Network Working Group
Internet-Draft
Expires: April 24, 2005

K. Poon
Sun
N. Demizu
NICT
October 24, 2004

Use of TCP timestamp option to defend against blind spoofing attack
draft-poon-tcp-tstamp-mod-01.txt

Status of this Memo

This Internet-Draft will expire on April 24, 2005.

Copyright Notice

Abstract

The US-CERT alert (TA04-111A) shows that the well-known weakness in TCP's segment acceptance test is easier to exploit than previously thought. While there are already mechanisms, such as RFC 2385 for BGP and IPSEC, to defend against this kind of attack, we propose a light weight method making use of TCP timestamp (RFC 1323) option as an alternative.

Table of Contents

1. Requirements notation

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

As specified in [RFC793], TCP's segment acceptance test in ESTABLISHED state is based on the current expected receive sequence number RCV.NXT, the receive window RCV.WND, the received segment sequence number SEG.SEQ, and length SEG.LEN. For example, suppose TCP receives a spoofed RST (SEG.LEN is equal to 0), and the targeted connection's RCV.WND is 32K. As long as the SEG.SEQ of the RST is in [RCV.NXT, RCV.NXT + 32K), the received RST is considered acceptable. In this case, the TCP connection will be aborted. The probability of an attacker guessing the correct sequence number is thus RCV.WND / 2^32.

An attacker can also inject spoofed data to a TCP connection using a similar idea. In addition to causing data corruption, spoofed data injection may also lead to an ACK storm if the SEG.SEQ of the spoofed segment is of a proper value. As stated on Page 72 of RFC 793, if an ACK acknowledges some data never sent, TCP should send back an ACK and drop the ACK. For example, if the SEG.SEQ of the spoofed segment is RCV.NXT for receiver A, A will send back an ACK acknowledging that segment. A's peer B will send back an ACK, per Page 72 of RFC 793. But this ACK has SEG.SEQ equals to A's original RCV.NXT, which is not acceptable. A will then send back an ACK, per RFC 793. This will lead to an ACK storm.

As RCV.NXT of a connection is constantly changing, and RCV.WND for most TCP connections is small comparing to the entire sequence number space, it has been thought that as long as the initial sequence number is generated in a proper way, as described in [RFC1948], blind spoofing attack on this mechanism is not considered a serious threat. An example of a blind spoofing attack is to send a large number of RSTs with different sequence numbers spread over the sequence number space to a known TCP end point. If one RST is acceptable, the connection will be aborted resulting in a denial of service. This attack can also take advantage of the fact that some protocols use well known ports so that the attacker does not need to also search the port number space for the attack to be successful. A recent study described in NISCC Vulnerability Advisory 236929 [NISCC] has shown that this kind of attack is very easy to exploit in some type of TCP connection, such as the persistent connection used in routers supporting BGP [RFC1771]. The number of segments needed to have a successful attack is far less than previously thought. And as the receive window used in TCP connection is getting larger as network speed is getting faster, this TCP weakness also becomes easier to

exploit.

[RFC2385] describes a method of using a TCP MD5 signature option to strength the segment acceptance test. The option contains a MD5

digest of the segment and a key known only to the two end points of a TCP connection. The receiver can then use this option to verify the segment's acceptability. This method not only can defend against blind spoofing attack, it can also defend against attacker who can obtain segments exchanged in a connection. IPSEC [RFC2401] is another obvious choice to use for authentication. It not only protects TCP, but other protocols over IP as well.

In this document, we propose a light weight alternative to the above methods by making use of the TCP timestamp option [RFC1323]. The timestamp option already provides PAWS (Protect Against Wrapped Sequence numbers) and RTTM (Round Trip Time Measurement). It can also be used to strengthen the TCP segment acceptance test if the handling is modified a little.

3. Motivation

With the current TCP segment acceptance test, the probability of a successful blind spoofing attack is proportional to the receive window used. This assumes that an attacker can find out the ports and IP addresses being used in a connection. The obvious solution is to use cryptographic method to authenticate all TCP segments, as suggested in the other methods mentioned in the Introduction. Another possible solution to address this weakness is to put a "cookie" in every TCP segments. If the cookie is only known to the end points of a connection, it allows the receiver to verify that a segment indeed belongs to the connection by checking the segment's cookie. This is similar to the verification tag (vtag) in SCTP [RFC2960]. Note that the SCTP vtag is the same in every packets and its use is for verifying an incoming packet indeed belongs to the intended SCTP association. An attacker needs to find out the vtag of an association to spoof any packet. The issue with the cookie mechanism is that it does not protect against a man in the middle attack. Cryptographic methods, such as the TCP MD5 option, do not have this problem.

A cookie requires a new TCP option to hold the value. As the option space in a TCP header is limited, we look at the existing timestamp option to see if it can be used instead.

4. Basic Idea

As specified in RFC 1323, the TCP timestamp option contains two values, the sender's timestamp (TSval) and the echo reply (TSecr) of the timestamp received from its peer end point. From a receiver's view, the value of TSval in each incoming segment (SEG.TSval) is set in a monotonically increasing manner by the peer. Each value of SEG.TSval is thus greater than or equal to the value of the previous received segment unless reordering occurs or the TCP connection is idle for a significant amount of time (e.g., more than 24.8 days) such that the value wraps around. TSval is used for PAWS checking. The check is in section 4.2.1 of RFC 1323

> If SEG.TSval < TS.Recent and TS.Recent is valid, the segment is rejected.

The probability of guessing a valid SEG.TSval that passes the PAWS test is 1 in 2. Also see Appendix A about a specification inconsistency in [1323bis].

The value of TSecr in each incoming segment (SEG.TSecr) is the value of TSval in the last segment received by the peer that advances its RCV.NXT. The possible range of SEG.TSecr would vary from a few RTTs to a few seconds. Hence, the probability of guessing a SEG.TSecr that falls within this range would vary from 1 in 2^32 during idle time in a lossless network to around 1 in 2^20 when an ACK segment is lost, where RTO is 3 seconds and the timestamp clock frequency is 1 ms. The value of SEG.TSecr is not checked as specified in RFC 1323. TCP only uses SEG.TSecr to do RTTM.

While the TSval can be considered as an "extension" to the sequence number space, the TSecr may be considered as a "pseudo cookie." The reason is that a TCP end point should know exactly what TSval values it has used to send segments to its peer, and the TSecr in every incoming segments should only contain those values. This means that a TCP end point should be able to verify that a TSecr value is one of those TSval it has used. As noted above, the probability of guessing a TSecr to fall within this valid range is not high. So the TSecr value can be treated as a "pseudo cookie."

The timestamp option is normally used if the receive window of a connection is large, and this is when TCP is more vulnerable to blind

spoofing attack. This makes the use of timestamp option to protect against blind spoofing attack more attractive as it is "free."

4.1 Tracking the TSecr range

The first issue to resolve is how to track the TSecr valid range such

that a receiver can easily determine if an incoming segment is spoofed. We propose to add two new variables to the TCP connection state, TS.SndMax and TS.SndMin.

TS.SndMax holds the maximum value of TSval that has been used. It is set whenever a new data segment is sent to the peer. Since the value of TSecr in every segments is copied from TS.Recent on the peer node, and the value of TS.Recent is copied from TSval, TSecr in every received segments should never exceed TS.SndMax. TS.SndMax is the upper bound of valid TSecr values in every non-spoofed segments.

TS.SndMin holds the value of TSecr in every acceptable segments received which SEG.SEQ is equal to RCV.NXT. Its initial value is the value of TSval in the first segment sent, which is either the SYN or SYN|ACK segment. As TS.Recent in the peer node is monotonically increasing and TSecr is copied from TS.Recent, TS.SndMin is the lower bound of valid TSecr values in every non-spoofed segments.

With these two variables, a receiver can verify if an incoming segment is valid or not by the following condition

If TS.SndMin <= SEG.TSecr <= TS.SndMax, the segment is accepted.

We refer the above condition as PASA (Protection Against Spoofing Attack) [PASA] test.

4.2 Narrowing Down the Valid TSecr Range

To make the PASA test more effective, the valid range of TSecr must be as narrow as possible. Because of a specification inconsistency (refer to Appendix A), there can be more than one way by which TS.Recent is updated. For example, if an implementation follows section 3.4 of 1323bis, TSval in pure ACK segment can be used to update TS.Recent. But an implementation following RFC 1323 does not allow this. This causes different dynamics in the valid TSecr range and makes narrowing down the range difficult. Refer to Appendix B for a detailed explanation on how the different ways of updating TS.Recent can cause problems with the PASA test.

As a sender can control what TSval to put in an outgoing segment, we

propose to change the rule in generating TSval so that the PASA test can work well with different implementations using different methods of updating TS.Recent. The new rule is

> For TSval on segments with SEG.LEN > 0, use the current timestamp clock as specified in RFC 1323. For TSval on segments with SEG.LEN = 0, use the last value of TSval sent to the peer.

Making use of the newly introduced TS.SndMax, the above rule is equivalent to

> When sending a segment with SEG.LEN > 0, set TS.SndMax to the current timestamp clock. Use this TS.SndMax as TSval in the outgoing segment.
>
> Use TS.SndMax as TSval on segment with SEG.LEN = 0.

Using this new rule, only TSval in a segment with SEG.LEN > 0 will update the TS.Recent value as other segments contain an old TSval. This rule narrows down the valid range of TSecr. Note that it does not affect RTTM as only TSecr in an incoming segment which acknowledges data is used to make measurement. This means that only the TSval in a segment with SEG.LEN > 0 is needed to update TS.Recent for RTTM. Also see Appendix C for another proposal on how to deal with this problem.

4.3 Unpredictable Timestamp

To make PASA robust, it is important to keep the value of TSval used unpredictable from a malicious, off-path third party. Normally, the SEG.TSval contains the value of the local timestamp clock (my.TSclock) when a segment is sent. If a node uses one global timestamp clock as the my.TSclock for all TCP connections, it will be easy for a malicious, off-path third party to guess the valid value for TSecr. This is because the current TSval could easily be obtained by establishing another TCP connection with the target node.

To defend against such attack, we propose to have a random offset to the timestamp clock for each connection or destination. The TCP connection state is augmented by one 32-bit unsigned integer, TS.SndOff. Each TSval is then calculated as my.TSclock + TS.SndOff, instead of using my.TSclock directly.

The value of TS.SndOff may be a pseudo-random number or the result of F(local-node, local-port, remote-node, remote-port) where F is a cryptographic hash function. The latter is similar to the method of choosing a good initial sequence number as discussed in RFC 1948.

If a new TCP connection is opened with the same four tuple of addresses of an existing TCP connection in TIME-WAIT state, the TS.SndOff should be carefully chosen so that:

1. Duplicate delayed segments are not accepted as valid segments.

2. It is hard to infer the new valid range of TSecr from the connection state of the previous TCP connection. This is to

avoid spoofing attack by the previous address holder in a DHCP or mobile environment.

We suggest adding a random number to the existing TS.SndOff instead of assigning a newly generated random number to TS.SndOff.

4.4 TSval to use after the connection is idle

When a data segment is sent after the connection is idle for a long period of time, TS.SndMax is set to the value of the current my.TSclock, and the difference between TS.SndMax and TS.SndMin will be very large until an ACK segment is returned. If the data segment or the ACK segment in reply is lost, TS.SndMin will be unchanged until the data segment sent is acknowledged after retransmission timeout. During the period when the difference is large, the probability of guessing a valid TSecr will also be large. Therefore, a method of keeping the difference small is desired. Note that this window of vulnerability is normally only a couple of RTOs.

We introduce an upper limit to the advancement of TSval. If no data segment has been sent to the peer for more than the upper limit of time, then the next value of TSval used should be TS.SndMax plus the upper limit instead of the value of my.TSclock + TS.SndOff (Section 4.3). Note that TS.SndMax is still monotonically increasing with this modification.

The upper limit should be long enough (at least longer than both RTT and MSL) so that there would be no side effect on other mechanisms, such as RTTM, PAWS, and Eifel [RFC3522]. A possible candidate for the upper limit will be ten minutes. This solution can easily be implemented by tweaking TS.SndOff as follows.

The TCP connection state is augmented by a new variable, TS.MaxAdv, which is the upper limit of the advancement of TSval. Before sending a segment (only when TS.SndMax is updated),

```
if (my.TSclock - TS.SndMax > TS.MaxAdv) then
    TS.SndOff = TS.MaxAdv - (my.TSclock - TS.SndMax)
```

Since each TSval is calculated as my.TSclock + TS.SndOff (refer to

Section 4.3), when no segment is sent for longer than TS.MaxAdv, the next TSval value used will be

```
TSval = TS.SndMax
      = my.TSclock + TS.SndOff
      = my.TSclock + (TS.MaxAdv - (my.TSclock -
        old_TS.SndMax))
      = old_TS.SndMax + TS.MaxAdv
```

4.5 RST Handling

On Page 18 of RFC 1323, it is recommended that a RST should not contain a timestamp option and a TCP stack should ignore the timestamp option of an incoming segment if it is a RST. To make use of timestamp option for PASA test, a RST needs to carry the timestamp option. We propose to change the way RST is generated in all TCP states as follows.

If a RST is sent because of one of the reasons stated in RFC 793 for an incoming segment and it has a timetstamp option, the RST MUST be

If the ACK bit is off, sequence number zero is used,

```
<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>
<TSval=SEG.TSecr><TSecr=SEG.TSval>
```

If the ACK bit is on,

```
<SEQ=SEG.ACK><CTL=RST><TSval=SEG.TSecr><TSecr=SEG.TSval>
```

If the incoming segment does not contain a timestamp option, send a RST without timestamp option according to RFC 793.

If a RST is sent because a connection is aborted, the RST MUST be

```
<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>
<TSval=TS.SndMax><TSecr=TS.Recent>
```

With the above changes, a valid RST containing the above TSecr will pass the PASA test and be accepted. And it is hard for an attacker to spoof such a RST segment. The problem with RST handling is that the TCP stack cannot know if its peer has this modification or not. If it receives a RST without a timestamp option, should the RST be accepted or not? This issue is discussed in the next section Section 4.6.

4.6 Level of Protection

Level 0 - No Protection

The PASA test requires the use of timestamp option. If the timestamp option is not used for a connection, there is no protection at all.

Level 1 - Protection against spoofed segment, except RST

If the timestamps option is used for a connection, a node implementing the PASA test is protected against "any" spoofed segment

except RST. As discussed in Section 4.5, the problem with RST is that it requires the peer to be modified for RST segment handling also. There is currently no way to communicate with the peer about this information.

One solution is to introduce a new TCP option called PASA-OK (refer to Appendix D). Its purpose is to indicate that the sender of the PASA-OK option is modified to do what Section 4.5 suggests. Then the TCP stack can know if the PASA test can be used to test against RST segment. This leads to the next level of protection.

Level 2 - Protection against any spoofed segment

If both sides of a TCP connection know that its peer is PASA capable by using the PASA-OK option, then "no" spoofed segment will be accepted. If PASA-OK is not supported in either the local or peer node, a TCP stack SHOULD allow an application to specify that any RST segment not containing a timestamp option to be dropped. In this way, the TCP connection is protected against any spoofed segment. The only catch is that if the peer is not modified to do the RST handling in Section 4.5, the application SHOULD have some form of keep-alive probing mechanism to check the status of its peer.

A PASA capable implementation SHOULD allow an application to specify the required level of protection for individual TCP connection. If required level cannot be met, the TCP connection SHOULD be aborted and a RST segment as described in Section 4.5 SHOULD be sent to its peer. If the application specifies a protection level 2 and the peer is not PASA capable (because PASA-OK option is not implemented in either the local or peer node), the stack SHOULD let the application know about this. The application can then decide whether to abort this connection.

5. Modified Timestamp Option Handling

A TCP implementation MAY modify the timestamp option handling to support PASA as described below. If PASA is used, the implementation MUST provide a mechanism, such as a TCP socket option, for this handling to be turned on or off for individual TCP connection.

The TCP connection state SHOULD be augmented by four variables: TS.SndMax, TS.SndMin, TS.SndOff, and TS.MaxAdv. They should be initialized as follows. Refer to Section 4.1, Section 4.3, and Section 4.4.

```
TS.SndOff = random number
TS.SndMin = my.TSclock + TS.SndOff
TS.SndMax = my.TSclock + TS.SndOff
TS.MaxAdv = the upper limit to the advancement of TSval
```

5.1 Segment Sending

When sending:

1. SYN or SYN|ACK, follow Appendix D and Section 4.6 if an application specifies a protection level.
2. RST, follow Section 4.5.
3. All other segments,

```
if (SEG.LEN > 0) {
    if (my.TSclock - TS.SndMax > TS.MaxAdv)
        TS.SndOff = TS.MaxAdv - ( my.TSclock - TS.SndMax );
    TS.SndMax = my.TSclock + TS.SndOff;
}
```

```
<SEQ=SNT.NXT><ACK=RCV.NXT><CTL=ACK>
<TSval=TS.SndMax><TSecr=TS.Recent>
```

5.2 Segment Receiving

If a TCP connection cannot be found for an incoming segment, send back a RST according to Section 4.5. If an application specifies protection level 0, follow the receive processing of RFC 1323. Otherwise, perform the following checks.

5.2.1 LISTEN, and SYN-SENT States

Follow the receive processing as in RFC 793. After the segment is accepted, it SHOULD be tested for whether it satisfies the protection

level required by the application (either 1 or 2). This means that the received segment MUST carry the TCP Timestamps option. If not, send back a RST according to Section 4.5. If the TCP is in SYN-SENT state, the connection should be aborted.

5.2.2 Other States

The incoming segment is a RST.

1. Protection level is 1: Follow the RFC 1323 receive processing.
2. Protection level is 2: If the segment does not contain the timestamp option, drop it. Otherwise, perform the PASA test on the. If the test fails, drop the RST segment. If not, follow the RFC 1323 receive processing.

The incoming segment is not a RST.

If the segment does not contain the timestamp option, drop it. Otherwise, perform the PASA test on the incoming segment. If the test fails, an ACK segment SHOULD be sent in reply as specified on page 69 of RFC 793. Then drop the segment. Note that this rule is the same as that of PAWS in RFC 1323.

If the PASA test succeeds,

```
        if (TS.SndMin < SEG.TSecr && SEG.SEQ == RCV.NXT)
            TS.SndMin = SEG.TSecr;
```

Follow RFC 1323 for the rest of processing.

6. Security Considerations

The proposed method in this document is not a replacement of other cryptographic mechanisms for authenticating TCP segments. It only reduces the probability of a successful blind spoofing attack provided that the attacker cannot obtain segments exchanged between the two TCP end points of a connection.

7. Acknowledgement

The idea of using TCP timestamp option originates from discussion on the TCPM WG mailing list [TCPM].

8 References

[1323bis] Jacobson, V., Braden, B. and D. Borman, "TCP Extensions for High Performance", (work in progress) draft-jacobson-tsvwg-1323bis-00.txt, August 2003.

[NISCC] National Infrastructure Security Co-ordination Centre, "NISCC Vulnerability Advisory 236929", April 2004, <http://www.uniras.gov.uk/vuls/2004/236929/index.htm>.

[PASA] Demizu, N., "Protection Against Spoofing Attacks by Using the TCP Timestamps Option", (work in progress) draft-demizu-tcpm-pasa-issues-00.txt, October 2004.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", RFC 1122, October 1989.

[RFC1323] Jacobson, V., Braden, B. and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC1771] Rekhter, Y. and T. Li, "A Border Gateway Protocol 4 (BGP-4)", RFC 1771, March 1995.

[RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, May 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[TCPM] IETF TCPM Working Group, "http://www.ietf.org/html.charters/tcpm-charter.html", April 2004.

Authors' Addresses

Kacheong Poon
Sun Microsystems, Inc.
M/S USJC01-201
4150 Network Circle
Santa Clara, CA 95054
USA

EMail: kacheong.poon@sun.com

Noritoshi Demizu
National Institute of Information and Communications Technology
4-2-1 Nukui-Kitamachi, Koganei
Tokyo 184-8795
Japan

EMail: demizu@nict.go.jp

Appendix A. Timestamp Specification Inconsistency

When RFC 1323 was updated by 1323bis, one important change was when to update TS.Recent as there was an inconsistency in RFC 1323. Section 3.4 of RFC 1323 specifies that if the following condition is true, TS.Recent is updated.

```
SEG.SEQ <= Last.ACK.sent < SEG.SEQ + SEG.LEN          - (A)
```

But in section 4.2.1 of RFC 1323, it uses this condition instead.

```
SEG.SEQ <= Last.ACK.sent
```

In 1323bis, this inconsistency was clarified and the condition was updated as

```
SEG.TSval >= TS.Recent and SEG.SEQ <= Last.ACK.sent    - (B)
```

There is still a problem. Note that there is a step (R2) in section 4.2 of both RFC 1323 and 1323bis. (R2) specifies that if an incoming segment is not inside the receive window, the segment is rejected. This is done before the condition comparing SEG.SEQ and Last.ACK.Sent is checked. In effect, if (R2) is done, the condition (B) will become

```
if (SEG.LEN > 0)
    SEG.SEQ <= Last.ACK.sent <= RCV.NXT < SEG.SEQ + SEG.LEN
else if (SEG.LEN == 0)
    SEG.SEQ == Last.ACK.sent == RCV.NXT               - (B')
```

Another confusing fact is that the step (R2) is not shown in the Appendix E PSEUDO-CODE SUMMARY of 1323bis. Because of the above, an implementation may choose any one of the conditions for updating TS.Recent.

As commented in Appendix C of 1323bis, it is good to be able to use the TSval in "a retransmitted segment that resulted from a lost ACK" (meaning duplicate segment). Both (A) and (B') do not allow this. We propose to change the condition to be

```
SEG.TSval >= TS.Recent and
RCV.NXT - RCV.WND <= SEG.SEQ <= Last.ACK.sent
```

Using this condition, TS.Recent is also updated by duplicate segments.

Appendix B. PASA Issues with Different TS.Recent Update Methods

As noted in Appendix A, there are different methods in updating TS.Recent. As TSecr in a segment is copied from TS.Recent, this means that the valid range of TSecr can be different if the peer uses different methods to update TS.Recent. This affects the effectiveness of the PASA test.

If a peer node uses condition (A) in Appendix A to update TS.Recent, the valid range of SEG.TSecr would be within a few RTTs almost all the time. The reason is that TS.Recent on the peer node is updated only by new data segments. It is never updated by duplicate data segments, window updates, or keep-alive segments. Therefore, in cases where no segment is lost, the difference between TS.SndMax and TS.SndMin will be at most around one RTT. And when a TCP connection becomes idle, the difference will typically be zero. Even in cases where some segments are lost, if the losses are recovered soon, say within a few RTTs, the difference will be at most around a few RTTs.

If a peer node uses condition (B), the valid range of SEG.TSecr can be very large for a long period of time. The reason is that TS.Recent on the peer node is updated by non-data segments, such as window updates and keep-alive probes. Assuming that the TCP also follows the rule specified in RFC 1323 in generating TSval in an outgoing segment. This means that TS.SndMax is updated whenever any segment is sent. When a non-data segment is sent, TS.SndMin may not be updated soon because no segment may be sent back in reaction to that non-data segment. Consequently, the difference between TS.SndMax and TS.SndMin can become very large for a long period of time. The following are examples of scenarios where the valid range of TSecr is very large.

Note: In the examples below, TS.SndMax is updated whenever any segment is sent.

Example 1 - Window updates

Window updates can be delayed for a long period of time, because they are sent when an application reads data. For example, when an application is displaying a modal dialog and waiting for a user's input, it often does not read the received data until the modal dialog is closed. As another example, if a user suspends an

application for a long time, it cannot read received data during the suspended period. In such cases, upon sending a window update, TS.SndMax is advanced while TS.SndMin stays unchanged. The difference between TS.SndMax and TS.SndMin can become very large. And TS.SndMax and TS.SndMin will stay unchanged until some segments are exchanged.

Example 2 - TCP receives a keep-alive probe from its peer

The interval between TCP keep-alive probes is typically a couple of hours. When a peer node sends a TCP keep-alive probe, the TSecr in the probe segment contains an old timestamp near the value of TS.SndMin, which is a couple of hours earlier. TCP responds to the probe with an ACK segment which has a TSval equals to the current timestamp clock, TS.SndMax also needs to be updated to that value. But TS.SndMin is not updated in this scenario. Hence, the difference between TS.SndMax and TS.SndMin will become a couple of hours until some segments are exchanged afterwards.

Example 3 - TCP sends a keep-alive probe to its peer

In the case when TCP sends a keep-alive probe to its peer, if the keep-alive probe or its corresponding ACK segment is lost, the difference between TS.SndMax and TS.SndMin will be a couple of hours until the TCP keep-alive procedure finishes successfully after retransmitting the probe. At that time, TS.SndMax and TS.SndMin will become the same again. Fortunately, this should normally take a couple of RTTs.

Appendix C. New Condition for Updating TS.Recent

As seen in Appendix B, the essential difference between conditions (A) and (B) is that TS.Recent in a node using (A) is updated only by new data segments while TS.Recent in a node using (B) can be updated by duplicate data segments and non-data segments. Taking into account of (B') in Appendix A, we propose the following condition to replace both (A) and (B)

```
RCV.NXT - RCV.WND <= SEG.SEQ <= Last.ACK.sent &&
SEG.LEN > 0                                          - (C)
```

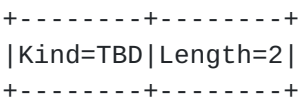
If a peer node uses (C), the valid range of SEG.TSecr would be narrower than that of a peer node using (A) or (B). It does not have the problems discussed in Appendix B. Furthermore, (C) also provides RTT measurements for retransmitted segments like (B).

Using condition (C) to update TS.Recent can be considered as an alternative to the method proposed in Section 4.2. The problem with this is that it requires all peer nodes to be modified. This is a difficult deployment issue to overcome.

Appendix D. TCP PASA-OK Option

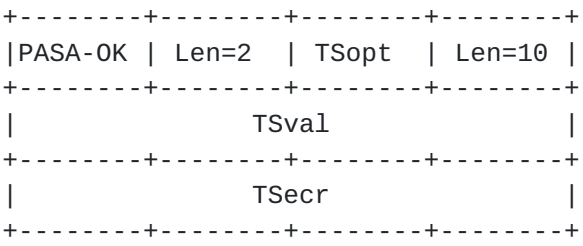
The TCP PASA-OK option indicates that the sender supports PASA. This option MAY be sent on a SYN segment. It also MAY be sent on a SYN|ACK segment, but only if the option is received in the corresponding SYN segment. This option SHOULD not be sent in other segments. As discussed in Section 4.6, this option is not mandatory in order to provide protection level 2.

The format of the TCP PASA-OK option is



If the TCP PASA-OK option is sent on a SYN or SYN|ACK segment, the TCP Timestamps option MUST be sent on the same segment.

Implementation note: These two TCP options can be sent in the following format.



In the above diagram, the TSval is equal to TS.SndMax. The TSecr is equal to 0 if it is a SYN segment. If it is a SYN|ACK segment, the TSecr is equal to the TSval in the corresponding SYN segment.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.