Network Working Group   W. Eddy
Request for Comments: 4987   Verizon
Category: Informational   August 2007

               TCP SYN Flooding Attacks and Common Mitigations

Status of This Memo

Copyright Notice

Abstract

   This document describes TCP SYN flooding attacks, which have been
   well-known to the community for several years. Various
   countermeasures against these attacks, and the trade-offs of each,
   are described. This document archives explanations of the attack and
   common defense techniques for the benefit of TCP implementers and
   administrators of TCP servers or networks, but does not make any
   standards-level recommendations.

Table of Contents

1. Introduction

   The SYN flooding attack is a denial-of-service method affecting hosts
   that run TCP server processes. The attack takes advantage of the
   state retention TCP performs for some time after receiving a SYN
   segment to a port that has been put into the LISTEN state. The basic
   idea is to exploit this behavior by causing a host to retain enough
   state for bogus half-connections that there are no resources left to
   establish new legitimate connections.

   This SYN flooding attack has been well-known to the community for
   many years, and has been observed in the wild by network operators
   and end hosts. A number of methods have been developed and deployed
   to make SYN flooding less effective. Despite the notoriety of the
   attack, and the widely available countermeasures, the RFC series only
   documented the vulnerability as an example motivation for ingress
   filtering [RFC2827], and has not suggested any mitigation techniques
   for TCP implementations. This document addresses both points, but
   does not define any standards. Formal specifications and
   requirements of defense mechanisms are outside the scope of this
   document. Many defenses only impact an end host's implementation
   without changing interoperability. These may not require
   standardization, but their side-effects should at least be well
   understood.

   This document intentionally focuses on SYN flooding attacks from an
   individual end host or application's perspective, as a means to deny
   service to that specific entity. High packet-rate attacks that
   target the network's packet-processing capability and capacity have
   been observed operationally. Since such attacks target the network,
   and not a TCP implementation, they are out of scope for this
   document, whether or not they happen to use TCP SYN segments as part
   of the attack, as the nature of the packets used is irrelevant in
   comparison to the packet-rate in such attacks.

   The majority of this document consists of three sections. Section 2
   explains the SYN flooding attack in greater detail. Several common
   mitigation techniques are described in Section 3. An analysis and
   discussion of these techniques and their use is presented in
   Section 4. Further information on SYN cookies is contained in
   Appendix A.

2. Attack Description

   This section describes both the history and the technical basis of
   the SYN flooding attack.

2.1. History

   The TCP SYN flooding weakness was discovered as early as 1994 by Bill
   Cheswick and Steve Bellovin [B96]. They included, and then removed,
   a paragraph on the attack in their book "Firewalls and Internet
   Security: Repelling the Wily Hacker" [CB94]. Unfortunately, no
   countermeasures were developed within the next two years.

   The SYN flooding attack was first publicized in 1996, with the
   release of a description and exploit tool in Phrack Magazine
   [P48-13]. Aside from some minor inaccuracies, this article is of
   high enough quality to be useful, and code from the article was
   widely distributed and used.

   By September of 1996, SYN flooding attacks had been observed in the
   wild. Particularly, an attack against one ISP's mail servers caused
   well-publicized outages. CERT quickly released an advisory on the
   attack [CA-96.21]. SYN flooding was particularly serious in
   comparison to other known denial-of-service attacks at the time.
   Rather than relying on the common brute-force tactic of simply
   exhausting the network's resources, SYN flooding targets end-host
   resources, which require fewer packets to deplete.

   The community quickly developed many widely differing techniques for
   preventing or limiting the impact of SYN flooding attacks. Many of
   these have been deployed to varying degrees on the Internet, in both
   end hosts and intervening routers. Some of these techniques have
   become important pieces of the TCP implementations in certain
   operating systems, although some significantly diverge from the TCP
   specification and none of these techniques have yet been standardized
   or sanctioned by the IETF process.

2.2. Theory of Operation

   As described in RFC 793, a TCP implementation may allow the LISTEN
   state to be entered with either all, some, or none of the pair of IP
   addresses and port numbers specified by the application. In many
   common applications like web servers, none of the remote host's
   information is pre-known or preconfigured, so that a connection can
   be established with any client whose details are unknown to the
   server ahead of time. This type of "unbound" LISTEN is the target of
   SYN flooding attacks due to the way it is typically implemented by
   operating systems.

   For success, the SYN flooding attack relies on the victim host TCP
   implementation's behavior. In particular, it assumes that the victim
   allocates state for every TCP SYN segment when it is received, and
   that there is a limit on the amount of such state than can be kept at

any time. The current base TCP specification, RFC 793 [RFC0793],
describes the standard processing of incoming SYN segments. RFC 793
describes the concept of a Transmission Control Block (TCB) data
structure to store all the state information for an individual
connection. In practice, operating systems may implement this
concept rather differently, but the key is that each TCP connection
requires some memory space.

Per RFC 793, when a SYN is received for a local TCP port where a
connection is in the LISTEN state, then the state transitions to SYN-
RECEIVED, and some of the TCB is initialized with information from
the header fields of the received SYN segment. In practice, many
operating systems do not alter the TCB in LISTEN, but instead make a
copy of the TCB and perform the state transition and update on the
copy. This is done so that the local TCP port may be shared amongst
several distinct connections. This TCB-copying behavior is not
actually essential for this purpose, but influences the way in which
applications that wish to handle multiple simultaneous connections
through a single TCP port are written. The crucial result of this
behavior is that, instead of updating already-allocated memory, new
(or unused) memory must be devoted to the copied TCB.

As an example, in the Linux 2.6.10 networking code, a "sock"
structure is used to implement the TCB concept. By examination, this
structure takes over 1300 bytes to store in memory. In other systems
that implement less-complex TCP algorithms and options, the overhead
may be less, although it typically exceeds 280 bytes [SKK+97].

To protect host memory from being exhausted by connection requests,
the number of TCB structures that can be resident at any time is
usually limited by operating system kernels. Systems vary on whether
limits are globally applied or local to a particular port number.
There is also variation on whether the limits apply to fully
established connections as well as those in SYN-RECEIVED. Commonly,
systems implement a parameter to the typical listen() system call
that allows the application to suggest a value for this limit, called
the backlog. When the backlog limit is reached, then either incoming
SYN segments are ignored, or uncompleted connections in the backlog
are replaced. The concept of using a backlog is not described in the
standards documents, so the failure behavior when the backlog is
reached might differ between stacks (for instance, TCP RSTs might be
generated). The exact failure behavior will determine whether
initiating hosts continue to retransmit SYN segments over time, or
quickly cease. These differences in implementation are acceptable
since they only affect the behavior of the local stack when its
resources are constrained, and do not cause interoperability
problems.

The SYN flooding attack does not attempt to overload the network's
resources or the end host's memory, but merely attempts to exhaust
the backlog of half-open connections associated with a port number.
The goal is to send a quick barrage of SYN segments from IP addresses
(often spoofed) that will not generate replies to the SYN-ACKs that
are produced. By keeping the backlog full of bogus half-opened
connections, legitimate requests will be rejected. Three important
attack parameters for success are the size of the barrage, the
frequency with which barrages are generated, and the means of
selecting IP addresses to spoof.

Barrage Size

   To be effective, the size of the barrage must be made large enough
   to reach the backlog. Ideally, the barrage size is no larger than
   the backlog, minimizing the volume of traffic the attacker must
   source. Typical default backlog values vary from a half-dozen to
   several dozen, so the attack might be tailored to the particular
   value determined by the victim host and application. On machines
   intended to be servers, especially for a high volume of traffic,
   the backlogs are often administratively configured to higher
   values.

Barrage Frequency

   To limit the lifetime of half-opened connection state, TCP
   implementations commonly reclaim memory from half-opened
   connections if they do not become fully opened after some time
   period. For instance, a timer of 75 seconds [SKK+97] might be set
   when the first SYN-ACK is sent, and on expiration cause SYN-ACK
   retransmissions to cease and the TCB to be released. The TCP
   specifications do not include this behavior of giving up on
   connection establishment after an arbitrary time. Some purists
   have expressed that the TCP implementation should continue
   retransmitting SYN and SYN-ACK segments without artificial bounds
   (but with exponential backoff to some conservative rate) until the
   application gives up. Despite this, common operating systems
   today do implement some artificial limit on half-open TCB
   lifetime. For instance, backing off and stopping after a total of
   511 seconds can be observed in 4.4 BSD-Lite [Ste95], and is still
   practiced in some operating systems derived from this code.

   To remain effective, a SYN flooding attack needs to send new
   barrages of bogus connection requests as soon as the TCBs from the
   previous barrage begin to be reclaimed. The frequency of barrages
   are tailored to the victim TCP implementation's TCB reclamation
   timer. Frequencies higher than needed source more packets,
   potentially drawing more attention, and frequencies that are too

low will allow windows of time where legitimate connections can be
established.

   IP Address Selection

      For an effective attack, it is important that the spoofed IP
      addresses be unresponsive to the SYN-ACK segments that the victim
      will generate. If addresses of normal connected hosts are used,
      then those hosts will send the victim a TCP reset segment that
      will immediately free the corresponding TCB and allow room in the
      backlog for legitimate connections to be made. The code
      distributed in the original Phrack article used a single source
      address for all spoofed SYN segments. This makes the attack
      segments somewhat easier to identify and filter. A strong
      attacker will have a list of unresponsive and unrelated addresses
      that it chooses spoofed source addresses from.

   It is important to note that this attack is directed at particular
   listening applications on a host, and not the host itself or the
   network. The attack also attempts to prevent only the establishment
   of new incoming connections to the victim port, and does not impact
   outgoing connection requests, nor previously established connections
   to the victim port.

   In practice, an attacker might choose not to use spoofed IP
   addresses, but instead to use a multitude of hosts to initiate a SYN
   flooding attack. For instance, a collection of compromised hosts
   under the attacker's control (i.e., a "botnet") could be used. In
   this case, each host utilized in the attack would have to suppress
   its operating system's native response to the SYN-ACKs coming from
   the target. It is also possible for the attack TCP segments to
   arrive in a more continuous fashion than the "barrage" terminology
   used here suggests; as long as the rate of new SYNs exceeds the rate
   at which TCBs are reaped, the attack will be successful.

3. Common Defenses

   This section discusses a number of defense techniques that are known
   to the community, many of which are available in off-the-shelf
   products.

3.1. Filtering

   Since in the absence of an army of controlled hosts, the ability to
   send packets with spoofed source IP addresses is required for this
   attack to work, removing an attacker's ability to send spoofed IP
   packets is an effective solution that requires no modifications to
   TCP. The filtering techniques described in RFCs 2827, 3013, and 3704

represent the best current practices for packet filtering based on IP
addresses [RFC2827][RFC3013][RFC3704]. While perfectly effective,
end hosts should not rely on filtering policies to prevent attacks
from spoofed segments, as global deployment of filters is neither
guaranteed nor likely. An attacker with the ability to use a group
of compromised hosts or to rapidly change between different access
providers will also make filtering an impotent solution.

3.2. Increasing Backlog

An obvious attempt at a defense is for end hosts to use a larger
backlog. Lemon has shown that in FreeBSD 4.4, this tactic has some
serious negative aspects as the size of the backlog grows [Lem02].
The implementation has not been designed to scale past backlogs of a
few hundred, and the data structures and search algorithms that it
uses are inefficient with larger backlogs. It is reasonable to
assume that other TCP implementations have similar design factors
that limit their performance with large backlogs, and there seems to
be no compelling reason why stacks should be re-engineered to support
extremely large backlogs, since other solutions are available.
However, experiments with large backlogs using efficient data
structures and search algorithms have not been conducted, to our
knowledge.

3.3. Reducing SYN-RECEIVED Timer

Another quickly implementable defense is shortening the timeout
period between receiving a SYN and reaping the created TCB for lack
of progress. Decreasing the timer that limits the lifetime of TCBs
in SYN-RECEIVED is also flawed. While a shorter timer will keep
bogus connection attempts from persisting for as long in the backlog,
and thus free up space for legitimate connections sooner, it can
prevent some fraction of legitimate connections from becoming fully
established. This tactic is also ineffective because it only
requires the attacker to increase the barrage frequency by a linearly
proportional amount. This timer reduction is sometimes implemented
as a response to crossing some threshold in the backlog occupancy, or
some rate of SYN reception.

3.4. Recycling the Oldest Half-Open TCB

Once the entire backlog is exhausted, some implementations allow
incoming SYNs to overwrite the oldest half-open TCB entry. This
works under the assumption that legitimate connections can be fully
established in less time than the backlog can be filled by incoming
attack SYNs. This can fail when the attacking packet rate is high
and/or the backlog size is small, and is not a robust defense.

3.5. SYN Cache

   The SYN cache, best described by Lemon [Lem02], is based on
   minimizing the amount of state that a SYN allocates, i.e., not
   immediately allocating a full TCB. The full state allocation is
   delayed until the connection has been fully established. Hosts
   implementing a SYN cache have some secret bits that they select from
   the incoming SYN segments. The secret bits are hashed along with the
   IP addresses and TCP ports of a segment, and the hash value
   determines the location in a global hash table where the incomplete
   TCB is stored. There is a bucket limit for each hash value, and when
   this limit is reached, the oldest entry is dropped.

   The SYN cache technique is effective because the secret bits prevent
   an attacker from being able to target specific hash values for
   overflowing the bucket limit, and it bounds both the CPU time and
   memory requirements. Lemon's evaluation of the SYN cache shows that
   even under conditions where a SYN flooding attack is not being
   performed, due to the modified processing path, connection
   establishment is slightly more expedient. Under active attack, SYN
   cache performance was observed to approximately linearly shift the
   distribution of times to establish legitimate connections to about
   15% longer than when not under attack [Lem02].

   If data accompanies the SYN segment, then this data is not
   acknowledged or stored by the receiver, and will require
   retransmission. This does not affect the reliability of TCP's data
   transfer service, but it does affect its performance to some small
   extent. SYNs carrying data are used by the T/TCP extensions
   [RFC1644]. While T/TCP is implemented in a number of popular
   operating systems [GN00], it currently seems to be rarely used.
   Measurements at one site's border router [All07] logged 2,545,785 SYN
   segments (not SYN-ACKs), of which 36 carried the T/TCP CCNEW option
   (or 0.001%). These came from 26 unique hosts, and no other T/TCP
   options were seen. 2,287 SYN segments with data were seen (or 0.09%
   of all SYN segments), all of which had exactly 24 bytes of data.
   These observations indicate that issues with SYN caches and data on
   SYN segments may not be significant in deployment.

3.6. SYN Cookies

   SYN cookies go a step further and allocate no state at all for
   connections in SYN-RECEIVED. Instead, they encode most of the state
   (and all of the strictly required) state that they would normally
   keep into the sequence number transmitted on the SYN-ACK. If the SYN
   was not spoofed, then the acknowledgement number (along with several
   other fields) in the ACK that completes the handshake can be used to
   reconstruct the state to be put into the TCB. To date, one of the

best references on SYN cookies can be found on Dan Bernstein's web
site [cr.yp.to]. This technique exploits the long-understood low
entropy in TCP header fields [RFC1144][RFC4413]. In Appendix A, we
describe the SYN cookie technique, to avoid the possibility that the
web page will become unavailable.

The exact mechanism for encoding state into the SYN-ACK sequence
number can be implementation dependent. A common consideration is
that to prevent replay, some time-dependent random bits must be
embedded in the sequence number. One technique used 7 bits for these
bits and 25 bits for the other data [Lem02]. One way to encode these
bits has been to XOR the initial sequence number received with a
truncated cryptographic hash of the IP address and TCP port number
pairs, and secret bits. In practice, this hash has been generated
using MD5 [RFC1321]. Any similar one-way hash could be used instead
without impacting interoperability since the hash value is checked by
the same host who generates it.

The problem with SYN cookies is that commonly implemented schemes are
incompatible with some TCP options, if the cookie generation scheme
does not consider them. For example, an encoding of the Maximum
Segment Size (MSS) advertised on the SYN has been accommodated by
using 2 sequence number bits to represent 4 predefined common MSS
values. Similar techniques would be required for some other TCP
options, while negotiated use of other TCP options can be detected
implicitly. A timestamp on the ACK, as an example, indicates that
Timestamp use was successfully negotiated on the SYN and SYN-ACK,
while the reception of a Selective Acknowledgement (SACK) option at
some point during the connection implies that SACK was negotiated.
Note that SACK blocks should normally not be sent by a host using TCP
cookies unless they are first received. For the common
unidirectional data flow in many TCP connections, this can be a
problem, as it limits SACK usage. For this reason, SYN cookies
typically are not used by default on systems that implement them, and
are only enabled either under high-stress conditions indicative of an
attack, or via administrative action.

Recently, a new SYN cookie technique developed for release in FreeBSD
7.0 leverages the bits of the Timestamp option in addition to the
sequence number bits for encoding state. Since the Timestamp value
is echoed back in the Timestamp Echo field of the ACK packet, any
state stored in the Timestamp option can be restored similarly to the
way that it is from the sequence number / acknowledgement in a basic
SYN cookie. Using the Timestamp bits, it is possible to explicitly
store state bits for things like send and receive window scales,
SACK-allowed, and TCP-MD5-enabled, for which there is no room in a
typical SYN cookie. This use of Timestamps to improve the
compromises inherent in SYN cookies is unique to the FreeBSD

implementation, to our knowledge. A limitation is that the technique
can only be used if the SYN itself contains a Timestamp option, but
this option seems to be widely implemented today, and hosts that
support window scaling and SACK typically support timestamps as well.

Similarly to SYN caches, SYN cookies do not handle application data
piggybacked on the SYN segment.

Another problem with SYN cookies is for applications where the first
application data is sent by the passive host. If this host is
handling a large number of connections, then packet loss may be
likely. When a handshake-completing ACK from the initiator is lost,
the passive side's application layer never is notified of the
connection's existence and never sends data, even though the
initiator thinks that the connection has been successfully
established. An example application where the first application-
layer data is sent by the passive side is SMTP, if implemented
according to RFC 2821, where a "service ready" message is sent by the
passive side after the TCP handshake is completed.

Although SYN cookie implementations exist and are deployed, the use
of SYN cookies is often disabled in default configurations, so it is
unclear how much operational experience actually exists with them or
if using them opens up new vulnerabilities. Anecdotes of incidents
where SYN cookies have been used on typical web servers seem to
indicate that the added processing burden of computing MD5 sums for
every SYN packet received is not significant in comparison to the
loss of application availability when undefended. For some
computationally constrained mobile or embedded devices, this
situation might be different.

3.7. Hybrid Approaches

   The SYN cache and SYN cookie techniques can be combined. For
   example, in the event that the cache becomes full, then SYN cookies
   can be sent instead of purging cache entries upon the arrival of new
   SYNs. Such hybrid approaches may provide a strong combination of the
   positive aspects of each approach. Lemon has demonstrated the
   utility of this hybrid [Lem02].

3.8. Firewalls and Proxies

   Firewall-based tactics may also be used to defend end hosts from SYN
   flooding attacks. The basic concept is to offload the connection
   establishment procedures onto a firewall that screens connection
   attempts until they are completed and then proxies them back to
   protected end hosts. This moves the problem away from end hosts to
   become the firewall's or proxy's problem, and may introduce other

problems related to altering TCP's expected end-to-end semantics. A
common tactic used in these firewall and proxy products is to
implement one of the end host based techniques discussed above, and
screen incoming SYNs from the protected network until the connection
is fully established. This is accomplished by spoofing the source
addresses of several packets to the initiator and listener at various
stages of the handshake [Eddy06].

4. Analysis

   Several of the defenses discussed in the previous section rely on
   changes to behavior inside the network; via router filtering,
   firewalls, and proxies. These may be highly effective, and often
   require no modification or configuration of end-host software. Given
   the mobile nature and dynamic connectivity of many end hosts, it is
   optimistic for TCP implementers to assume the presence of such
   protective devices. TCP implementers should provide some means of
   defense to SYN flooding attacks in end-host implementations.

   Among end-host modifications, the SYN cache and SYN cookie approaches
   seem to be the only viable techniques discovered to date. Increasing
   the backlog and reducing the SYN-RECEIVED timer are measurably
   problematic. The SYN cache implies a higher memory footprint than
   SYN cookies; however, SYN cookies may not be fully compatible with
   some TCP options, and may hamper development of future TCP extensions
   that require state. For these reasons, SYN cookies should not be
   enabled by default on systems that provide them. SYN caches do not
   have the same negative implications and may be enabled as a default
   mode of processing.

   In October of 1996, Dave Borman implemented a SYN cache at BSDi for
   BSD/OS, which was given to the community with no restrictions. This
   code seems to be the basis for the SYN cache implementations adopted
   later in other BSD variants. The cache was used when the backlog
   became full, rather than by default, as we have described. A note to
   the tcp-impl mailing list explains that this code does not retransmit
   SYN-ACKs [B97]. More recent implementations have chosen to reverse
   this decision and retransmit SYN-ACKs. It is known that loss of SYN-
   ACK packets is not uncommon [SD01] and can severely slow the
   performance of connections when initial retransmission timers for
   SYNs are overly conservative (as in some operating systems) or
   retransmitted SYNs are lost. Furthermore, if a SYN flooding attacker
   has a high sending rate, loss of retransmitted SYNs is likely, so if
   SYN-ACKs are not retransmitted, the chance of efficiently
   establishing legitimate connections is reduced.

In 1997, NetBSD incorporated a modified version of Borman's code.
Two notable differences from the original code stem from the decision
to use the cache by default (for all connections). This implied the
need to perform retransmissions for SYN-ACKs, and to use larger
structures to keep more complete data. The original structure was 32
bytes long for IPv4 connections and 56 bytes with IPv6 support, while
the current FreeBSD structure is 196 bytes long. As previously
cited, Lemon implemented the SYN cache and cookie techniques in
FreeBSD 4.4 [Lem02]. Lemon notes that a SYN cache structure took up
160 bytes compared to 736 for the full TCB (now 196 bytes for the
cache structure). We have examined the OpenBSD 3.6 code and
determined that it includes a similar SYN cache.

Linux 2.6.5 code, also by examination, contains a SYN cookie
implementation that encodes 8 MSS values, and does not use SYN
cookies by default. This functionality has been present in the Linux
kernel for several years previous to 2.6.5.

When a SYN cache and/or SYN cookies are implemented with IPv6, the
IPv6 flow label value used on the SYN-ACK should be consistent with
the flow label used for the rest of the packets within that flow.
There have been implementation bugs that caused random flow labels to
be used in SYN-ACKs generated by SYN cache and SYN cookie code
[MM05].

Beginning with Windows 2000, Microsoft's Windows operating systems
have had a "TCP SYN attack protection" feature, which can be toggled
on or off in the registry. This defaulted to off, until Windows 2003
SP1, in which it is on by default. With this feature enabled, when
the number of half-open connections and half-open connections with
retransmitted SYN-ACKs exceeds configurable thresholds, then the
number of times that SYN-ACKs are retransmitted before giving up is
reduced, and the "Route Cache Entry" creation is delayed, which
prevents some features (e.g., window scaling) from being used
[win2k3-wp].

Several vendors of commercial firewall products sell devices that can
mitigate SYN flooding's effects on end hosts by proxying connections.

Discovery and exploitation of the SYN flooding vulnerability in TCP's
design provided a valuable lesson for protocol designers. The Stream
Control Transmission Protocol [RFC2960], which was designed more
recently, incorporated a 4-way handshake with a stateless cookie-
based component for the listening end. In this way, the passive-
opening side has better evidence that the initiator really exists at
the given address before it allocates any state. The Host Identity
Protocol base exchange [MNJH07] is similarly designed as a 4-way
handshake, but also involves a puzzle sent to the initiator that must

be solved before any state is reserved by the responder. The general
concept of designing statelessness into protocol setup to avoid
denial-of-service attacks has been discussed by Aura and Nikander
[AN97].

5. Security Considerations

The SYN flooding attack on TCP has been described in numerous other
publications, and the details and code needed to perform the attack
have been easily available for years. Describing the attack in this
document does not pose any danger of further publicizing this
weakness in unmodified TCP stacks. Several widely deployed operating
systems implement the mitigation techniques that this document
discusses for defeating SYN flooding attacks. In at least some
cases, these operating systems do not enable these countermeasures by
default; however, the mechanisms for defeating SYN flooding are well
deployed, and easily enabled by end-users. The publication of this
document should not influence the number of SYN flooding attacks
observed, and might increase the robustness of the Internet to such
attacks by encouraging use of the commonly available mitigations.

6. Acknowledgements

A conversation with Ted Faber was the impetus for writing this
document. Comments and suggestions from Joe Touch, Dave Borman,
Fernando Gont, Jean-Baptiste Marchand, Christian Huitema, Caitlin
Bestler, Pekka Savola, Andre Oppermann, Alfred Hoenes, Mark Allman,
Lars Eggert, Pasi Eronen, Warren Kumari, David Malone, Ron Bonica,
and Lisa Dusseault were useful in strengthening this document. The
original work on TCP SYN cookies presented in Appendix A is due to
D.J. Bernstein.

Work on this document was performed at NASA's Glenn Research Center.
Funding was partially provided by a combination of NASA's Advanced
Communications, Navigation, and Surveillance Architectures and System
Technologies (ACAST) project, the Sensis Corporation, NASA's Space
Communications Architecture Working Group, and NASA's Earth Science
Technology Office.

7. Informative References

   [AN97]   Aura, T. and P. Nikander, "Stateless Connections",
                  Proceedings of the First International Conference on
                  Information and Communication Security, 1997.

   [All07]   Allman, M., "personal communication", February 2007.

   [B96]   Bennahum, D., "PANIX ATTACK", MEME 2.12, October 1996,
                <http://memex.org/meme2-12.html>.

   [B97]   Borman, D., "Re: SYN/RST cookies (was Re: a quick
                clarification...)", IETF tcp-impl mailing list,
                June 1997.

   [CA-96.21]   CERT, "CERT Advisory CA-1996-21 TCP SYN Flooding and IP
                Spoofing Attacks", September 1996.

   [CB94]   Cheswick, W. and S. Bellovin, "Firewalls and Internet
                Security", ISBN: 0201633574, January 1994.

   [Eddy06]   Eddy, W., "Defenses Against TCP SYN Flooding Attacks",
                Cisco Internet Protocol Journal Volume 8, Number 4,
                December 2006.

   [GN00]   Griffin, M. and J. Nelson, "T/TCP: TCP for
                Transactions", Linux Journal, February 2000.

   [Lem02]   Lemon, J., "Resisting SYN Flood DoS Attacks with a SYN
                Cache", BSDCON 2002, February 2002.

   [MM05]   McGann, O. and D. Malone, "Flow Label Filtering
                Feasibility", European Conference on Computer Network
                Defense 2005, December 2005.

   [MNJH07]   Moskowitz, R., Nikander, P., Jokela, P., and T.
                Henderson, "Host Identity Protocol", Work in Progress,
                June 2007.

   [P48-13]   daemon9, route, and infinity, "Project Neptune", Phrack
                Magazine, Volume 7, Issue 48, File 13 of 18, July 1996.

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

   [RFC1144]   Jacobson, V., "Compressing TCP/IP headers for low-speed
                serial links", RFC 1144, February 1990.

   [RFC1321]   Rivest, R., "The MD5 Message-Digest Algorithm",
                RFC 1321, April 1992.

   [RFC1644]   Braden, B., "T/TCP -- TCP Extensions for Transactions
                Functional Specification", RFC 1644, July 1994.

   [RFC2827]   Ferguson, P. and D. Senie, "Network Ingress Filtering:
                Defeating Denial of Service Attacks which employ IP
                Source Address Spoofing", BCP 38, RFC 2827, May 2000.

   [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.

   [RFC3013]   Killalea, T., "Recommended Internet Service Provider
                Security Services and Procedures", BCP 46, RFC 3013,
                November 2000.

   [RFC3704]   Baker, F. and P. Savola, "Ingress Filtering for
                Multihomed Networks", BCP 84, RFC 3704, March 2004.

   [RFC4413]   West, M. and S. McCann, "TCP/IP Field Behavior",
                RFC 4413, March 2006.

   [SD01]   Seddigh, N. and M. Devetsikiotis, "Studies of TCP's
                Retransmission Timeout Mechanism", Proceedings of the
                2001 IEEE International Conference on Communications
                (ICC 2001), volume 6, pages 1834-1840, June 2001.

   [SKK+97]   Schuba, C., Krsul, I., Kuhn, M., Spafford, E., Sundaram,
                A., and D. Zamboni, "Analysis of a Denial of Service
                Attack on TCP", Proceedings of the 1997 IEEE Symposium
                on Security and Privacy 1997.

   [Ste95]   Stevens, W. and G. Wright, "TCP/IP Illustrated, Volume
                2: The Implementation", January 1995.

   [cr.yp.to]   Bernstein, D., "SYN cookies", visited in December 2005,
                <http://cr.yp.to/syncookies.html>.

   [win2k3-wp] Microsoft Corporation, "Microsoft Windows Server 2003
                TCP/IP Implementation Details", White Paper, July 2005.

Appendix A. SYN Cookies Description

   This information is taken from Bernstein's web page on SYN cookies
   [cr.yp.to]. This is a rewriting of the technical information on that
   web page and not a full replacement. There are other slightly
   different ways of implementing the SYN cookie concept than the exact
   means described here, although the basic idea of encoding data into
   the SYN-ACK sequence number is constant.

   A SYN cookie is an initial sequence number sent in the SYN-ACK, that
   is chosen based on the connection initiator's initial sequence
   number, MSS, a time counter, and the relevant addresses and port
   numbers. The actual bits comprising the SYN cookie are chosen to be
   the bitwise difference (exclusive-or) between the SYN's sequence
   number and a 32 bit quantity computed so that the top five bits come
   from a 32-bit counter value modulo 32, where the counter increases
   every 64 seconds, the next 3 bits encode a usable MSS near to the one
   in the SYN, and the bottom 24 bits are a server-selected secret
   function of pair of IP addresses, the pair of port numbers, and the
   32-bit counter used for the first 5 bits. This means of selecting an
   initial sequence number for use in the SYN-ACK complies with the rule
   that TCP sequence numbers increase slowly.

   When a connection in LISTEN receives a SYN segment, it can generate a
   SYN cookie and send it in the sequence number of a SYN-ACK, without
   allocating any other state. If an ACK comes back, the difference
   between the acknowledged sequence number and the sequence number of
   the ACK segment can be checked against recent values of the counter
   and the secret function's output given those counter values and the
   IP addresses and port numbers in the ACK segment. If there is a
   match, the connection can be accepted, since it is statistically very
   likely that the other side received the SYN cookie and did not simply
   guess a valid cookie value. If there is not a match, the connection
   can be rejected under the heuristic that it is probably not in
   response to a recently sent SYN-ACK.

   With SYN cookies enabled, a host will be able to remain responsive
   even when under a SYN flooding attack. The largest price to be paid
   for using SYN cookies is in the disabling of the window scaling
   option, which disables high performance.

   Bernstein's web page [cr.yp.to] contains more information about the
   initial conceptualization and implementation of SYN cookies, and
   archives of emails documenting this history. It also lists some
   false negative claims that have been made about SYN cookies, and
   discusses reducing the vulnerability of SYN cookie implementations to
   blind connection forgery by an attacker guessing valid cookies.

The best description of the exact SYN cookie algorithms is in a part
of an email from Bernstein, that is archived on the web site (notice
it does not set the top five bits from the counter modulo 32, as the
previous description did, but instead uses 29 bits from the second
MD5 operation and 3 bits for the index into the MSS table;
establishing the secret values is also not discussed). The remainder
of this section is excerpted from Bernstein's email [cr.yp.to]:

Here's what an implementation would involve:

Maintain two (constant) secret keys, sec1 and sec2.

Maintain a (constant) sorted table of 8 common MSS values,
msstab[8].

Keep track of a "last overflow time".

Maintain a counter that increases slowly over time and never
repeats, such as "number of seconds since 1970, shifted right 6
bits".

When a SYN comes in from (saddr,sport) to (daddr,dport) with
ISN x, find the largest i for which msstab[i] <= the incoming
MSS. Compute

   z = MD5(sec1,saddr,sport,daddr,dport,sec1)

      + x

      + (counter << 24)

      + (MD5(sec2,counter,saddr,sport,daddr,dport,sec2) % (1 <<
      24))

and then

   y = (i << 29) + (z % (1 << 29))

Create a TCB as usual, with y as our ISN. Send back a SYNACK.

Exception: _If_ we're out of memory for TCBs, set the "last
overflow time" to the current time. Send the SYNACK anyway,
with all fancy options turned off.

When an ACK comes back, follow this procedure to find a TCB:

(1) Look for a (saddr,sport,daddr,dport) TCB. If it's there,
     done.

(2) If the "last overflow time" is earlier than a few minutes
     ago, give up.

(3) Figure out whether our alleged ISN makes sense. This
     means recomputing y as above, for each of the counters
     that could have been used in the last few minutes (say,
     the last four counters), and seeing whether any of the y's
     match the ISN in the bottom 29 bits. If none of them do,
     give up.

(4) Create a new TCB. The top three bits of our ISN give a
     usable MSS. Turn off all fancy options.

Author's Address

   Wesley M. Eddy
   Verizon Federal Network Systems
   NASA Glenn Research Center
   21000 Brookpark Rd, MS 54-5
   Cleveland, OH 44135

   Phone: 216-433-6682
   EMail: weddy@grc.nasa.gov

Full Copyright Statement

Intellectual Property

Acknowledgement