Network Working Group
Request for Comments: 1337

R. Braden
ISI

May 1992

TIME-WAIT Assassination Hazards in TCP

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Abstract

This note describes some theoretically-possible failure modes for TCP connections and discusses possible remedies. In particular, one very simple fix is identified.

1. INTRODUCTION

Experiments to validate the recently-proposed TCP extensions [RFC-1323] have led to the discovery of a new class of TCP failures, which have been dubbed the "TIME-WAIT Assassination hazards". This note describes these hazards, gives examples, and discusses possible prevention measures.

The failures in question all result from old duplicate segments. In brief, the TCP mechanisms to protect against old duplicate segments are [RFC-793]:

(1) The 3-way handshake rejects old duplicate initial <SYN> segments, avoiding the hazard of replaying a connection.

(2) Sequence numbers are used to reject old duplicate data and ACK segments from the current incarnation of a given connection (defined by a particular host and port pair). Sequence numbers are also used to reject old duplicate <SYN,ACK> segments.

    For very high-speed connections, Jacobson's PAWS ("Protect Against Wrapped Sequences") mechanism [RFC-1323] effectively extends the sequence numbers so wrap-around will not introduce a hazard within the same incarnation.

(3) There are two mechanisms to avoid hazards due to old duplicate segments from an earlier instance of the same connection; see the Appendix to [RFC-1185] for details.

For "short and slow" connections [RFC-1185], the clock-driven ISN (initial sequence number) selection prevents the overlap of the sequence spaces of the old and new incarnations [RFC-793]. (The algorithm used by Berkeley BSD TCP for stepping ISN complicates the analysis slightly but does not change the conclusions.)

(4) TIME-WAIT state removes the hazard of old duplicates for "fast" or "long" connections, in which clock-driven ISN selection is unable to prevent overlap of the old and new sequence spaces. The TIME-WAIT delay allows all old duplicate segments time enough to die in the Internet before the connection is reopened.

(5) After a system crash, the Quiet Time at system startup allows old duplicates to disappear before any connections are opened.

Our new observation is that (4) is unreliable: TIME-WAIT state can be prematurely terminated ("assassinated") by an old duplicate data or ACK segment from the current or an earlier incarnation of the same connection. We refer to this as "TIME-WAIT Assassination" (TWA).

Figure 1 shows an example of TIME-WAIT assassination. Segments 1-5 are copied exactly from Figure 13 of RFC-793, showing a normal close handshake. Packets 5.1, 5.2, and 5.3 are an extension to this sequence, illustrating TWA. Here 5.1 is *any* old segment that is unacceptable to TCP A. It might be unacceptable because of its sequence number or because of an old PAWS timestamp. In either case, TCP A sends an ACK segment 5.2 for its current SND.NXT and RCV.NXT. Since it has no state for this connection, TCP B reflects this as RST segment 5.3, which assassinates the TIME-WAIT state at A!

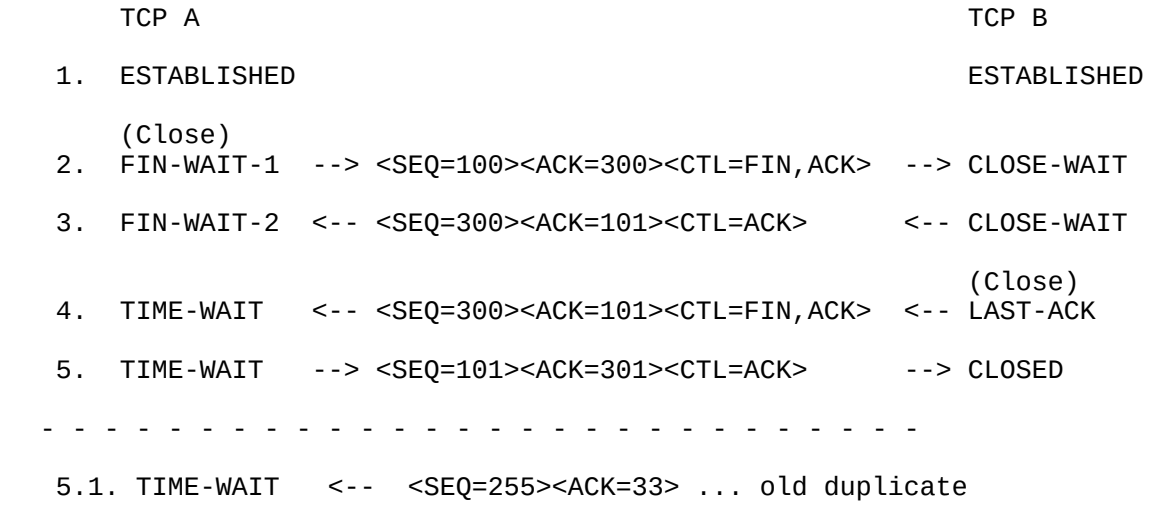

Figure 1. TWA Example

Note that TWA is not at all an unlikely event if there are any duplicate segments that may be delayed in the network. Furthermore, TWA cannot be prevented by PAWS timestamps; the event may happen within the same tick of the timestamp clock. TWA is a consequence of TCP's half-open connection discovery mechanism (see pp 33-34 of [RFC-793]), which is designed to clean up after a system crash.

2. The TWA Hazards

2.1 Introduction

If the connection is immediately reopened after a TWA event, the new incarnation will be exposed to old duplicate segments (except for the initial <SYN> segment, which is handled by the 3-way handshake). There are three possible hazards that result:

H1. Old duplicate data may be accepted erroneously.

H2. The new connection may be de-synchronized, with the two ends in permanent disagreement on the state. Following the spec of RFC-793, this desynchronization results in an infinite ACK

loop. (It might be reasonable to change this aspect of RFC-793 and kill the connection instead.)

This hazard results from acknowledging something that was not sent. This may result from an old duplicate ACK or as a side-effect of hazard H1.

H3. The new connection may die.

A duplicate segment (data or ACK) arriving in SYN-SENT state may kill the new connection after it has apparently opened successfully.

Each of these hazards requires that the seqence space of the new connection overlap to some extent with the sequence space of the previous incarnation. As noted above, this is only possible for "fast" or "long" connections. Since these hazards all require the coincidence of an old duplicate falling into a particular range of new sequence numbers, they are much less probable than TWA itself.

TWA and the three hazards H1, H2, and H3 have been demonstrated on a stock Sun OS 4.1.1 TCP running in an simulated environment that massively duplicates segments. This environment is far more hazardous than most real TCP's must cope with, and the conditions were carefully tuned to create the necessary conditions for the failures. However, these demonstrations are in effect an existence proof for the hazards.

We now present example scenarios for each of these hazards. Each scenario is assumed to follow immediately after a TWA event terminated the previous incarnation of the same connection.

2.2 HAZARD H1: Acceptance of erroneous old duplicate data.

Without the protection of the TIME-WAIT delay, it is possible for erroneous old duplicate data from the earlier incarnation to be accepted. Figure 2 shows precisely how this might happen.

```
      TCP A                                                TCP B

1. ESTABL.  --> <SEQ=400><ACK=101><DATA=100><CTL=ACK> --> ESTABL.

2. ESTABL.  <--     <SEQ=101><ACK=500><CTL=ACK>     <--   ESTABL.

3.  (old dupl)...<SEQ=560><ACK=101><DATA=80><CTL=ACK> --> ESTABL.

4. ESTABL.  <--     <SEQ=101><ACK=500><CTL=ACK>     <--   ESTABL.

5. ESTABL.  --> <SEQ=500><ACK=101><DATA=100><CTL=ACK> --> ESTABL.

6.             ...  <SEQ=101><ACK=640><CTL=ACK>     <--   ESTABL.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

7a. ESTABL. --> <SEQ=600><ACK=101><DATA=100><CTL=ACK> --> ESTABL.

8a. ESTABL.  <--    <SEQ=101><ACK=640><CTL=ACK> ...

9a. ESTABL. --> <SEQ=700><ACK=101><DATA=100><CTL=ACK> --> ESTABL.
```

Figure 2: Accepting Erroneous Data

The connection has already been successfully reopened after the assumed TWA event. Segment 1 is a normal data segment and segment 2 is the corresponding ACK segment. Old duplicate data segment 3 from the earlier incarnation happens to fall within the current receive window, resulting in a duplicate ACK segment #4. The erroneous data is queued and "lurks" in the TCP reassembly queue until data segment 5 overlaps it. At that point, either 80 or 40 bytes of erroneous data is delivered to the user B; the choice depends upon the particulars of the reassembly algorithm, which may accept the first or the last duplicate data.

As a result, B sends segment 6, an ACK for sequence = 640, which is 40 beyond any data sent by A. Assume for the present that this ACK arrives at A *after* A has sent segment 7a, the next full data segment. In that case, the ACK segment 8a acknowledges data that has been sent, and the error goes undetected. Another possible continuation after segment 6 leads to hazard H3, shown below.

2.3 HAZARD H2: De-synchronized Connection

This hazard may result either as a side effect of H1 or directly from an old duplicate ACK that happens to be acceptable but acknowledges something that has not been sent.

Referring to Figure 2 above, suppose that the ACK generated by the old duplicate data segment arrived before the next data segment had been sent. The result is an infinite ACK loop, as shown by the following alternate continuation of Figure 2.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 7b. ESTABL.  <--    <SEQ=101><ACK=640><CTL=ACK>   ...
(ACK something not yet
 sent => send ACK)

 8b. ESTABL.  -->    <SEQ=600><ACK101><CTL=ACK>       -->   ESTABL.
                                                  (Below window =>
                                                       send ACK)

 9b. ESTABL.  <--    <SEQ=101><ACK=640><CTL=ACK>     <--    ESTABL.

                          (etc.!)
```

Figure 3: Infinite ACK loop

2.4 HAZARD H3: Connection Failure

An old duplicate ACK segment may lead to an apparent refusal of TCP A's next connection attempt, as illustrated in Figure 4. Here <W=...> indicates the TCP window field SEG.WIND.*

```
   TCP A                                                  TCP B

1. CLOSED                                                 LISTEN

2. SYN-SENT    --> <SEQ=100><CTL=SYN>                --> SYN-RCVD

3.         ... <SEQ=400><ACK=101><CTL=SYN,ACK><W=800>  <-- SYN-RCVD

4. SYN-SENT    <-- <SEQ=300><ACK=123><CTL=ACK> ... (old duplicate)

5. SYN-SENT    --> <SEQ=123><CTL=RST>                    --> LISTEN

6. ESTABLISHED <-- <SEQ=400><ACK=101><CTL=SYN,ACK><W=900> ...

7. ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>          --> LISTEN

8. CLOSED      <--  <SEQ=401><CTL=RST>                  <-- LISTEN
```

Figure 4: Connection Failure from Old Duplicate

The key to the failure in Figure 4 is that the RST segment 5 is acceptable to TCP B in SYN-RECEIVED state, because the sequence space of the earlier connection that produced this old duplicate overlaps the new connection space. Thus, <SEQ=123> in segment #5 falls within TCP B's receive window [101,900). In experiments, this failure mode was very easy to demonstrate. (Kurt Matthys has pointed out that this scenario is time-dependent: if TCP A should timeout and retransmit the initial SYN after segment 5 arrives and before segment 6, then the open will complete successfully.)

3. Fixes for TWA Hazards

We discuss three possible fixes to TCP to avoid these hazards.

(F1) Ignore RST segments in TIME-WAIT state.

If the 2 minute MSL is enforced, this fix avoids all three hazards.

This is the simplest fix. One could also argue that it is formally the correct thing to do; since allowing time for old duplicate segments to die is one of TIME-WAIT state's functions, the state should not be truncated by a RST segment.

(F2) Use PAWS to avoid the hazards.

Suppose that the TCP ignores RST segments in TIME-WAIT state, but only long enough to guarantee that the timestamp clocks on both ends have ticked. Then the PAWS mechanism [RFC-1323] will prevent old duplicate data segments from interfering with the new incarnation, eliminating hazard H1. For reasons explained below, however, it may not eliminate all old duplicate ACK segments, so hazards H2 and H3 will still exist.

In the language of the TCP Extensions RFC [RFC-1323]:

When processing a RST bit in TIME-WAIT state:

```
If (Snd.TS.OK is off) or (Time.in.TW.state() >= W)
    then enter the CLOSED state, delete the TCB,
    drop the RST segment, and return.

else simply drop the RST segment and return.
```

Here "Time.in.TW.state()" is a function returning the elapsed time since TIME-WAIT state was entered, and W is a constant that is at least twice the longest possible period for timestamp clocks, i.e., W = 2 secs [RFC-1323].

This assumes that the timestamp clock at each end continues to advance at a constant rate whether or not there are any open connections. We do not have to consider what happens across a system crash (e.g., the timestamp clock may jump randomly), because of the assumed Quiet Time at system startup.

Once this change is in place, the initial timestamps that occur on the SYN and {SYN,ACK} segments reopening the connection will be larger than any timestamp on a segment from earlier incarnations. As a result, the PAWS mechanism operating in the new connection incarnation will avoid the H1 hazard, ie. acceptance of old duplicate data.

The effectiveness of fix (F2) in preventing acceptance of old duplicate data segments, i.e., hazard H1, has been demonstrated in the Sun OS TCP mentioned earlier. Unfortunately, these tests revealed a somewhat surprising fact: old duplicate ACKs from the earlier incarnation can still slip past PAWS, so that (F2) will not prevent failures H2 or H3. What happens is that TIME-WAIT state effectively regenerates the timestamp of an old duplicate ACK. That is, when an old duplicate arrives in TIME-WAIT state, an extended TCP will send out its own ACK with a timestamp option containing its CURRENT timestamp clock value. If this happens immediately before the TWA mechanism kills TIME-WAIT state, the result will be a "new old duplicate" segment with a current timestamp that may pass the PAWS test on the reopened connection.

Whether H2 and H3 are critical depends upon how often they happen and what assumptions the applications make about TCP semantics. In the case of the H3 hazard, merely trying the open again is likely to succeed. Furthermore, many production TCPs have (despite the advice of the researchers who developed TCP) incorporated a "keep-alive" mechanism, which may kill connections unnecessarily. The frequency of occurrence of H2 and H3 may well be much lower than keep-alive failures or transient internet routing failures.

(F3) Use 64-bit Sequence Numbers

O'Malley and Peterson [RFC-1264] have suggested expansion of the TCP sequence space to 64 bits as an alternative to PAWS for avoiding the hazard of wrapped sequence numbers within the same incarnation. It is worthwhile to inquire whether 64-bit sequence numbers could be used to avoid the TWA hazards as well.

Using 64 bit sequence numbers would not prevent TWA - the early termination of TIME-WAIT state. However, it appears that a

combination of 64-bit sequence numbers with an appropriate modification of the TCP parameters could defeat all of the TWA hazards H1, H2, and H3. The basis for this is explained in an appendix to this memo. In summary, it could be arranged that the same sequence space would be reused only after a very long period of time, so every connection would be "slow" and "short".

4. Conclusions

Of the three fixes described in the previous section, fix (F1), ignoring RST segments in TIME-WAIT state, seems like the best short-term solution. It is certainly the simplest. It would be very desirable to do an extended test of this change in a production environment, to ensure there is no unexpected bad effect of ignoring RSTs in TIME-WAIT state.

Fix (F2) is more complex and is at best a partial fix. (F3), using 64-bit sequence numbers, would be a significant change in the protocol, and its implications need to be thoroughly understood. (F3) may turn out to be a long-term fix for the hazards discussed in this note.

APPENDIX: Using 64-bit Sequence Numbers

This appendix provides a justification of our statement that 64-bit sequence numbers could prevent the TWA hazards.

The theoretical ISN calculation used by TCP is:

    ISN = (R*T) mod 2**n.

where T is the real time in seconds (from an arbitrary origin, fixed when the system is started), R is a constant, currently 250 KBps, and n = 32 is the size of the sequence number field.

The limitations of current TCP are established by n, R, and the maximum segment lifetime MSL = 4 minutes. The shortest time Twrap to wrap the sequence space is:

    Twrap = (2**n)/r

where r is the maximum transfer rate. To avoid old duplicate segments in the same connection, we require that Twrap > MSL (in practice, we need Twrap >> MSL).

The clock-driven ISN numbers wrap in time TwrapISN:

    TwrapISN = (2**n)/R

For current TCP, TwrapISN = 4.55 hours.

The cases for old duplicates from previous connections can be divided into four regions along two dimensions:

* Slow vs. fast connections, corresponding to r < R or r >= R.

* Short vs. long connections, corresponding to duration E < TwrapISN or E >= TwrapISN.

On short slow connections, the clock-driven ISN selection rejects old duplicates. For all other cases, the TIME-WAIT delay of 2*MSL is required so old duplicates can expire before they infect a new incarnation. This is discussed in detail in the Appendix to [RFC-1185].

With this background, we can consider the effect of increasing n to 64. We would like to increase both R and TwrapISN far enough that all connections will be short and slow, i.e., so that the clock-driven ISN selection will reject all old duplicates. Put another way, we want to every connection to have a unique chunk of the seqence space. For this purpose, we need R larger than the maximum foreseeable rate r, and TwrapISN greater than the longest foreseeable connection duration E.

In fact, this appears feasible with n = 64 bits. Suppose that we use R = 2**33 Bps; this is approximately 8 gigabytes per second, a reasonable upper limit on throughput of a single TCP connection. Then TwrapISN = 68 years, a reasonable upper limit on TCP connection duration. Note that this particular choice of R corresponds to incrementing the ISN by 2**32 every 0.5 seconds, as would happen with the Berkeley BSD implementation of TCP. Then the low-order 32 bits of a 64-bit ISN would always be exactly zero.

REFERENCES

[RFC-793] Postel, J., "Transmission Control Protocol", RFC-793, USC/Information Sciences Institute, September 1981.

[RFC-1185] Jacobson, V., Braden, R., and Zhang, L., "TCP Extension for High-Speed Paths", RFC-1185, Lawrence Berkeley Labs, USC/Information Sciences Institute, and Xerox Palo Alto Research Center, October 1990.

[RFC-1263] O'Malley, S. and L. Peterson, "TCP Extensions Considered Harmful", RFC-1263, University of Arizona, October 1991.

[RFC-1323] Jacobson, V., Braden, R. and D. Borman "TCP Extensions for High Performance", RFC-1323, Lawrence Berkeley Labs, USC/Information Sciences Institute, and Cray Research, May 1992.

Security Considerations

Security issues are not discussed in this memo.

Author's Address:

Bob Braden
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (213) 822-1511
EMail: Braden@ISI.EDU