Independent Submission
Request for Comments: 6013
Category: Experimental
ISSN: 2070-1721

W. Simpson
DayDreamer

January 2011

TCP Cookie Transactions (TCPCT)

Abstract

TCP Cookie Transactions (TCPCT) deter spoofing of connections and prevent resource exhaustion, eliminating Responder (server) state during the initial handshake. The Initiator (client) has sole responsibility for ensuring required delays between connections. The cookie exchange may carry data, limited to inhibit amplification and reflection denial of service attacks.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6013.

Table of Contents

1. Introduction

TCP Cookie Transactions (TCPCT) provide a cryptologically secure mechanism to guard against simple flooding attacks sent with bogus IP [RFC791] Sources or TCP [RFC793] Ports. The initial TCP <SYN> exchange is vulnerable to forged IP Addresses, predictable Ports, and discoverable Sequence Numbers [Morris1985] [Gont2009]. (See also [RFC2827], [RFC3704], and [RFC4953].)

During connection establishment, the cookie (nonce) exchange negotiates elimination of Responder (server) state. These cookies are later used to inhibit premature closing of connections, and reduce retention of state after the connection has terminated.

The cookie pair is much too large to fit with the other recommended options in the maximal 60 byte TCP header (40 bytes of option space). A successful option exchange signals availability of the TCP header extension, adding space for additional options.

Also, implementations may optionally exchange limited amounts of transaction data during the initial cookie exchange, reducing network latency and host task context switching.

Finally, implementations may optionally rapidly recycle prior connections. For otherwise stateless applications, this transparently facilitates persistent connections and pipelining of requests over each connection.

Many of these ideas have been previously proposed in one form or another (see History and Acknowledgments sections). This specification integrates these improvements into a coherent whole. Further motivation and rationale were detailed in [MSV2009].

1.1. Terminology

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

byte An 8-bit quantity; also known as "octet" in standardese.

2. Protocol Overview

The TCPCT extensions consist of several simple phases:

1. Each party passes a "cookie" to the other. Due to limited space, only the most basic options are included.

The Cookie option also indicates that optional <SYN> data is acceptable. This data MAY be ignored by either party.

A Responder that understands the Cookie option remains stateless.

2. During the remainder of the standard TCP three-way handshake, the Timestamps and Cookie-Pair options guard the exchange.

   Other options present in the original <SYN> that were successfully returned in the <SYN,ACK(SYN)> MUST be included with the <ACK(SYN)>. Additional options MAY also be included as desired.

   As there is no Responder state, it has no record of acknowledging previous data. Any optional <SYN> data MUST be retransmitted.

   Upon verification of the Timestamps and Cookie-Pair, the Responder creates its Transport Control Block (TCB) [RFC793].

   Note that the Responder returns the Cookie-Pair with its initial data, but subsequent data segments need only the Timestamps.

3. During close (or reset) of the TCP connection, the Timestamps and Cookie-Pair options guard the exchange.

   Upon verification of the Timestamps and Cookie-Pair, the Responder removes its TCB.

The sequence of messages is summarized in the diagram below.

2.1. Message Summary (Simplified)

```
   Initiator                              Responder
   =========                              =========
   <SYN>                          ->
   base options
   Timestamps
   Cookie
   [request data]
                                   <-     <SYN,ACK(SYN)>
                                          base options
                                          Timestamps
                                          Cookie
                                          [response data]
                                          (stateless)

   <ACK(SYN)>                     ->
   full options
   Timestamps
   Cookie-Pair
   [Sack(response)]
   data
                                   <-     <ACK>
                                          full options
                                          Timestamps
                                          Cookie-Pair
                                          data
                                          (TCB state created)
                                   <-     <ACK>
                                          Timestamps
                                          data

                                   <-     <FIN,ACK>
                                          Timestamps
                                          Cookie-Pair
   <FIN,ACK(FIN)>                 ->
   Timestamps
   Cookie-Pair
                                   <-     <ACK(FIN)>
                                          Timestamps
                                          Cookie-Pair
                                          (TCB state removed)
   TIME-WAIT
```

2.2. Compatibility and Transparency

It is usually better that data arrive slowly, than not at all.

Many/most unmanaged middleboxes [RFC3234] (such as stateless firewalls, load balancers, intrusion detection systems, or network address translators [RFC3022]) cannot carry transport traffic other than TCP and UDP.

Every TCP implementation MUST ignore without error any TCP option it does not implement ([RFC1122] section 4.2.2.5). In a study of the effects of middleboxes on transport protocols [MAF2004], the vast majority of modern TCP stacks correctly handle unknown TCP options. But it is still prudent to follow the [RFC793] "general principle of robustness: be conservative in what you do, be liberal in what you accept from others."

Therefore, for each of the extensions defined here, an extension option will be sent in a <SYN,ACK(SYN)> segment only after the corresponding option was received in the original <SYN> segment.

Furthermore, TCP options will be sent on later segments only after an exchange of options has indicated that both parties understand the extension (see [RFC1323] [rfc1323bis] and its antecedents).

Unfortunately, not all middleware adheres to these long-standing requirements. Instead, unknown <SYN> options are copied to the <SYN,ACK(SYN)>. This is indistinguishable from a Monkey in the Middle (MITM) reflection attack.

2.3. Fully Loaded Cookies

One Kind to aid them all, One Kind to find them,
One Kind to hold them all and in the header bind them.

The cookie exchange provides a singular opportunity to extend TCP with backward compatibility. Semantics for the option have been "overloaded" with a baker’s dozen of capabilities and facilities.

A. First and foremost, the cookie exchange improves operational security for vulnerable servers against flooding attacks. The cookie exchange indicates that the Responder (server) will discard its initial state. All other semantics are subordinate.

B. Together with Sequence and Timestamp values, Cookie values protect against insertion and reflection attacks.

C. Cookie values allow applications to detect replay attacks.

D. Cookie values MAY be used as an index or nonce for application security protocols. This facility is beyond the scope of this specification.

E. The <SYN> and <SYN,ACK(SYN)> MAY carry application data. This feature is entirely optional, and data is not guaranteed to pass successfully through middleware. Nor are the parties guaranteed to process this data without changes to the Application Program Interface (API). Such changes are beyond the scope of this specification.

F. The size of the cookies precludes most other options in the standard TCP header space. The cookie exchange negotiates TCP header extension.

G. The cookie exchange and resulting TCP header extension permit negotiation of larger 64-bit (or 128-bit) Timestamps for paths with large bandwidth-delay products.

H. TCP header extension frees some space for additional options.

I. Previously SYN-only options can be updated.

J. The cookie exchange indicates agreement to use accelerated close.

K. The cookie exchange indicates agreement that only the Initiator (client) handles TIME-WAIT state.

L. The Timestamps and Cookie-Pair combination inhibits third parties from disrupting communications with <FIN> and <RST>.

M. The Timestamps and Cookie-Pair combination facilitates rapid reuse of the TCP Source Port with a common destination.

2.4. TCP Header Extension

Once the Cookie option has been successfully exchanged, TCP header extension is permitted. The Timestamps extended option (defined below) indicates the presence of the header extension.

Validation of known timestamp values protects against data corruption by misbehaving middleboxes.

2.5. <SYN> Option Handling

As the Responder retains no TCB state after the initial TCP <SYN> exchange, all options present in the original <SYN> MUST be repeated.

For example, an option defined in the [RFC793] original specification -- Maximum Segment Size (MSS) -- previously appeared only in a <SYN> bearing segment (including <SYN,ACK(SYN)>). If present, MSS will be repeated in the Initiator <ACK(SYN)>, together with any additional options.

Generally, the Initiator MAY propose SYN-only options -- such as MSS -- anytime both Timestamps and Cookie-Pair options are present. These options are treated the same as with an original <SYN>. The Responder acknowledges using a subsequent <ACK> segment containing both Timestamps and Cookie-Pair options (similar to <SYN,ACK(SYN)> processing).

This facility allows previously SYN-only options to be updated from time to time. They take effect upon receipt.

However, <ACK> segments without data will not be delivered reliably. Any otherwise SYN-only options sent without data MUST be retransmitted with successive segments until sent with data (or <FIN>), and an <ACK> is received.

3. Protocol Details

Another solution [RFC5452] describes use of an unpredictable Source Port. That is RECOMMENDED by this specification. See [RFC6056] for further information.

An earlier solution [RFC1948] describes an unpredictable Initial Sequence Number (ISN). That is REQUIRED by this specification.

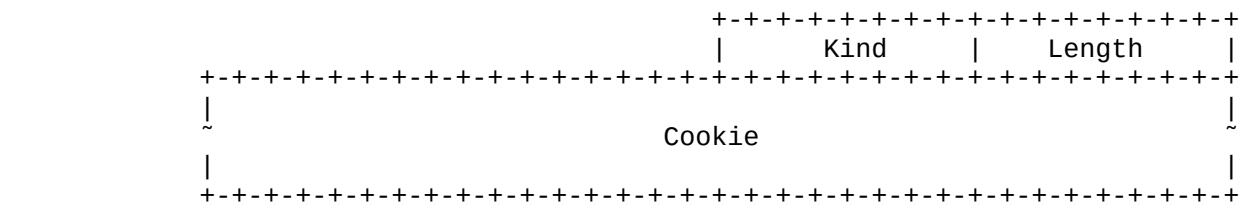
Support for the (32-bit) TCP Timestamps Option [RFC1323] is REQUIRED. A TSoffset SHOULD be generated per connection [GO2010]. The Don’t Fragment (DF) bit MUST be set in the IP (v4) header.

The TCP User Timeout Option [RFC5482] is RECOMMENDED.

Only one instance is permitted of any of the Cookie, Cookie-less, or Cookie-Pair option(s). Segments with duplicative or mutually exclusive options MUST be silently discarded.

For examples, see Appendix A.

3.1. TCP Cookie Option



Kind 1 byte: constant 253 (experimental).

Length 1 byte: range 10 to 18 (bytes); limited by remaining space in the options field. The number MUST be even; the cookie is a multiple of 16 bits.

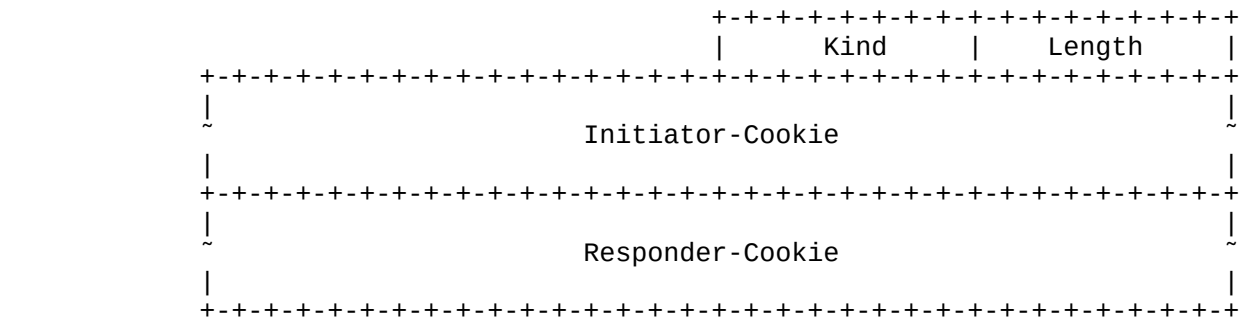
Cookie 8 to 16 bytes (Length - 2): an unpredictable value.

Options with invalid Length values MUST be ignored. The minimum Cookie size is 64 bits. If there is not sufficient space for a 64-bit cookie, this option MUST NOT be used.

The Responder Cookie MUST be the same size as the Initiator Cookie. The cookie pair is a multiple of 32 bits.

Although the diagram shows a cookie aligned on 32-bit boundaries, that is not required.

3.2. TCP Cookie-Pair Standard Option



Kind 1 byte: constant 253 (experimental).

Length 1 byte: range 18 to 34 (bytes). The number MUST be even; the cookie pair is a multiple of 32 bits.

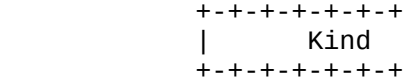
Initiator-Cookie 8 to 16 bytes, from the original <SYN>.

Responder-Cookie 8 to 16 bytes, from the <SYN,ACK(SYN)>.

The Cookie-Pair standard option only appears after the Timestamps extended option (below).

Options with invalid Length values MUST be ignored. As the minimum Initiator-Cookie size is 64 bits, the minimum cookie pair is 128 bits (64 bits followed by 64 bits), while the maximum is 256 bits (128 bits followed by 128 bits).

3.3. TCP Cookie-less Option

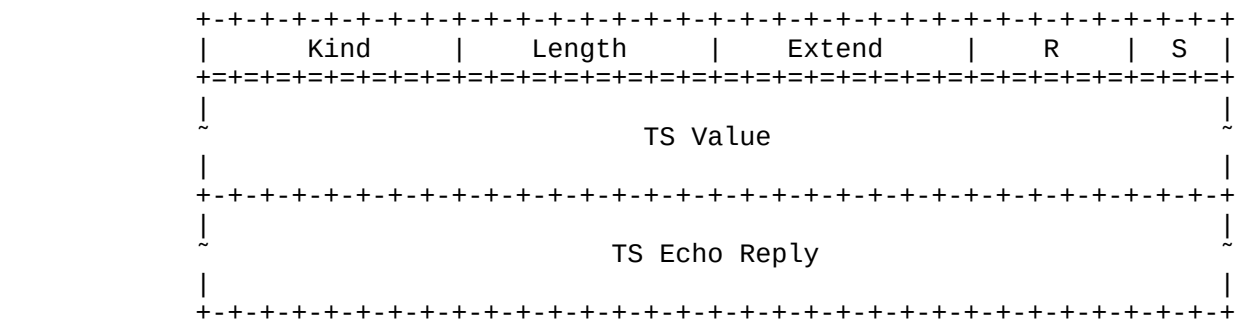


Kind 1 byte: constant 253 (experimental).

Length 1 byte: constant 2 (bytes). This distinguishes the option from other Cookie options.

Although no cookie is attached, this indicates that other features of this specification are available, including TCP header extension, Accelerated Close, Accelerated Open, and Advisory Reset. This is intended for use with TCP authentication options, beyond the scope of this specification.

3.4. TCP Timestamps Extended Option



Kind 1 byte: constant 254 (experimental).

Length 1 byte: constant 4 (bytes).

Extend	1 byte: range 9 to 255; the data offset (in 32-bit words) following the standard TCP header. Note this value MUST include the timestamp pair indicated by (S)ize.
(R)eserved	5 bits: default zero. Reserved for future use.
(S)ize	3 bits: 1. 32-bit timestamps. 2. 64-bit timestamps. 4. 128-bit timestamps. Other values are beyond the scope of this specification.
TS Value	4, 8, or 16 bytes. The current value of the timestamp for the sender.
TS Echo Reply	4, 8, or 16 bytes. A copy of the most recently received TS Value.

The full timestamp pair follows the TCP header (indicated by +=+ delimiters) and maintains 32-bit alignment.

This TCP header extension is ignored for sequence number computations. The Sequence Number of the first byte of segment data will be the Initial Sequence Number (ISN) plus one (1) for the <SYN>.

Every TCPCT implementation MUST recognize a Timestamps extended option. The larger 64-bit (or 128-bit) timestamps only appear in an extended option.

Segments with invalid Extend values MUST be silently discarded.

Only one instance is permitted of either the (32-bit) Timestamps standard option or this Timestamps extended option. Segments with duplicative or mutually exclusive options MUST be silently discarded.

Implementation Notes:

Serendipitous alignment allows simple loads and stores, instead of slower byte by byte iterations.

When the TCP header is aligned on a 32-bit boundary and this is the only option, the timestamps in the extended header SHOULD be aligned on a 64-bit boundary. For both 32-bit and 64-bit timestamps, any data following the extended header will be aligned on a 64-bit boundary.

However, the 128-bit timestamps are not 128-bit aligned.

3.5. Cookie Generation

The technique by which a party generates a cookie is implementation dependent. The method chosen must satisfy some basic requirements:

1. The cookie MUST depend on the specific parties. This prevents an attacker from obtaining a cookie using a real IP address and TCP port, and then using it to swamp the victim with requests from randomly chosen IP addresses or ports.

2. It MUST NOT be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local secret information in the generation and subsequent verification of a cookie. It must not be possible to deduce this secret information from any particular cookie.

3. The cookie generation and verification methods MUST be fast to thwart attacks intended to sabotage CPU resources.

A recommended technique is to use a cryptographic hashing function.

An incoming cookie can be verified at any time by regenerating it locally from values contained in the incoming datagram and the local secret random value.

3.5.1. Initiator Cookie

The Initiator secret value that affects its cookie SHOULD change for each new exchange, and is thereafter internally cached per TCB. This provides improved synchronization and protection against replay attacks.

An alternative is to cache the cookie instead of the secret value. Incoming cookies can be compared directly without the computational cost of regeneration.

It is RECOMMENDED that the cookie be calculated over the secret value, the IP Source and Destination addresses, the TCP Source and Destination ports, and any (optional) Initiator <SYN> segment data.

Implementation Notes:

Although the recommendation includes the TCP Source Port, this is very implementation specific. For example, it might not be included when the value is constant or unknown.

Likewise, segment data might not be included directly. For example, a pointer to the data could be included instead, with care taken to ensure the pointer changes anytime the data changes.

However, it is important that the implementation protect mutually suspicious users of the same system from generating the same cookie.

3.5.2. Responder Cookie

The Responder secret value that affects its cookies remains the same for many different Initiators. However, this secret SHOULD be changed periodically to limit the time for use of its cookies (typically each 600 seconds).

The Responder-Cookie calculation MUST include its own TCP Sequence and Acknowledgment Numbers (after updating values), its own TCP Timestamps value, and the Initiator-Cookie value. This provides improved synchronization and protection against replay attacks.

It is RECOMMENDED that the cookie be calculated over the secret value, the IP Source and Destination addresses, its own TCP Destination Port (that is, the incoming Source Port), and the required values (above), followed by the secret value again.

The cookie is not cached per Initiator to avoid saving state during the initial TCP <SYN> exchange. On receipt of a TCP <ACK(SYN)>, the Responder regenerates its cookie for validation.

Implementation Notes:

Although the recommendation does not include the TCP Source Port, this is very implementation specific. It might be successfully included in some variants.

The Responder Cookie depends on the TCP Sequence and Acknowledgment Numbers as they will appear for future verification. The Sequence Number will be the Initial Sequence

Number (ISN) plus one (1) for its <SYN> that will be acknowledged. The Acknowledgment Number will be the Initial Sequence Number (ISN) plus one (1) for the <SYN> that it is now acknowledging.

The (32-bit) TCP Timestamps standard option MAY change to the larger 64-bit (or 128-bit) extended form; only the least significant 32 bits are included. The Initiator Timestamp field value MAY increment during the exchange; it MUST NOT be included.

The secret value is included twice to better protect against pre-calculated attacks using substitutions for variable length data. Some examples using this technique are IP-MAC and H-MAC, and it is likely that existing code could be shared.

The Responder SHOULD designate a (fixed or randomly selected) bit of its cookie to distinguish each changed secret value. The bit is set to a (fixed or randomly selected) constant 0 or 1, and checked upon receipt before further verification. This ensures that only one verification calculation is necessary (on average) during Denial of Service (DoS) attacks.

If a Responder Cookie is identical to the Initiator Cookie, the Responder SHOULD change one or more bits of its cookie to prevent its accidental appearance as a reflection attack.

3.5.3. Responder Secret Value

Each Responder maintains up to two secret values concurrently for efficient secret rollover. Each secret value has 4 states:

Generating
: Generates new Responder-Cookies, but not yet used for primary verification. This is a short-term state, typically lasting only one Round Trip Time (RTT).

Primary
: Used both for generation and primary verification.

Retiring
: Used for verification, until the first failure that can be verified by the newer Generating secret. At that time, this cookie’s state is changed to Secondary, and the Generating cookie’s state is changed to Primary. This is a short-term state, typically lasting only one RTT.

Secondary
Used for secondary verification, after primary verification failures. This state lasts no more than twice the Maximum Segment Lifetime (2MSL). Then, the secret is discarded.

Implementation Notes:

Care MUST be taken to ensure that any expired secrets are promptly wiped from memory, and secrets are never saved to external storage.

The first secret after initialization begins in Primary state. The system might have shutdown and restarted rapidly during the previous first secret. Thus, the first secret MUST be partially time dependent, to ensure that it differs from previous first secrets, usually by appending a time to lengthen the first secret. Those that are not the first secret SHOULD NOT include the time.

At the same time, there is no TCP TIME-WAIT requirement before accepting connections, and there may be pent up demand for a busy service. Also, there may be outstanding datagrams attempting to complete an earlier cookie exchange. The first secret is likely to be the weakest, as no recent entropy has been included.

Therefore, while terminating outstanding exchanges with the first secret, a new Generating secret SHOULD be created after no more than one Maximum Segment Lifetime (1MSL). Subsequent secrets SHOULD be generated at the usual rate (typically 600 seconds).

The implementation SHOULD continually gather additional entropy from checksums, cookies, timestamps, and packet arrival timing.

4. Cookie Exchange

A successful option exchange signals availability of additional features.

4.1. Initiator <SYN>

The Cookie exchange MAY be initiated at any time, limited only by the frequency of the timestamp clock.

If the TCB exists from a prior (or ongoing) connection, the timestamp MUST be incremented in the option.

The Initiator generates its unpredictable cookie value, and includes the Cookie option.

During the initial exchange, the Initiator is solely responsible for retransmission. Although the cookie and sequence have not changed, each retransmission appears to the Responder as another original <SYN>.

Implementation Notes:

Sending the <SYN> SHOULD NOT affect any existing TCB. This allows an additional RTT for duplicate or out-of-sequence segments to drain.

The new TCB information SHOULD be temporarily cached until a valid matching <SYN,ACK(SYN)> arrives. Then, any old TCB values are replaced.

4.2. Responder <SYN,ACK(SYN)>

Upon receipt of the <SYN> with a Cookie option, the Responder determines whether there are sufficient resources to begin another connection.

If the TCB exists from a prior (or ongoing) connection, the timestamp MUST be incremented in the option.

Each Sequence Number MUST be randomized [RFC1948].

The Responder generates its unpredictable cookie value, and includes the Cookie option.

As the Responder retains no TCB state, retransmission timers are not available. Arrival of an Initiator's retransmission appears to be an original <SYN> transmission. There are no differences in processing.

Implementation Notes:

Sending the <SYN,ACK(SYN)> MUST NOT affect any existing TCB. This allows an additional RTT for duplicate or out-of-sequence segments to drain.

This also inhibits third parties from disrupting communications.

4.3. Initiator <ACK(SYN)>

Upon receipt of the <SYN,ACK(SYN)> with a Cookie option, the Initiator validates its cookie, timestamp, and corresponding Acknowledgment Number. The existing TCB is updated as necessary.

All Initiator <SYN> options are always retransmitted on this first <ACK(SYN)>, allowing the Responder to validate its cookie and establish its state.

This segment contains both Timestamps and Cookie-Pair options.

The Initiator sends the Timestamps extended option with an appropriate Size -- chosen by a configurable parameter, or automatically based on its analysis of the bandwidth-delay product discovered through the RTT of its <SYN> timestamp. When the chosen Size is greater than 32 bits, the Initiator adds a random prefix to its own timestamp, and a random prefix to the Responder timestamp echo reply.

Implementation Notes:

A Responder Cookie identical to the Initiator Cookie MUST be discarded. This is usually an indication of a Monkey in the Middle (MITM) reflection attack or a seriously misconfigured network, and SHOULD be logged.

4.4. Responder <ACK>

Upon receipt of the <ACK(SYN)> with a Cookie-Pair option, the Responder validates its cookie, timestamp, and corresponding Acknowledgment Number, and establishes state for the connection. Any existing TCB is updated as necessary.

This segment contains both Timestamps and Cookie-Pair options.

However, the Responder MAY refuse to negotiate the larger 64-bit (or 128-bit) Timestamps extended option by returning the least significant bits in a smaller Timestamps extended option.

Implementation Notes:

An <ACK(SYN)> that fails to validate MUST be discarded, and SHOULD be logged.

4.5. Simultaneous Open

TCP allows two parties to simultaneously initiate the connection. Both parties send and receive an original <SYN> without an intervening <SYN,ACK(SYN)> (see [RFC793] section 3.4 and Figure 8). Each party receives a Cookie for a <Source Address, Source Port, Destination Address, Destination Port> connection that has also issued a Cookie.

This condition will be unusual. The Source Port SHOULD be randomized [RFC5452], and SHOULD be chosen to differ from the Destination Port. In particular, the Source Port SHOULD be greater than 1024, preventing intervening network equipment from incorrectly classifying the return traffic. The Destination Port is most likely to be a well-known port less than 1024 [RFC3232].

In the event that these protections are insufficient, the conflict is resolved in an orderly fashion:

a. The lesser TCP Port number becomes the Responder;

b. The lesser IP Address becomes the Responder;

c. The lesser Cookie becomes the Responder;

d. All of the above being equal, there is an egregiously insufficient source of randomness, but both Initiators are probably present on the same host: the lesser TCB memory address becomes the Responder.

The Initiator silently discards the simultaneous <SYN>. The Responder revises its Cookie option, and sends the <SYN,ACK(SYN)> as usual, but without removing its existing TCB.

Implementation Notes:

This is usually an indication of a Monkey in the Middle (MITM) reflection attack or a seriously misconfigured network, and SHOULD be logged.

5. Accelerated Close

Support for accelerated close is REQUIRED. Accelerated close relies on the presence of cookies and timestamps. This provides improved synchronization and protection against replay attacks.

Either party MAY close with <FIN> at any time. This <FIN> SHOULD be sent with the final data segment.

This segment contains both Timestamps and Cookie-Pair options.

When all segments preceding the <FIN> have been processed and acknowledged, each party SHOULD acknowledge the <FIN>.

In general, <FIN> is treated as advisory. A persistent connection can be rapidly re-established. This also inhibits third parties from disrupting communications.

Rapidly closing the connection expedites removing Responder state. Any <FIN> bearing segment SHOULD terminate delayed <ACK> [RFC5681]. Retransmit at the latest Timestamps estimated Smoothed Round Trip Time (SRTT). Backoff SHOULD NOT be used for <FIN> bearing retransmissions [RFC2988].

As the Responder retains no TCB state after closing, a successful option exchange signals the Initiator will be responsible for handling TIME-WAIT state. (For previous proposal and rationale, see [FTY1999] section 3.)

A new Cookie exchange MAY be initiated at any time. This facilitates persistent connections through intervening network equipment.

5.1. Initiator Close

Upon receipt of the Initiator <FIN> (and verification of the Timestamps and Cookie-Pair options), the Responder sends its <FIN,ACK(FIN)> unless there is additional data pending. In the latter case, the <FIN> is ignored until the data has been processed and acknowledged.

Upon receipt of the Responder <FIN,ACK(FIN)> (and verification of the Timestamps and Cookie-Pair options), the Initiator sends its final <ACK(FIN)> unless there is additional data pending. The Initiator enters TIME-WAIT state.

This segment contains both Timestamps and Cookie-Pair options.

Upon receipt of the Initiator <ACK(FIN)> (and verification of the Timestamps and Cookie-Pair options), the Responder removes its TCB.

Upon arrival of more data prompting a new Cookie exchange, the Initiator SHOULD NOT send a final <ACK(FIN)> and/or SHOULD NOT wait the remaining TIME-WAIT interval. Any existing TSoffset SHOULD be incremented. TSoffset will be removed (with the TCB itself) at the conclusion of a future TIME-WAIT state.

5.2. Responder Close

Upon receipt of the Responder <FIN> (and verification of the Timestamps and Cookie-Pair options), the Initiator sends its <FIN,ACK(FIN)> unless there is additional data pending. In the latter case, the <FIN> is ignored until the data has been processed and acknowledged.

Upon receipt of the Initiator <FIN,ACK(FIN)> (and verification of the Timestamps and Cookie-Pair options), the Responder sends its final <ACK(FIN)> and removes its TCB.

This segment contains both Timestamps and Cookie-Pair options.

If the Responder's final <ACK(FIN)> is lost, the Responder is likely to send a <RST> (as the Responder retains no TCB state). This distinguished <RST> SHOULD copy both Timestamps and Cookie-Pair options.

Upon receipt of the Responder's final <ACK(FIN)> (and verification of the Timestamps and Cookie-Pair options), the Initiator enters TIME-WAIT state.

Upon arrival of more data prompting a new Cookie exchange, the Initiator SHOULD NOT send a <FIN,ACK(FIN)> and/or SHOULD NOT wait the remaining TIME-WAIT interval. Any existing TSoffset SHOULD be incremented. TSoffset will be removed (with the TCB itself) at the conclusion of a future TIME-WAIT state.

6. Accelerated Open

Support for accelerated open is OPTIONAL.

When an application is capable of idempotent transactions (such as a query that returns a consistent result or service response heading), the application sets the appropriate limit separately for each port or connection. Applications are responsible for ensuring that retransmissions do not cause duplication of data.

This facility allows single data segment transactions without establishing TCB state at the Responder (server). For longer transactions, a short look-ahead of upcoming data allows the Initiator (client) to select alternatives for further processing.

6.1. Initiator <SYN> Data

By default, the Initiator <SYN> does not contain data. The application sets the TCP_SYN_DATA_LIMIT to indicate that the <SYN> MAY be sent with data.

The Responder Maximum Segment Size (MSS) is unknown, and the default MSS (536 bytes) MUST be used instead ([RFC1122] section 4.2.2.6). This is further reduced by the total length of the TCP options (in this case, commonly 496 bytes). Applications MAY specify a shorter limit.

If the data will not entirely fit within the initial segment, data MUST NOT be sent until after the Responder's <SYN,ACK(SYN)> is received.

Unlike T/TCP [RFC1644], <FIN> SHOULD NOT be sent with <SYN> data. This facilitates persistent connections.

Likewise, <PSH> SHOULD NOT be set. Although the application might use push to indicate that its data is ready to send, the push is implied for <SYN> data segments.

During the initial exchange, the Initiator is solely responsible for retransmission. Although the cookie and sequence have not changed, each retransmission appears to the Responder as another original <SYN>.

Implementation Notes:

Initiator <SYN,FIN> with the Cookie option and no segment data is permitted in a test environment. This combination SHOULD be silently discarded.

Initiator <SYN,FIN> with both the Cookie option and segment data is similar to T/TCP [RFC1644]. However, whenever the Responder <SYN,ACK(SYN),FIN> has been sent with data (there is no further data expected), TCB state has not been saved at the Responder. There is no need to send <FIN> to close the connection.

6.2. Responder <SYN,ACK(SYN)> Data

By default, the Responder <SYN,ACK(SYN)> does not contain data. The application sets the TCP_SYN_ACK_DATA_LIMIT to indicate that the <SYN,ACK(SYN)> MAY be sent with data.

Segment data is limited to the Maximum Transmission Unit (MTU). Applications MAY specify a shorter limit to prevent spoofed amplification and reflection attacks [RFC5358].

Upon receipt of the <SYN> with a Cookie option, the Responder MAY process any data present. If the initial data is not accepted, the Acknowledgment Number will be the received Sequence Number plus one (1) for the <SYN>.

If the segment data is the entire response (there is no further data expected), <FIN> MAY be set.

However, <PSH> SHOULD NOT be set. Although the application might use push to indicate that its data is ready to send, the push is implied for <FIN> data segments (see [RFC793] section 3.7, page 41).

As the Responder retains no TCB state, retransmission timers are not available. Arrival of an Initiator's retransmission appears to be an original <SYN> transmission. There are no differences in processing.

Implementation Notes:

The Responder Cookie depends on the TCP Sequence and Acknowledgment Numbers after processing <SYN>. Therefore, neither will include data.

6.3. Initiator <ACK(SYN)> Data

Upon receipt of the <SYN,ACK(SYN)> with a Cookie option, the Initiator MAY process any data present. In this case, the internal RCV.NXT is advanced to provide at-most-once semantics.

If the segment data is the entire response (there is no further data expected), the Initiator enters TIME-WAIT state.

Otherwise, original <SYN> data is retransmitted in <ACK(SYN)>, as its processing is optional. The Acknowledgment Number will be the received Sequence Number plus one (1) for the <SYN>. The Sequence Number will be the Initial Sequence Number (ISN) plus one (1) for the <SYN>.

Unlike T/TCP [RFC1644], there is no implicit acknowledgment.

If the Selective Acknowledgment (Sack) option [RFC2018] has been successfully negotiated, a short Sack acknowledging the response data MAY be sent following the Cookie-Pair in the extended header.

At this time, any second segment may be sent without awaiting an <ACK>, according to the usual [RFC5681] TCP congestion control process.

Implementation Notes:

Upon arrival of more data prompting a new Cookie exchange, there is no need to increment the previous timestamp; TCB state has not been saved at the Responder. Instead, use the saved RCV.NXT, plus one (1) for the (actual or implied) <FIN>.

Initiator <ACK(SYN),FIN> with the Cookie-Pair option and no segment data is never required; TCB state has not been saved at the Responder. This combination MUST be silently discarded.

6.4. Responder <ACK> Data

Upon receipt of the <ACK(SYN)> with a Cookie-Pair option (and verification of the Timestamps and Cookie-Pair options), the Responder SHOULD process any data present.

Since the TCP Sequence and Acknowledgment Numbers have not advanced, the Responder will process the same incoming data, and transmit the same response.

If the Selective Acknowledgment (Sack) option [RFC2018] has been successfully negotiated, with a short Sack covering earlier response data, only additional unacknowledged response data is sent.

At this time, any second segment may be sent without awaiting an <ACK>, according to the usual [RFC5681] TCP congestion control process.

7. Advisory Reset

When a TCB with matching Addresses and Ports is found, but the Cookie-Pair fails to verify, the datagram MUST be silently discarded.

When no TCB with matching Addresses and Ports is found, a <RST> is sent as usual. The Timestamps option SHOULD be copied [RFC1323]. A Cookie-Pair option MUST also be copied. The Cookie option (or Cookie-less option) MUST NOT be copied.

Any <RST> is always treated as advisory. A <RST> without a matching Cookie-Pair option could be caused by antique duplicates. Receipt has no effect on the operation of the protocol. The implementation SHOULD continue until a USER TIMEOUT expires. (See [RFC5482] for additional information.)

This also inhibits third parties from disrupting communications.

8. Interactions with Other Options

A successful Cookie (or Cookie-less) option exchange signals availability of the TCP header extension. Other options with large data portions MAY also use this feature. The extended option data is processed in the order that the options appear.

8.1. TCP Selective Acknowledgment

(Kind 5 [RFC2018].) The pairs of 32-bit fields are well suited to the header extension. Because of its variable size, this is RECOMMENDED as the final extended option.

During the cookie exchange, the <ACK(SYN)> MAY include this option to acknowledge any optional transaction response data.

8.2. TCP Timestamps

(Kind 8 [RFC1323].) Support is REQUIRED. See also section 3.

When a segment needs no header extension, and 32-bit timestamps have been negotiated, this option MUST be sent.

8.3. TCP Extensions for Transactions

(Kinds 11-13 [RFC1644].) Incompatible with this specification, and MUST be ignored on receipt.

8.4. TCP MD5 Signature

(Kind 19 [RFC2385].) This option is beyond the scope of this specification. Because specific configuration is required, sending is under the complete control of the operator. Segments lacking this option will be silently discarded.

The size of the option itself precludes use with the Cookie option in the <SYN>. Regardless of the system default, the Cookie option MUST NOT be sent, and MUST be ignored on receipt. Instead, the Cookie-less extension option indicates that other features of this specification are available.

8.5. TCP Authentication

(Kind 29 [RFC5925].) This option is beyond the scope of this specification. Because specific configuration is required, sending is under the complete control of the operator. Segments lacking this option will be silently discarded.

The size of the option itself precludes use with the Cookie option in the <SYN>. Regardless of the system default, the Cookie option MUST NOT be sent, and MUST be ignored on receipt. Instead, the Cookie-less extension option indicates that other features of this specification are available.

9. History

T/TCP [RFC1379] [RFC1644] permits lightweight TCP transactions for applications that traditionally have used UDP. However, T/TCP has unacceptable security issues [Hannum1996] [Phrack1998].

The initial specification [KS1995] of Photuris [RFC2522], now called version 1 (December 1994 to March 1995), was based on a short list of design requirements, and simple experimental code by Phil Karn. A "Cookie" Exchange guards against simple flooding attacks sent with bogus IP Sources or UDP Ports.

During 1995, the Photuris efficient secret rollover and many other extensions were specified. Multiple interoperable implementations were produced.

By September 1996, the long anticipated Denial of Service (DoS) attacks in the form of TCP SYN floods were devastating popular (and unpopular) servers and sites. Phil Karn informally mentioned adapting anti-clogging cookies to TCP. Perry Metzger proposed adding Karn's cookies as part of a "TCP++" effort [Metzger1996].

Later in 1996, Daniel J. Bernstein implemented "SYN cookies", small cookies embedded in the TCP SYN Initial Sequence Number (ISN). This technique was exceptionally clever, because it did not require cooperation of the remote party and could be deployed unilaterally. However, SYN cookies can only be used in emergencies; they are incompatible with most TCP options. As there is insufficient space in the Sequence Number, the cookie is not considered cryptologically secure. Therefore, the mechanism remains inactive until the system is under attack, and thus is not well tested in operation. SYN cookies were not accepted for publication until recently [RFC4987].

In 1998, Perry Metzger proposed adding Karn's cookies as part of a "TCPng" discussion [Metzger1998].

In 1999, Faber, Touch, and Yue [FTY1999] proposed using an option to negotiate the party that would maintain TIME-WAIT state. This permits a server to entirely eliminate state after closing a connection.

In 2000, the Stream Control Transmission Protocol (SCTP) [RFC2960] was published with an inadequate partial cookie mechanism claiming to be based upon Photuris. It featured a deficient checksum (replaced in 2002 by [RFC3309] without graceful transition), and has undergone subsequent revisions [RFC4960].

In 2006, the Datagram Congestion Control Protocol (DCCP) [RFC4340] was published with a mechanism analogous to SYN cookies.

10. Acknowledgments

Andre Broido informally described utilizing cookies for Transport Layer Security (TLS) session identifiers, in place of the [RFC5077] ticket. Rapid TLS session resumption would improve both latency and privacy, but is beyond the scope of this specification. Also, he provided numerous helpful comments and additional references, such as [KBC2005].

H. K. Jerry Chu and Arvind Jain informally described retaining existing cookies for accelerated open on subsequent connections. That feature was subsumed by this specification.

Wesley M. Eddy and Adam Langley previously proposed another pair of options [EL2008] extending the TCP header option space.

Adam Langley previously proposed another option [Langley2008] permitting <SYN,ACK(SYN)> constant payload data. His (August 2008) code was a base for the initial TCPCT implementation.

Joe Touch postulated a (hopefully hypothetical) failure mode: options re-ordered by middleware. This caused a change in specifications, and has considerably complicated option interactions and processing. His helpful comments were appreciated.

Many thanks to Fernando Gont for suggestions, and Rick Jones for performance testing.

11. IESG Considerations

Two TCP Option numbers are reserved for general experimental use under the rules laid out in [RFC4727] and [RFC3692] section 1. Such values reserved for experimental use are never to be made permanent; permanent assignments should be obtained through standard processes. Experimental numbers are intended for experimentation and testing and are not intended for wide or general deployments.

For further information, contact the author.

12. Operational Considerations

Any implementation of this specification SHOULD be configurable, separately for each port or connection.

TCPCT_COOKIE_DESIRED
: Values: 0 (disabled), 8, 10, 12, 14, 16. Default: 16. Send the Cookie option with the <SYN>.

TCPCT_EXTEND_TS[32|64|128]
: Default: off. If defined, may designate 32-bit, 64-bit, or 128-bit timestamps extension.

TCPCT_IN_ALWAYS
: Default: off. Silently discard any incoming <SYN> that is missing the Cookie option.

TCPCT_OUT_NEVER
: Default: off. Refuse to send (override) the Cookie option.

TCP_SYN_DATA_LIMIT
: Default: 0. Maximum: 496. The maximum amount of data transmitted with the <SYN>. Wait for data before sending.

TCP_SYN_ACK_DATA_LIMIT
: Default: 0. Maximum: 1220. The maximum amount of data transmitted with the <SYN,ACK(SYN)>. Wait for data before sending.

13. Security Considerations

TCPCT was based on currently available tools, by experienced network protocol designers with an interest in cryptography, rather than by cryptographers with an interest in network protocols. This specification is intended to be readily implementable without requiring an extensive background in cryptology.

Therefore, only minimal background cryptologic discussion and rationale is included in this document. Although some review has been provided by the general cryptologic community, it is anticipated that design decisions and tradeoffs will be thoroughly analysed in subsequent dissertations and debated for many years to come. Cryptologic details are reserved for separate documents that may be more readily and timely updated with new analysis.

The security depends on the quality of the random numbers generated by each party. Generating cryptographic quality random numbers on a general purpose computer without hardware assistance is a very tricky problem (see [RFC4086] for discussion).

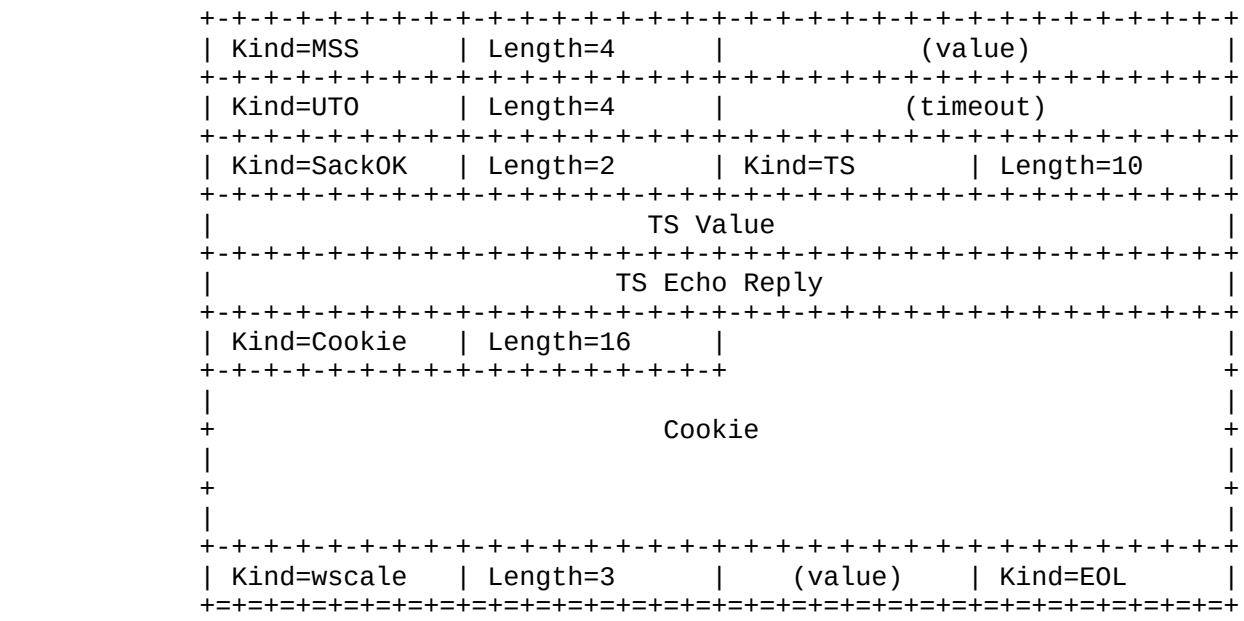
TCPCT is not intended to prevent or recover from all possible security threats. Rather, it is designed to inhibit inadvertent middlebox interference, while protecting against Denial of Service (DoS) attacks. (See [RFC4732], and [RFC3552] section 4.6.3 et seq.)

The cookie exchange does not protect against an interloper that can race to substitute another value, nor an interceptor that can modify and/or replace a value. These attacks are considerably more difficult than passive vacuum-cleaner monitoring.

Note that each incoming <SYN,ACK(SYN)> replaces the Responder cookie. The initial exchange is most fragile, as protection against spoofing relies entirely upon the sequence and timestamp. This replacement strategy allows the correct pair to pass through, while any others will be filtered via Responder verification later.

Appendix A. Example Headers

A.1. Example <SYN>

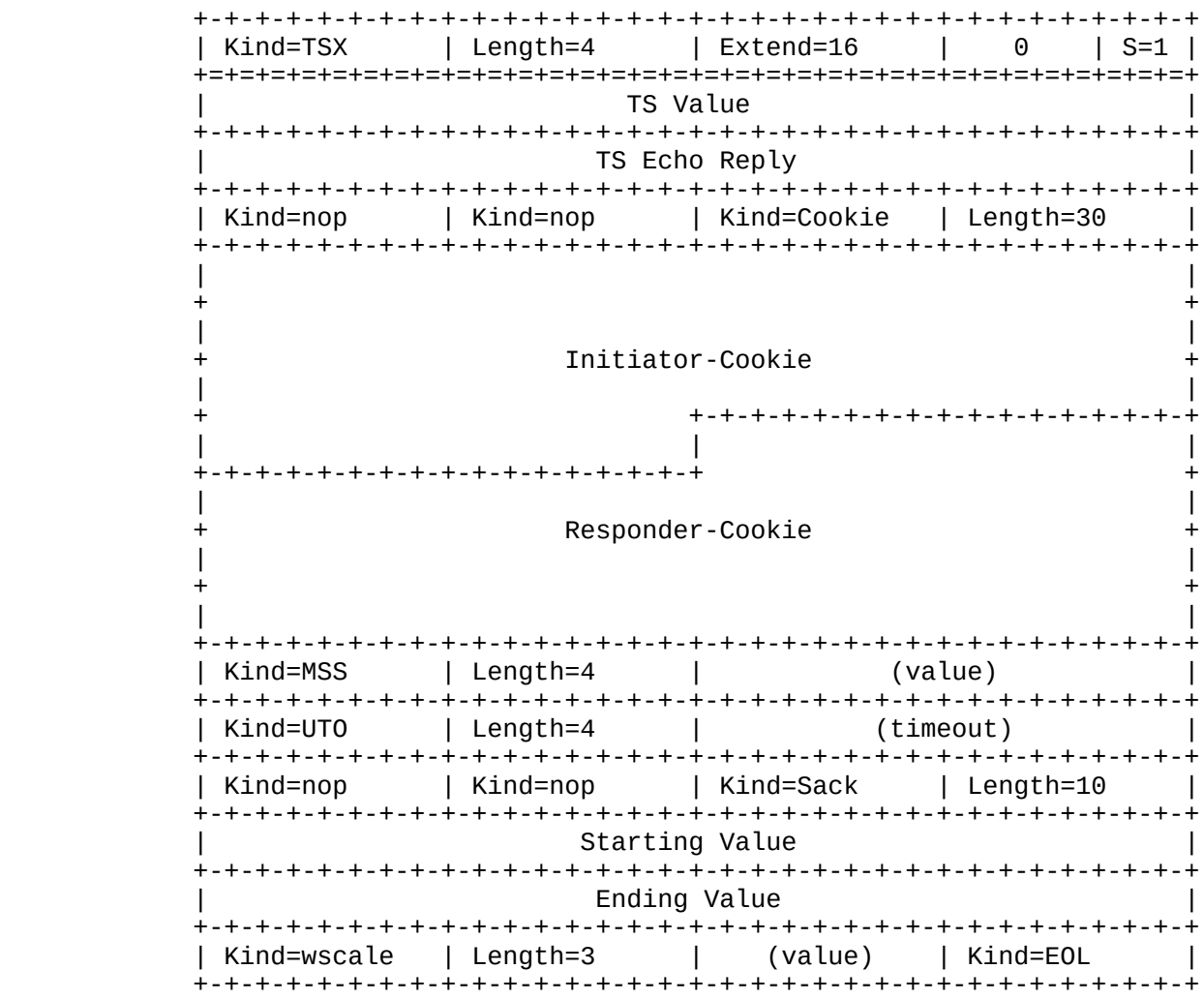


A 14 byte (112-bit) Cookie barely fits with the other recommended options in the maximal 60 byte TCP header (40 bytes of option space).

Since the cookies are required to be the same size and meet a 32-bit alignment requirement, the implementor recognizes that this order provides optimal packing.

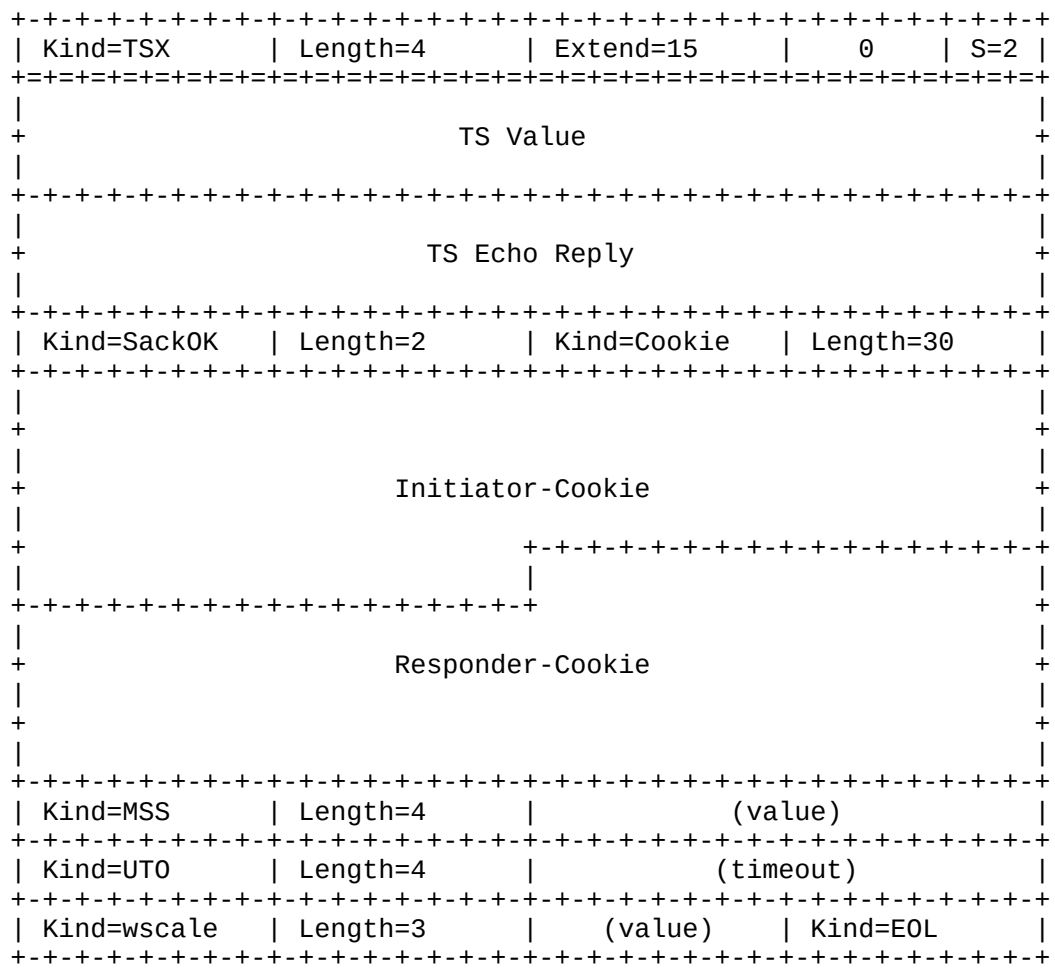
The UserTimeOut (UTO) option can appear in other locations instead, such as following the Cookie option. Because some middleboxes are sensitive to the order of options, UTO should not appear before MSS nor between the TS and Cookie.

A.2. Example <ACK(SYN)> with Sack



Sack implies SackOK.

A.3. Example <ACK(SYN)> with 64-bit Timestamps



The larger 64-bit (or 128-bit) Timestamps extended option MUST be recognized, although the Responder MAY return a smaller Timestamps extended option.

Normative References

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

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

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

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

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

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

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

[RFC2988] Paxson, V. and M. Allman, "Computing TCP’s Retransmission Timer", RFC 2988, November 2000.

[RFC3232] Reynolds, J., Ed., "Assigned Numbers: RFC 1700 is Replaced by an On-line Database", RFC 3232, January 2002.

[RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers", RFC 5452, January 2009.

[RFC5482] Eggert, L. and F. Gont, "TCP User Timeout Option", RFC 5482, March 2009.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

Informative References

[EL2008] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", Work in Progress, July 2008.

[FTY1999] Faber, T., Touch, J., and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", IEEE INFOCOM 99, pp. 1573-1584.

[Gont2009] Gont, F., "Security assessment of the Transmission Control Protocol (TCP)", February 2009. https://www.cpni.gov.uk/Docs/tn-03-09-security-assessment-TCP.pdf

[GO2010] Gont, F. and A. Oppermann, "On the generation of TCP timestamps", Work in Progress, June 2010.

[Hannum1996] Hannum, C., "Security Problems Associated With T/TCP", unpublished work in progress, September 1996. http://www.mid-way.org/doc/ttcp-sec.txt

[KBC2005] Kohno, T., Broido, A., and K. C. Claffy, "Remote physical device fingerprinting", IEEE Symposium on Security and Privacy, May 2005. http://www.caida.org/outreach/papers/2005/fingerprinting/KohnoBroidoClaffy05-devicefingerprinting.pdf

[KS1995] Karn, P. and W. Simpson, "The Photuris Session Key Management Protocol", March 1995.

Published as: "Photuris: Design Criteria", Proceedings of Sixth Annual Workshop on Selected Areas in Cryptography, LNCS 1758, Springer-Verlag. August 1999.

[Langley2008] Langley, A., "Faster application handshakes with SYN/ACK payloads", Work in Progress, August 2008.

[MAF2004] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", Proceedings 4th ACM SIGCOMM/USENIX Conference on Internet Measurement, October 2004. http://www.icsi.berkeley.edu/pubs/networking/tbit-Aug2004.pdf

[Metzger1996]
Metzger, P., "Re: SYN floods (was: does history repeat itself?)", September 9, 1996. http://www.merit.net/mail.archives/nanog/1996-09/msg00235.html

[Metzger1998]
Metzger, P., "Re: what a new TCP header might look like", May 12, 1998. ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail

[Morris1985]
Morris, R., "A Weakness in the 4.2BSD Unix TCP/IP Software", Technical Report CSTR-117, AT&T Bell Laboratories, February 1985. http://pdos.csail.mit.edu/~rtm/papers/117.pdf

[MSV2009] Metzger, P., Simpson, W., and P. Vixie, "Improving TCP Security With Robust Cookies", Usenix ;login:, December 2009. http://www.usenix.org/publications/login/2009-12/openpdfs/metzger.pdf

[Phrack1998]
route|daemon9, "T/TCP vulnerabilities", Phrack Magazine, Volume 8, Issue 53, July 8, 1998. http://www.phrack.org/issues.html?issue=53&id=6

[RFC1379] Braden, R., "Extending TCP for Transactions -- Concepts", RFC 1379, November 1992.

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

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

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

[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.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, January 2001.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[RFC3309] Stone, J., Stewart, R., and D. Otis, "Stream Control Transmission Protocol (SCTP) Checksum Change", RFC 3309, September 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

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

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC4727] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and Internet Architecture Board, "Internet Denial-of-Service Considerations", RFC 4732, November 2006.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, October 2008.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[RFC6056] Larson, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, January 2011.

[rfc1323bis] Borman, D., Braden, R., and V. Jacobson., "TCP Extensions for High Performance", Work in Progress, March 2009.

Author's Address

Questions about this document can be directed to:

William Allen Simpson
DayDreamer
Computer Systems Consulting Services
1384 Fontaine
Madison Heights, Michigan 48071

EMail: William.Allen.Simpson@Gmail.com