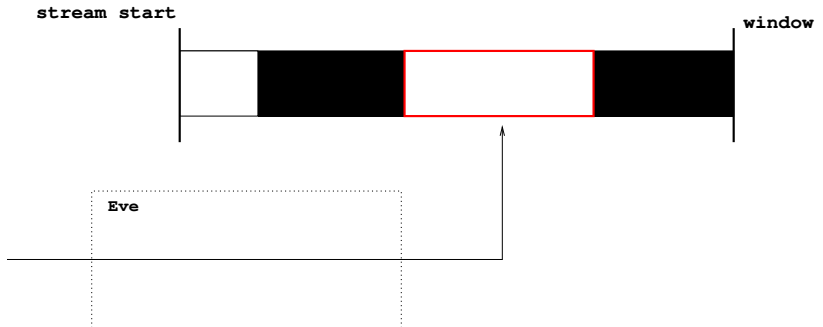
Bob's TCP stream from Alice



Bob's TCP stream from Alice



Bob's TCP stream from Alice



TCP Session Hijacking

- $A \rightarrow B$:
 - srcIP:srcPort to destIP:destPort, seq X, ack Y
 - HTTP GET /login.html
- $A \leftarrow E$:
 - destIP:destPort to srcIP:srcPort, seq Y, ack X+20
 - 200 OK ... (bad spoofed data)
- then shortly after
- $A \leftarrow B$:
 - destIP:destPort to srcIP:srcPort, seq Y, ack X+20
 - 200 OK ... (actual data)
- B's reply is ignored, treated as duplicate packet
 - why doesn't OS check they aren't equal and warn?

This technique can be used to spoof entire session

This technique can be used to spoof entire session
allows attacker blame a client for behaviour

This technique can be used to spoof entire session
allows attacker blame a client for behaviour
allows attacker to receive IP-based credibility

This technique can be used to spoof entire session
allows attacker blame a client for behaviour
allows attacker to receive IP-based credibility
has to otherwise see traffic

TCP Threat: Blind Injection

TCP Threat: Blind Injection
is it possible for an attacker to inject
into a TCP connection even if they can't see?

TCP Threat: Blind Injection

is it possible for an attacker to inject
into a TCP connection even if they can't see?

YES: just need to guess sequence number and port

TCP Threat: Blind Injection

is it possible for an attacker to inject
into a TCP connection even if they can't see?

YES: just need to guess sequence number and port
security by design: ports and seq numbers
weren't designed to fight this threat!

- optional defense to improve blind injection
 - require ACKs on messages that try to add data
 - ACK is checked for correctness
 - ACKs out of window are ignored
 - ACKs in window are ignored
 - peer is ACKing data I haven't sent

- problem: TCP allowed half the ACK space to be valid
 - divide the 2^{32} possible ACKs into “future” and “past”
 - future is 2^{31} ACKS beyond current sent data
 - past is the other half
 - blind attacker needs two guesses
- solution: track the peer's max receiving window size
 - the most data it ever claimed to be willing to accept without ACKing
 - any ACK will be within this range of the current ACK
 - or use maximum possible window size of 65535
 - blind attacker needs $\frac{2^{32}}{W}$ guesses

```
/* If the ack is older than previous acks
 * then we can probably ignore it.
 */
if (before(ack, prior_snd_una)) {
    u32 max_window;

    /* do not accept ACK for bytes we never sent. */
    max_window = min_t(u64, tp->max_window, tp->bytes_acked);
    /* RFC 5961 5.2 [Blind Data Injection Attack].[Mitigation] */
    if (before(ack, prior_snd_una - max_window)) {
        if (!(flag & FLAG_NO_CHALLENGE_ACK))
            tcp_send_challenge_ack(sk);
        return -SKB_DROP_REASON_TCP_TOO_OLD_ACK;
    }
    goto old_ack;
}
```


TCP Threat: Blind Spoofing

- can spoof an entire session
 - not just add a packet
- attacker spoofs the initial SYN with a fake IP
- attacker then anticipates the rest of the session
 - protocols help this
- but what can go wrong?

A

B

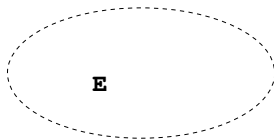
A

B

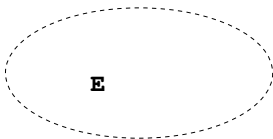



E

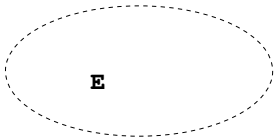
A $\xrightarrow{\text{SYN 239472}}$ **B**



A ← **B**
SYN 157329
ACK 239472



A  **B**
ACK 157329



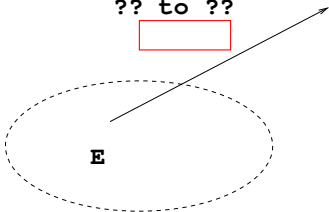
A

?? to ??



B

E



A

B

E

A dashed ellipse is drawn around the letter 'E', which is positioned in the lower center of the image. The ellipse is horizontally oriented and its boundary is composed of short, evenly spaced dashes.

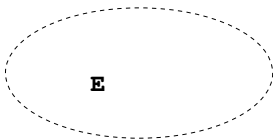
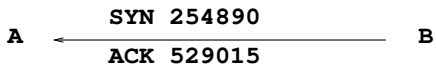
A

B

from A
SYN 254890

E

The diagram consists of three points labeled A, B, and E. Point A is located in the upper left. Point B is located in the upper right. Point E is located in the lower center and is enclosed within a dashed oval. A straight arrow points from point E to point B. To the right of the arrow, the text 'from A' and 'SYN 254890' is written.



A

B

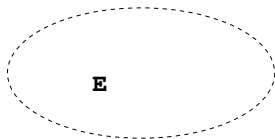
ACK ???

E



A diagram illustrating a network communication scenario. A dashed oval labeled 'E' is positioned in the lower-left area. An arrow originates from the right side of this oval and points towards the text 'ACK ???' located in the upper-right area. The labels 'A' and 'B' are positioned at the top left and top right respectively.

A $\xrightarrow{\text{RST}}$ **B**



TCP Threat: Blind Spoofing

- problem 1

- if a client receives the response from server they will send RST
- if they aren't reachable then the attacker can fake ACKs
- if client does respond then attacker must do this in a hurry
 - or DoS them first

- problem 2

- attacker doesn't see reply!
- so attacker doesn't know the server's SYN number to ACK
- so how can the attacker figure out what to ack?

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

What happens if attacker sends a packet with the source IP and port equal to the destination IP and port and SYN set?

What happens if attacker sends a packet
with the source IP and port equal to
the destination IP and port and SYN set?
Triggers loop in Windows XP SP2
implementation of TCP/IP and locks CPU.

TCP Threat: Syn Flood

- $E \rightarrow S: \text{SYN} = X$
- $E \leftarrow S: \text{SYN}=Y, \text{ACK}=X, \text{remember } X, Y, \text{state, etc.}$
- $E \rightarrow S: \text{SYN} = X'$
- $E \leftarrow S: \text{SYN}=Y', \text{ACK}=X', \text{remember } X', Y', \text{state', etc.}$
- $E \rightarrow S: \text{SYN} = X''$
- $E \leftarrow S: \text{SYN}=Y'', \text{ACK}=X'', \text{remember } X'', Y'', \text{state'', etc.}$
- ...
- what kind of attack is this?

TCP Threat: SYN Flood

- classic denial of service
- server must allocate resources for each SYN
 - keeps state until a timeout is reached
 - fixed bound on number of half-open connections
 - prevents legitimate clients from connecting
- attacker can never continue with connection and not tell server
- attacker just keeps sending more SYNs
- server eventually runs out of storage and starts dropping connections
- server may spawn a thread for each request
 - depends on how it is implemented
 - asymmetries in security!
- first done in 1994 by Mitnick to take down a server and impersonate it

Defence: Random Deletion

- if SYN queue is full, delete a random one
 - legitimate connections have a chance to complete
 - if it is a DoS, likely to have a good chance to delete attacker connection (why?)
 - easy to implement

Defence: use a proxy service

- anti-DoS service handles creating TCP connections
 - does the establishing of TCP connections
 - forwards finished ones to the website

Defence: SYN Cookies

- from djb in 1996
- idea is to impose state storage on the client
 - server chooses the SYN number to actually store data
 - they don't store any state until they get an ACK
- server's SYN is, e.g.,
`HMAC(secret,srcAddr|srcPort|destAddr|destPort|time)`
- client must reply with this as their ACK to finish connection
- no state is stored from the initial SYN
- on the ACK, the SYN cookie is recomputed from the available data and checked

Why do SYN cookies fix the problem?
The attacker can just answer the SYN-ACK with the
cookie?

Does Linux use syncookies?

```

static __u32 secure_tcp_syn_cookie(__be32 saddr, __be32 daddr, __be16 sport,
                                   __be16 dport, __u32 sseq, __u32 data)
{
    /*
     * Compute the secure sequence number.
     * The output should be:
     *   HASH(sec1,saddr,sport,daddr,dport,sec1) + sseq + (count * 2^24)
     *   + (HASH(sec2,saddr,sport,daddr,dport,count,sec2) & 2^24).
     * Where sseq is their sequence number and count increases every
     * minute by 1.
     * As an extra hack, we add a small "data" value that encodes the
     * MSS into the second hash value.
     */
    u32 count = tcp_cookie_time();
    return (cookie_hash(saddr, daddr, sport, dport, 0, 0) +
            sseq + (count << COOKIEBITS) +
            ((cookie_hash(saddr, daddr, sport, dport, count, 1) + data)
             & COOKIEMASK));
}

```



```

/*
 * Return true if a syncookie should be sent
 */
static bool tcp_syn_flood_action(struct sock *sk, const char *proto)
{
    struct request_sock_queue *queue = &inet_csk(sk)->icsk_accept_queue;
    const char *msg = "Dropping request";
    struct net *net = sock_net(sk);
    bool want_cookie = false;
    u8 syncookies;

    syncookies = READ_ONCE(net->ipv4.sysctl_tcp_syncookies);

#ifdef CONFIG_SYN_COOKIES
    if (syncookies) {
        msg = "Sending cookies";
        want_cookie = true;
        __NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPREQQFULLDOCOOKIES);
    } else
#endif
        __NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPREQQFULLDROP);

    if (!READ_ONCE(queue->synflood_warned) && syncookies != 2 &&
        xchg(&queue->synflood_warned, 1) == 0) {
        if (IS_ENABLED(CONFIG_IPV6) && sk->sk_family == AF_INET6) {
            net_info_ratelimited("%s: Possible SYN flooding on port [%pI6c]:%u. %s.\n",
                                proto, inet6_rcv_saddr(sk),
                                sk->sk_num, msg);
        } else {
            net_info_ratelimited("%s: Possible SYN flooding on port %pI4:%u. %s.\n",
                                proto, &sk->sk_rcv_saddr,
                                sk->sk_num, msg);
        }
    }

    return want_cookie;
}

```

TCP Security Summary

- attacker who can observe TCP can manipulate it
 - terminate with RST packets
 - inject data in either direction
 - can adjust sequence numbers to avoid detection
- attacker who can predict ISN can “blind spoof” connections (why?)
 - makes it look like a client has connected
 - undermines any security based on trusting IP addresses
 - allows attacker to “frame” someone or avoid detection
 - “fixed” by random ISNs
- attacker can cause a DoS attack with a SYN flood