Network Working Group
Internet-Draft
Expires: August 17, 2006

R. Stewart
M. Dalal
Editor

February 13, 2006

Improving TCP's Robustness to Blind In-Window Attacks
draft-ietf-tcpm-tcpsecure-04.txt

Status of this Memo

Copyright Notice

Abstract

A recent study indicates that some types of TCP connections have an increased vulnerability to spoofed packet injection attacks than previously believed [SITW]. TCP has historically been considered protected against spoofed packet injection attacks by relying on the fact that it is difficult to guess the 4-tuple (the source and destination IP addresses and the source and destination ports) in combination with the 32 bit sequence number(s). A combination of increasing window sizes and applications using a longer term

connections (e.g. H-323 or Border Gateway Protocol [RFC1771]) have left modern TCP implementation more vulnerable to these types of spoofed packet injection attacks.

Note: Both [SITW] and [DTASA] provide charts which can give the reader an idea as to the time it takes to penetrate an unprotected system.

Many of these long term TCP applications tend to have predictable IP addresses and ports which makes it far easier for the 4-tuple to be guessed. Having guessed the 4-tuple correctly, an attacker can inject a RST, SYN or DATA segment into a TCP connection by carefly crafting the sequence number of the spoofed segment to be in the current receive window. This can cause the connection to either abort or possibly cause data corruption. This document proposes small modifications to the way TCP handles inbound segments that can reduce the probability of such an attack.

Table of Contents

1. Introduction 4
1.1. The RESET attack 4
1.2. Attack probabilities 5
2. Terminology 8
3. Blind reset attack using the RST bit 9
3.1. Description of the attack 9
3.2. Mitigation 9
4. Blind reset attack using the SYN bit 11
4.1. Description of the attack 11
4.2. Mitigation 11
5. Blind data injection attack 13
5.1. Description of the attack 13
5.2. Mitigation 13
6. ACK throttling 15
7. Backward Compatibility and Other considerations 16
8. Middlebox considerations 17
8.1. Middlebox that resend RST's 17
8.2. Middleboxes that advance sequence numbers 17
9. Interoperability Testing 19
10. Security Considerations 21
11. IANA Considerations 22
12. Contributors 23
13. Acknowledgments 24
14. References 25
14.1. Normative References 25
14.2. Informative References 25
Authors' Addresses 26
Intellectual Property and Copyright Statements 27

1. Introduction

TCP [RFC0793] is widely deployed and the most common reliable end to end transport protocol used for data communication in todays Internet. 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. TCP spoofing attacks are one such attack that are seen on the Internet today.

In a TCP spoofing attack, an off-path attacker crafts TCP packets by forging the IP source and destination addresses as well as the source and destination ports (commonly referred to as a 4-tuple value) so that a target TCP endpoint can associate such a packet with an existing TCP connection. Note that in and of itself guessing this 4-tuple value is not always easy for an attacker. But there are some applications (e.g. BGP [RFC1771]) that may have a tendency to use the same set(s) of ports on either endpoint making the odds of guessing correctly the 4-tuple value much easier. When an attacker is successful in guessing the 4-tuple value, one of three types of injection attacks may be waged against a long-lived connection.

RST - Where an attacker injects a reset segment hoping to cause the connection to be torn down.

SYN - Where an attacker injects a 'SYN' hoping to cause the receiver to believe the peer has restarted and so tear down the connection state.

DATA - Where an attacker tries to inject a "DATA" segment to corrupt the contents of the transmission.

1.1. The RESET attack

Focusing upon the RESET attack, let's examine this attack in more detail to get an overview as to how it works and how this document proposes addressing the issue. For this attack the goal is to cause one of the two endpoints of the connection to incorrectly tear down the connection state, effectively closing the connection. To do this the attacker needs to have or guess several pieces of information (namely):

1) The 4-tuple value containing the IP address and TCP port number of both ends of the connection. For one side (usually the server) guessing the port number is a trivial exercise. The client side may or may not be easy for an attacker to guess depending on a number of factors most notably the operating system and application involved.

2) A sequence number that will be used in the RST. This sequence number will be a starting point for a series of guesses to attempt to present a RST segment to a connection endpoint that would be acceptable to it. Any random value may be used to guess the initial sequence number.

3) The window size that the two endpoints are using. This value does NOT have to be the exact window size since a smaller value used in lieu of the correct one will just cause the attacker to generate more segments before succeeding in his mischieve. Most modern operating systems have a default window size which usually is applied to most connections. Some applications however may change the window size to better suit the needs of the application. So often times the attacker, with a fair degree of certainty (knowing the application that is under attack), can come up with a very close approximation as to the actual window size in use on the connection.

After assembling the above set of information the attacker begins sending spoofed TCP segments with the RST bit set and a guessed TCP sequence number. Each time a new RST segment is sent, the sequence number guess is incremented by the window size. Without mitigation [SITW] has shown that such an attack is much easier to accomplish then previously assumed. This is because RFC793 [RFC0793] specifies that any RST within the current window is acceptable.

A slight modification to the TCP state machine can be made which makes such an attack much more difficult to accomplish. If the receiver examines the incoming RST segment and validates that the sequence number exactly matches the sequence number that is next expected, then such an attack becomes much more difficult then outlined in [SITW] (i.e. the attacker would have to generate 1/2 the entire sequence space, on average). This document will discuss the exact details of what needs to be changed within TCP's state machine to mitigate all three types of attacks (RST, SYN and DATA).

1.2. Attack probabilities

Every application has control of a number of factors that effect drastically the probability of a successful spoofing attack. These factors include such things as:

Window Size - Normally settable by the application but often times defaulting to 32,768 or 65,535 depending upon the operating system (Medina05 [Medina05]).

Server Port number - This value is normally a fixed value so that a client will know where to connect to the peer at. Thus this value normally provides no additional protection.

Client Port number - This value may be a random ephemeral value, if so, this makes a spoofing attack more difficult. There are some clients, however, that for whatever reason either pick a fixed client port or have a very guessable one (due to the range of ephemeral ports available with their operating system or other application considerations) for such applications a spoofing attack becomes less difficult.

For the purposes of the rest of this discussion we will assume that the attacker knows the 4-tuple values. This assumption will help us focus on the effects of the window size verses the number of TCP packets an attacker must generate. This assumption will rarely be true in the real Internet since at least the client port number will provide us with some amount of randomness (depending on operating system).

To successfully inject a spoofed packet (RST, SYN or DATA), in the past, the entire sequence space (i.e. 2^32) was often considered available to make such an attack unlikely. [SITW] demonstrated that this assumption was incorrect and that instead of [1/2 X 2^32] packets (assuming a random distribution) [1/2 X (2^32/window)] packets is required.

Placing real numbers on this formula we see that for a window size of 32,768, an average of 65,536 packets would need to be transmitted in order to "spoof" a TCP segment that would be acceptable to a TCP receiver. A window size of 65,535 reduces this even further to 32,768 packets. With rises in bandwidth to both the home and office, it can only be expected that the values for default window sizes will continue to rise in order to better take advantage of the newly available bandwidth.

As we can see from the above discussion this weakness lowers the bar quite considerably for likely attacks. But there is one additional dependency which is the duration of the TCP connection. A TCP connection that lasts only a few brief packets, as often is the case for web traffic, would not be subject to such an attack since the connection may not be established long enough for an attacker to generate enough traffic. However there is a set of applications such as BGP [RFC1771] which is judged to be potentially most affected by this vulnerability. BGP relies on a persistent TCP session between BGP peers. Resetting the connection can result in medium term unavailability due to the need to rebuild routing tables and route flapping see [NISCC] for further details.

It should be noted that there are existing alternative protection against the threats that this document addresses. For further details regarding the attacks and the existing techniques, please refer to draft [DTASA]

2. Terminology

The key words "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 [RFC2119]. TCP terminology should be interpreted as described in RFC793 [RFC0793].

3. Blind reset attack using the RST bit

3.1. Description of the attack

As described in the introduction, it is possible for an attacker to generate a "RST" segment that would be acceptable to a TCP receiver by guessing a "in-window" sequence numbers. In particulary RFC 793, p37, states the following:

"In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields [sequence numbers]. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN."

3.2. Mitigation

RFC793 [RFC0793] 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 current receive window (SEG.SEQ <= RCV.NXT || SEG.SEQ > RCV.NXT+ RCV.WND) , 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, this document proposes the following changes should be made to provide protection against such an attack.

A) If the RST bit is set and the sequence number is outside the current receive window, silently drop the segment.

B) If the RST bit is set and the sequence number exactly matches the next expected sequence number (RCV.NXT), then TCP MUST 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 current receive window (RCV.NXT < SEG.SEQ < RCV.NXT+RCV.WND), TCP MUST send an acknowledgment (challenge ACK):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the challenge ACK, TCP MUST drop the unacceptable segment and stop processing the incoming packet further.

The previous text,quoted from RFC793 pg 37 would thus become:

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields [sequence numbers]. A reset is valid if its sequence number exactly matches the next expected sequence number. If the the RST arrives and its sequence number field does NOT match the next expected sequence number but is within the window, then the receiver should generate an ACK. In all other cases where the SEQ-field does not match and is outside the window, the receiver MUST silently discard the segment.

In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN. In all other cases the receiver MUST silently discard the segment.

With the above slight change to the TCP state machine, it becomes much harder for an attacker to generate an acceptable reset segment.

In cases where the remote peer did generate a RST but it fails to meet the above criteria (the RST sequence number was within the window but NOT the exact expected sequence number) when the challenge ACK is sent back, it will no longer have the transmission control block (TCB) related to this connection and hence as per RFC793 [RFC0793], the remote peer will send a second RST back. The sequence number of the second RST is derived from the acknowledgment number of the incoming ACK. This second RST if it reaches the sender will cause the connection to be aborted since the sequence number would now be an exact match.

Note that the above mitigation may cause a non-amplification ACK exchange. This concern is detailed in

4. Blind reset attack using the SYN bit

4.1. Description of the attack

Analysis of the reset attack using the RST bit, highlights another possible avenue for a blind attacker using a similar set of sequence number guessing. Instead of using the RST bit an attacker can use the SYN bit with the exact same symantics to tear down a connection.

4.2. Mitigation

RFC793 [RFC0793] 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 in the synchronized state to the following will mitigate this attack:

A) If the SYN bit is set, irrespective of the sequence number, TCP MUST send an ACK (also referred to as challenge ACK) to the remote peer:

   <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

   After sending the acknowledgment, TCP MUST drop the unacceptable segment and stop processing further.

By sending an ACK, the remote end sender is challenged to confirm the loss of the previous connection and the request to start a new connection. 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 back with the sequence number derived from the ACK field that caused the RST.

This RST will confirm that the remote TCP endpoint has indeed closed the previous connection. Upon receipt of a valid RST, the local TCP endpoint MUST terminate its connection. The local TCP endpoint should then rely on SYN retransmission from the remote end to re-establish the connection.

A spoofed SYN, on the other hand, will then have generated an additional ACK which the peer will discarded as a duplicate ACK and

will not affect the established connection.

Note that this mitigation does leave one corner case un-handled which will prevent the reset of a connection when it should be reset (i.e. it is a non spoofed SYN wherein a peer really did restart). This problem occurs when the restarting host chooses the exact same IP address and port number that it was using prior to its restart. By chance the restarted host must also choose an initial sequence number of exactly (RCV.NXT - 1) of the remote TCP endpoint that is still in the established state. Such a case would cause the receiver to generate a "challenge" ack as described above. But since the ACK would be within the outgoing connections window the inbound ACK would be acceptable, and the sender of the SYN will do nothing with the response ACK. This sequence will continue as the SYN sender continually times out and retransmits the SYN until such time as the connection attempt fails.

This corner case is a result of the RFC793 [RFC0793] specification and is not introduced by the proposed mitigations.

Note that the above mitigation may cause a non-amplification ACK exchange. This concern is detailed in Section 10

5. Blind data injection attack

5.1. Description of the attack

A third type of attack is also highlighted by both the RST and SYN attacks. It is also possible to inject data into a TCP connection by simply guessing the a sequence number within the current receive window of the victim. 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 has to guess two ACK values with every guessed sequence number so that the chances 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 packet 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 an ACK with an acknowledgment number less than RCV.NXT.

b) The sender will send enough data to the peer which will move RCV.NXT even further along past the injected data.

Depending upon the TCP implementation in question and the TCP traffic characteristics at that time, data corruption may result. In case (a) the connection will eventually be reset by one of the sides unless the sender produces more data that will transform the ACK war into case (b). The reset will usually occur via User Time Out (UTO) (see section 4.2.3.5 of [RFC1122]).

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 product of the attack itself and cannot be prevented (other than by preventing the data from being injected).

5.2. Mitigation

An additional input check should be added to any incoming segment. The ACK value MUST 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 its peer. This window may be a scaled value i.e. the

value may be larger than 65,535 bytes (RFC1323 [RFC1323]). This small check will greatly reduce the vulnerability to 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 mitigation reduces but does not eliminate the ability to generate false segments. It does however reduce the probability that invalid data will be injected.

Note that the above mitigation may cause a non-amplification ACK exchange. This concern is detailed in

6. ACK throttling

In order to alleviate multiple RSTs/SYNs from triggering multiple challenge ACKs, an ACK throttling mechanism SHOULD be implemented. Suggested values are to send no more than 10 challenge ACKs in a 5 second window. These numbers are empirical in nature and have been obtained from the RST throttling mechanism implemented in some OS's. These value MUST be tunable by a system administrator to accommodate different preceived threats.

An alternative mechanism may also be used that does not involve an additional timer. In such an implementation a sender would only send X acks between any window advancment. Note that such a limitation will not require a timer but must be implemented with care to avoid a deadlock in the face of ack loss.

An implementation SHOULD include a ACK throttling mechanism to be conservative. Currently there is no known bad behavior that can be attributed to the lack of ACK throttling, but as a general principle, if ever invoked, something incorrect is occuring and such a mechanisn will act as a failsafe that protects both the sender and the network.

An administrator who is more concerned about protecting his bandwidth and CPU utilization may set smaller ACK thottling values whereas an administrator who is more interested in faster cleanup of stale connections (i.e. concerned about excess TCP state) may decide to set a higher value thus allowing more RST's to be processed in any given time period.

The time limit SHOULD be tunable to help timeout brute force attacks faster than a potential legitimate flood of RSTs.

7. Backward Compatibility and Other considerations

1) All of the proposed mitigation techniques in this document are totally compatible with existing (RFC793 [RFC0793]) compliant TCP implementations as this document introduces no new assumptions or conditions.

2) There is a corner scenario in the above proposed mitigations 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. ESTAB        <-- <SEQ=300><ACK=101><CTL=ACK><DATA> <--  ESTAB
  b. (delayed)    ... <SEQ=400><ACK=101><CTL=ACK><DATA> <--  ESTAB
  c. (in flight)  ... <SEQ=500><ACK=101><CTL=RST>       <--  CLOSED
2.   ESTAB        --> <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. ESTAB        <-- <SEQ=400><ACK=101><CTL=ACK><DATA> 1.b reaches A
  b. ESTAB        --> <SEQ=101><ACK=500><CTL=ACK>
  c. (in flight)  ... <SEQ=500><ACK=0><CTL=RST>         <--  CLOSED
5.   RESPONSE arrives at A, but dropped since its outside of window.
6.   ESTAB        <-- <SEQ=500><ACK=0><CTL=RST>         4.c reaches A
7.   CLOSED                                                  CLOSED
```

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

8. Middlebox considerations

8.1. Middlebox that resend RST's

Consider a middlebox M-B tracking connection between two TCP endhosts E-A and E-C. If E-C sends a RST with a sequence number that is within the window but not an exact match to reset the connection and M-B does not have the fix proposed here, it may clear the connection and forward the RST to E-A saving an incorrect sequence number. If E-A does not have the fix it will clear the connection and everything will be fine. However if E-A does have the proposed fix above, it will send a challenge ACK to E-C. M-B, being a middlebox, may intercept this ACK and resend the RST on behalf of E-C with the old sequence number. This RST, will again, not acceptable and may trigger a challenge ACK.

This may cause a RST/ACK war to occur. However we believe that if such a case exists in the Internet the middle box design itself is flawed. Consider a similar scenario where the RST from M-B to E-A gets lost, E-A will continue to hold the connection and E-A might send an ACK an arbitrary time later after the connection state was destroyed at M-B. For this case, M-B will have to cache the RST for an arbitrary amount of time till until it is confirmed that the connection has been cleared at E-A. Further, this is not compliant to RFC793 which dictates that the sequence number of a RST has to be derived from the acknowledgment number of the incoming ACK segment.

8.2. Middleboxes that advance sequence numbers

Some middleboxes may compute RST sequence numbers at the higher end of the acceptable window. The scenario is the same as the earlier case, but in this case instead of sending the cached RST, the middlebox (M-B) sends a RST that computes its sequence number as a sum of the ack field in the ACK and the window advertised by the ACK that was sent by E-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.

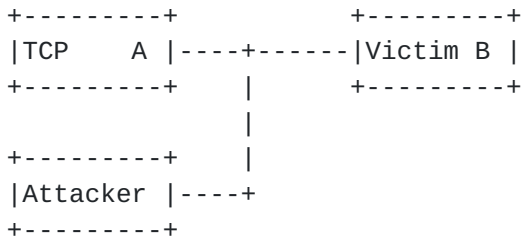
```
   TCP A                                                Middlebox

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 an implementation that does the above, it could be mitigated by implementing the ACK throttling mechanism described earlier.

9. Interoperability Testing

Interoperability testing was performed among the operating systems from Juniper Networks, WindRiver Systems, QNX Software and Cisco Systems. The following topology was used:



In the above topology B is the unit under test. TCP A is a remote peer and the attacker workstation is used to generate malicious spoofed packets.

First, an unmodifed stack had the following tests performed upon it. A TCP connection was brought up between TCP endpoint A and B. The 4-tuple of the connection was manually populated in the brute force attack script running on the attacker workstation. The script crafted TCP packets by using the 4-tuple and by generating sequence numbers that incremented from 0 to the max permissible sequence number (2^32 -1) in varying increments of window size of 8K to 64K (the window size agreed by A and B was larger than 8K, but was ignored to make the test conservative). The time it took to cause the connection to reset/corrupt was then recorded.

The test was repeated by then generating sequence numbers that were window (the one agreed between endpoints A and B) size apart. It was observed that increasing the window size caused the connection to be reset faster than with a smaller window. Further, it was also observed that the initial sequence number (ISN) selection also played a role in how fast a connection was aborted, with ISN selection in the lower half of the sequence space aborting sooner than an ISN in the upper half. Results were also found to be influenced by any active data transfer on the connection.

Active data transfer sometimes caused the connection to be reset faster and some other times to slow the attack. The window size selection at the attacker workstation and how it compares to the actual window size between A and B was also found to be a factor. The tests were repeated with a stack that did have the fix and as expected it became difficult as compared to the previous results to cause the connection to abort.

[Editors Note: Add time test data gathered at inter-op here!!]

10. Security Considerations

A reflector attack is possible with the proposed RST/SYN mitigation techniques. Here an off-path attacker can cause a victim to send an ACK segment for each spoofed RST/SYN segment that lies within the current receive window of the victim. This, however, does not cause any sort of amplification since the attacker must generate a segment for each one that the victim will generate.

11. IANA Considerations

This document contains no IANA considerations.

12. 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.

13. Acknowledgments

Special thanks to Mark Allman, Ted Faber, Steve Bellovin, Vern Paxson, Allison Mankin, Sharad Ahlawat, Damir Rajnovic, John Wong and the tcpm WG members for suggestions and comments. Some of the text in this document has been derived from

14. References

14.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

14.2. Informative References

[DTASA] Touch, J., "Defending TCP Against Spoofing Attacks", draft-touch-tcp-antispoof-00 (work in progress), July 2004.

[Medina05] Medina, A., Allman, M., and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet. ACM Computer Communication Review, 35(2), April 2005. http://www.icir.org/mallman/papers/tcp-evo-ccr05.ps (figure 6)".

[NISCC] NISCC, "NISCC Vulnerability Advisory 236929 - Vulnerability Issues in TCP".

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, 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.

[RFC3562] Leech, M., "Key Management Considerations for the TCP MD5 Signature Option", RFC 3562, July 2003.

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

Authors' Addresses

Randall R. Stewart
Editor
4875 Forest Drive
Suite 200
Columbia, SC 29206
USA

Phone:
Email: rrs@cisco.com

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

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

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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