Network Working Group   M. Dalal
Internet-Draft   Editor
Expires: May 23, 2005   November 22, 2004

Transmission Control Protocol security considerations
draft-ietf-tcpm-tcpsecure-02.txt

Status of this Memo

Copyright Notice

Abstract

TCP (RFC793 [1]) is widely deployed and one of the most often used reliable end to end protocols for data communication. Yet when it was defined over 20 years ago the internet, as we know it, was a different place lacking many of the threats that are now common. Recently several rather serious threats have been detailed that can pose new methods for both denial of service and possibly data injection by blind attackers. This document details those threats and also proposes some small changes to the way TCP handles inbound

segments that either eliminate the threats or at least minimize them to a more acceptable level.

Table of Contents

1. Introduction

TCP (RFC793 [1]) is widely deployed and one of the most often used reliable end to end protocols for data communication. Yet when it was defined over 20 years ago the internet, as we know it, was a different place lacking many of the threats that are now common. Recently several rather serious threats have been detailed that can pose new methods for both denial of service and possibly data injection by blind attackers. This document details those threats and also proposes some small changes to the way TCP handles inbound segments that either eliminate the threats or at least minimize them to a more acceptable level.

Most of these proposals modify handling procedures for DATA, RST and SYN's as defined in RFC793 [1] but do not cause interoperability issues. The authors feel that many of the changes proposed in this document would, if TCP were being standardized today, be required to be in the base TCP document and the lack of these procedures is more an artifact of the time when TCP was developed than any strict requirement of the protocol.

For some uses of TCP, an alternative protection against the threats that these changes address has already been implemented and deployed in the TCP MD5 Signature Option (RFC2385 [2]). Because this option is not negotiated and is implemented with a manually established shared key or password, it has been used for protecting uses of TCP in which the endpoints are managed, such as for BGP peers. RFC3562 [2] provides importance guidance for users of RFC2385 [2] for decreasing their vulnerability to key-guessing.

Yet another commonly known mitigation technique is cryptography, especially IPSec. For IPSec to work, both ends of the connection need to agree on the properties to use for the connection and also agree upon a pre-shared key. In the absence of PKI infrastr1ucture, this may be inconvenient. IPSec with manual keys can be used to avoid using ISAKMP. However, this adds considerable burden on the administration of such solution. If ISAKMP were to be used, this would typically require all firewalls in the path between the two TCP endpoints to allow UDP traffic for atleast ISAKMP to function. Further, IPSec and NAT have long been known to have interoperability issues.

TCP implementations SHOULD also introduce ephemeral port randomization. By randomizing ephemeral ports an attacker would have a less easy time in guessing the four tuples needed to mount a successful attack. Since ephemeral ports are 16 bit values and are a subset of the entire available port numbers, it is a weaker defense than an exact sequence number match as proposed here which is a

32-bit value and changes dramatically within the life of a connection. Nevertheless, both of them are complimentary solutions that will make it difficult to launch attacks discussed below.

Alternative proposals, including the use of cookies (or, use of the timestamp option as a cookie) require both peers to implement the changes before any additional protection can be realized.

2. Blind reset attack using the RST bit

2.1 Description of the attack

It has been traditionally thought that for a blind attacker to reset a TCP connection the attacker would have to guess a single sequence number in the TCP sequence space. This would in effect require an attacker to generate (2^^32) segments in order to reset a connection. Recent papers have shown this to not necessarily be the case. An attacker need only guess a number that lies between the last sequence number acknowledged and the last sequence number acknowledged added to the receiver window (RCV.WND)[4]. Modern operating systems normally default the RCV.WND to about 32,768 bytes. This means that a blind attacker need only guess 65,535 RST segments (2^^32/RCV.WND) in order to reset a connection. At DSL speeds this means that most connections (assuming the attacker can accurately guess both ports) can be reset in under 200 seconds (usually far less). With the rise of broadband availability and increasing available bandwidth, many Operating Systems have raised their default RCV.WND to as much as 64k, thus making these attacks even easier.

2.2 Solution

RFC793 [1] currently requires handling of a segment with the RST bit when in a synchronized state to be processed as follows:

1) If the RST bit is set and the sequence number is outside the expected window, silently drop the segment.
2) If the RST bit is set and the sequence number is acceptable i.e.: (RCV.NXT <= SEG.SEQ <= RCV.NXT+RCV.WND) then reset the connection.

Instead, the following changes should be made to provide some protection against such an attack.

A) If the RST bit is set and the sequence number is outside the expected window, silently drop the segment.
B) If the RST bit is set and the sequence number exactly matches the next expected sequence number, reset the connection.
C) If the RST bit is set and the sequence number does not exactly match the next expected sequence value, yet is within the acceptable window (RCV.NXT < SEG.SEQ <= RCV.NXT+RCV.WND) send an acknowledgment:
   <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
   After sending the acknowledgment, drop the unacceptable segment and return.
   This solution forms a challenge/response with any RST where the value does not exactly match the expected value and yet the RST is within the window. In cases of a legitimate reset without the exact sequence number, the consequences of this new challenge/response will be that the peer requires an extra round

trip time before the connection can be reset.

In order to alleviate multiple RSTs/SYNs from triggering multiple challenge ACKs, a ACK throttling mechanism SHOULD be implemented. Suggested values are to send no more than 10 challenge ACKs in a 5 second window. These values MUST be tunable to accomodate different requirements. ACK throttling if implemented successfully can lead to several advantages. Besides preventing the RST/ACK war outlined in the section below, it can also alleviate spurious fast retransmits at the remote end caused by flood of duplicate ACKs and also save spurious processing required to send an ACK on the victim side.

3. Blind reset attack using the SYN bit

3.1 Description of the attack

Analysis of the reset attack, which uses the RST flag bit, highlights another possible avenue for a blind attacker. Instead of using the RST bit an attacker can use the SYN bit as well to tear down a connection. Using the same guessing technique, repeated SYN's can be generated with sequence numbers incrementing by an amount not larger than the window size apart and thus eventually cause the connection to be terminated.

3.2 Solution

RFC793 [1] currently requires handling of a segment with the SYN bit set in the synchronized state to be as follows:

1) If the SYN bit is set and the sequence number is outside the expected window, send an ACK back to the sender.
2) If the SYN bit is set and the sequence number is acceptable i.e.: (RCV.NXT <= SEG.SEQ <= RCV.NXT+RCV.WND) then send a RST segment to the sender.

Instead, changing the handling of the SYN to the following will provide complete protection from this attack:

1) If the SYN bit is set, irrespective of the sequence number, send an ACK to the remote peer:
   <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
   After sending the acknowledgment, drop the unacceptable segment and return.
   This solution agains forms a challenge response with the peer as in the previous section.

By always sending an ACK to the sender, a challenge/response is setup with the peer. A legitimate peer, after restart, would not have a TCB in the synchronized state. Thus when the ACK arrives the peer should send a RST segment with the sequence number derived from the ACK field that caused the RST.

Note, there is one corner case for the SYN attack problem that will prevent the successful reset of the connection. This is a result of the RFC793 [1] specification and is nothing to do with the proposed solution. In this problem, if a restarting host generates a SYN with an initial sequence number that is exactly equal to RCV.NXT - 1 of the remote TCP endpoint that is still in the established state and if the SYN arrives at the peer that is still holding the stale connection, an ACK will be generated. This ACK will have an ack value of RCV.NXT and will be acceptable to the restarting host which will accept the ACK and do nothing. The SYN will then be

retransmitted and the behavior will repeat. This could lead to an initialization failure. Subsequent connection attempts will hopefully succeed by choosing a new ISN that is not equal to RCV.NXT - 1. A similar problem will be seen should the SYN contain data.

4. Blind data injection attack

4.1 Description of the attack

A third type of attack is also highlighted by both the RST and SYN attacks. It is quite possible to inject data into a TCP connection by simply guessing a sequence value that is within the window. The ACK value of any data segment is considered valid as long as it does not acknowledge data ahead of the next segment to send. In other words an ACK value is acceptable if it is (SND.UNA-(2^^31-1)) <= SEG.ACK <= SND.NXT). This means that an attacker simply need guess two ACK values with every guessed sequence number so that the chances of successfully injecting data into a connection are 1 in ((2^^32 / RCV.WND) * 2).

When an attacker successfully injects data into a connection the data will sit in the receiver's re-assembly queue until the peer sends enough data to bridge the gap between the RCV.NXT value and the injected data. At that point one of two things will occur either:
a) An ACK war will ensue with the receiver indicating that it has received data up until RCV.NXT (which includes the attackers data) and the sender sending a corrective ACK with a value less than RCV.NXT (the real sequence number of the last byte sent).
b) The sender will send enough data to the peer which will move RCV.NXT even further along past the injected data.
In either case the injected data will be made readable to the upper layer and in case <a> the connection will eventually be reset by one of the sides. Note that the protections illustrated in this section neither cause an ACK war nor prevent one from occurring if data is actually injected into a connection. The ACK war is a natural consequence of any data injection that is sucessful.

4.2 Solution

An additional input check should be added to any incoming segment. The ACK value should be acceptable only if it is in the range of ((SND.UNA - MAX.SND.WND) <= SEG.ACK <= SND.NXT). MAX.SND.WND is defined as the largest window that the local receiver has ever advertised to it's peer. This window is the scaled value i.e. the value may be larger than 65,535 bytes. This small check will greatly reduce the vulnerability of an attacker guessing a valid sequence number since not only must he/she guess the sequence number in window, but must also guess a proper ACK value within a scoped range. This solution reduces but does not eliminate the ability to generate false segments. It does however reduce the probability that invalid data will be injected to a more acceptable level. For those applications that wish to close this attack completely RFC2385 [2] should be deployed between the two endpoints.

5. Backward Compatibility and Other considerations

1) The proposed solution is backward compatible as it uses the semantics laid down in RFC793 [1] to defend against it's own weakness. Referring to the figure below, if we assume that the RST (1.c) was in flight when the ACK (2) left TCP A, TCP B has no way of knowing what triggered the ACK. For all it cares, the ACK might have been a result of the data or the RST that it had sent earlier. Hence in either case, TCP B must reply to this ACK with an appropriate RST that is in keeping with RFC793 [1].

2) Concerns have been raised that the challenge response mechanism will lead to a reflector kind of attack. In this attack, it is believed that an attacker with higher bandwidth can potentially spoof SYN or RST packets within the window and cause ACK flooding to a remote peer that may have a lower bandwidth. These concerns are misplaced because it is trivial to cause a victim to generate an ACK. A spoofer can simply send packets with sequence numbers that are outside the acceptable window of the attacker or send an ACK that acknowledges something that is not yet sent. Further, an attacker can also simply generate data packets that fall within the window to cause an ACK to be sent. RFC793 [1] also mandates that an ACK be sent if the incoming SYN to an established connection falls outside the acceptable window. All these scenarios can be used to launch a flood attack. However, the potential harm of such attacks are low and can be easily detected due to the volume of packets generated. The latter is a strong deterrent to such attacks.

3) There is a corner scenario in the above proposed solutions which will require more than one round trip time to successfully abort the connection as per the figure below. This scenario is similar to the one in which the original RST was lost in the network.

```
       TCP A                                                 TCP B
1.a. ESTABLISHED  <-- <SEQ=300><ACK=101><CTL=ACK><DATA> <--  ESTABLISHED
  b. (delayed)    ... <SEQ=400><ACK=101><CTL=ACK><DATA> <--  ESTABLISHED
  c. (in flight)  ... <SEQ=500><ACK=101><CTL=RST>       <--  CLOSED
2.   ESTABLISHED  --> <SEQ=101><ACK=400><CTL=ACK>       -->  CLOSED
    (ACK for 1.a)
                  ... <SEQ=400><ACK=0><CTL=RST>         <--  CLOSED
3.   CHALLENGE    --> <SEQ=101><ACK=400><CTL=ACK>       -->  CLOSED
     (for 1.c)
                  ... <SEQ=400><ACK=0><CTL=RST>         <--  RESPONSE
4.a. ESTABLISHED  <-- <SEQ=400><ACK=101><CTL=ACK><DATA>      1.b reaches A
  b. ESTABLISHED  --> <SEQ=101><ACK=500><CTL=ACK>
  c. (in flight)  ... <SEQ=500><ACK=0><CTL=RST>         <--  CLOSED
5.   RESPONSE arrives at A, but is dropped because of being out of window.
```

```
6.   ESTABLISHED  <-- <SEQ=500><ACK=0><CTL=RST>             4.c reaches A
7.   CLOSED                                                CLOSE
```

4) For the solution to be totally effective against the vulnerabilities discussed in this document, both ends of the TCP connection need to have the fix. Although, having it at one end might prevent that end from being exposed to the attack, the connection is still vulnerable at the other end.

6. Middlebox considerations

The following scenarios were brought to notice by the tcpm working group members as middlebox issues which may cause the proposed solution to behave in an unexpected manner.

6.1 Middlebox that cache RST's

Consider a middlebox B tracking connection between two TCP endhosts A and C. If C sends a RST with a sequence number that is within the window but not an exact match to reset the connection and if B does not have the fix proposed here, it will clear the connection and ask A to do the same. If A does not have the fix it will clear the connection and everything will be fine. However if A does have the proposed fix above, it will send a challenge ACK. B being a middlebox will intercept this ACK and resend the RST cached earlier that was responsible for bringing down the connection. However, the RST will again be not acceptable and will trigger a challenge ACK again. This will cause a RST/ACK war to happen. However, we are not aware of middleboxes that actually do this and believe the design itself is flawed in that given the scenario that the RST from B to A got lost on the way, A will continue to hold the connection and A might send an ACK an arbitrary time after the connection was brought down at B. In this case, B will have to cache the RST for an arbitrary amount of time till it's confirmed that the connection has been cleared at A.

6.2 Middleboxes that advance sequence numbers

Some Middleboxes may compute RST sequence numbers at the higher end of the acceptable window. The setup is the same as the earlier case, but in this case instead of sending the cached RST, the middlebox sends a RST that computes it's sequence number as a sum of the ack field in the ACK and the window advertised by the ACK that was sent by A to challenge the RST as depicted below. The difference in the sequence numbers between step 1 and 2 below is due to data lost in the network.

```
   TCP A                                                     Middlebox

1. ESTABLISHED  <-- <SEQ=500><ACK=100><CTL=RST>          <--  CLOSED

2. ESTABLISHED  --> <SEQ=100><ACK=300><WND=500><CTL=ACK> -->  CLOSED

3. ESTABLISHED  <-- <SEQ=800><ACK=100><CTL=RST>          <--  CLOSED

4. ESTABLISHED  --> <SEQ=100><ACK=300><WND=500><CTL=ACK> -->  CLOSED
```

```
5. ESTABLISHED  <-- <SEQ=800><ACK=100><CTL=RST>          <--  CLOSED
```

Although the authors are not aware of a working implementation that does the above, it could be mitigated by implementing the RST throttling mechanism described earlier.

7. Contributors

Mitesh Dalal and Amol Khare of Cisco Systems came up with the solution for the RST/SYN attacks. Anantha Ramaiah and Randall Stewart of Cisco Systems discovered the data injection vulnerability and together with Patrick Mahan and Peter Lei of Cisco Systems found solutions for the same. Paul Goyette, Mark Baushke, Frank Kastenholz, Art Stine and David Wang of Juniper Networks provided the insight that apart from RSTs, SYNs could also result in formidable attacks. Shrirang Bage of Cisco Systems, Qing Li and Preety Puri of Wind River Systems and Xiaodan Tang of QNX Software along with the folks above helped in ratifying and testing the interoperability of the suggested solutions.

8. Acknowledgments

Special thanks to Sharad Ahlawat, Mark Allman, Steve Bellovin, Vern Paxson, Allison Mankin, Damir Rajnovic, John Wong and the tcpm WG members for suggestions and comments.

9. References

9.1 Normative References

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

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

9.2 Informative References

[3] Leech, M., "Key Management Considerations for the TCP MD5 Signature Option", RFC 3562, July 2003.

[4] Watson, P., ""Slipping in the Window: TCP Reset attacks"".

Author's Address

Mitesh Dalal
Editor
170 Tasman Drive
San Jose, CA 95134
USA

Phone: +1-408-853-5257
EMail: mdalal@cisco.com

Acknowledgment

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