T/TCP vulnerabilities

Mike Schiffman <mike@infonexus.com> (1998)

Introduction and Impetus

T/TCP is TCP for Transactions. It is a backward compatible extension for TCP to facilitate faster and more efficient client/server transactions. T/TCP is not in wide deployment but it is in use (see appendix A) and it is supported by a handful of OS kernels including: FreeBSD, BSDi, Linux, and SunOS. This short paper will document the T/TCP protocol in light detail, and then cover weaknesses and vulnerabilities.

Background and primer

TCP is a protocol designed for reliability at the expense of expediency (readers unfamiliar with the TCP protocol are directed to the ancient-but-still-relevant: http://www.infonexus.com/~daemon9/Misc/TCPIP-primer.txt). Whenever an application is deemed to require reliability, it is usually built on top of TCP. This lack of speed is considered a necessary evil. Short lived client/server interactions desiring more speed (short in terms of time vs. amount of data flow) are typically built on top of UDP to preserve quick response times. One exception to this rule, of course, is http. The architects of http decided to use the reliable TCP transport for ephemeral connections (indeed a poorly designed protocol).

T/TCP is a small set of extensions to make a faster, more efficient TCP. It is designed to be a completely backward compatible set of extensions to speed up TCP connections. T/TCP achieves its speed increase from two major enhancements over TCP: TAO and TIME_WAIT state truncation. TAO is TCP Accelerated Open, which introduces new extended options to bypass the 3-way handshake entirely. Using TAO, a given T/TCP connection can approximate a UDP connection in terms of speed, while still maintaining the reliability of a TCP connection. In most single data packet exchanges (such is the case with transactional-oriented connections like http) the packet count is reduced by a third.

The second speed up is TIME_WAIT state truncation. TIME_WAIT state truncation allows a T/TCP client to shorten the TIME_WAIT state by up to a factor of 20. This can allow a client to make more efficient use of network socket primitives and system memory.

T/TCP TAO

TCP accelerated open is how T/TCP bypasses the 3-way handshake. Before we discuss TAO, we need to understand why TCP employs a 3-way handshake. According to RFC 793, the principal reason for the exchange is the prevention of old duplicate connection initiations wandering into current connections and causing confusion. With this in mind, in order to obviate the need for the 3-way handshake, there needs to be a mechanism for the receiver of a SYN to guarantee that that SYN is in fact new. This is accomplished with a new extended TCP header option, the connection count (CC).

The CC (referred as tcp_ccgen when on a host) is a simple monotonic variable that a T/TCP host keeps and increments for every TCP connection created on that host. Anytime a client host supporting T/TCP wishes to make a T/TCP connection to a server, it includes (in it's TAO packet) a CC (or CCnew) header option. If the server supports T/TCP, it will cache that client's included CC value and respond with a CCecho option (CC values are cached by T/TCP hosts on a per host basis). If the TAO test succeeds, the 3-way handshake is bypassed, otherwise the hosts fall back to the older process.

The first time a client host supporting T/TCP and a server host supporting T/TCP make a connection no CC state exists for that client on that server. Because of this fact, the 3-way handshake must be done. However, also at that time, the per host CC cache for that client host is initialized, and all subsequent connections can use TAO. The TAO test on the server simply checks to make sure the client's CC is greater then the last received CC from that client. Consider figure 1 below:

```
      Client                               Server
T   -----------------------------------------------------------------------
i   0       --TAO+data--(CC = 2)-->        ClientCC = 1
m   1                                      2 > 1; TAO test succeeds
e   2                                      accept data ---> (to application)
```

[fig 1]

Initially (0) the client sends a TAO encapsulated SYN to the server, with a CC of 2. Since the CC value on the server for this client is 1 (indicating they have had previous T/TCP-based communication) the TAO test succeeds (1). Since the TAO test was successful, the server can pass the data to application layer immediately (2). If the client's CC had not been greater than the server's cached value, the TAO test would have failed and forced the 3-way handshake.

T/TCP TIME_WAIT truncation

Before we can see why it is ok to shorten the TIME_WAIT state, we need to cover exactly what it is and why it exists.

Normally, when a client performs an active close on a TCP connection, it must hold onto state information for twice the maximum segment lifetime (2MSL) which is usually between 60 - 240 seconds (during this time, the socket pair that describes the connection cannot be reused). It is thought that any packet from this connection would be expired (due to IP TTL constraints) from the network. TCP must be consistent with its behavior across all contingencies and the TIME_WAIT state guarantees this consistency during the last phase of connection closedown. It keeps old network segments from wandering into a connection and causing problems and it helps implement the 4-way closedown procedure. For example, if a wandering packet happens to be a retransmission of the servers FIN (presumably due to the clients ACK being lost), the client must be sure to retransmit the final ACK, rather then a RST (which it would do if it had torn down all the state).

T/TCP allows for the truncation of the TIME_WAIT state. If a T/TCP connection only lasts for MSL seconds or less (which is usually the case with transactional-oriented connections) the TIME_WAIT state is truncated to as little as 12 seconds (8 times the retranmission timeout - RTO). This is allowable from a protocol standpoint because of two things: CC number protection against old duplicates and the fact that the 4-way closedown procedure packet loss scenario (see above) can be handled by waiting for the RTO (multiplied by a constant) as opposed to waiting for a whole 2MSL.

As long as the connection didn't last any longer then MSL, the CC number in the next connection will prevent old packets with an older CC number from being accepted. This will protect connections from old wandering packets (if the connection did last longer, it is possible for the CC values to wrap and potentially be erroneously delivered to a new incarnation of a connection).

Dominance of TAO

It is easy for an attacker to ensure the success or failure of the TAO test. There are two methods. The first relies on the second oldest hacking tool in the book. The second is more of a brutish technique, but is just as effective.

Packet Sniffing

If we are on the local network with one of the hosts, we can snoop the current CC value in use for a particular connection. Since the tcp_ccgen is incremented monotonically we can precisely spoof the next expected value by incrementing the snooped number. Not only will this ensure the success of our TAO test, but it will ensure the failure of the next TAO test for the client we are spoofing.

The Numbers Game

The other method of TAO dominance is a bit rougher, but works almost as well. The CC is an unsigned 32-bit number (ranging in value from 0 - 4,294,967,295). Under all observed implementations, the tcp_ccgen is initialized to 1. If an attacker needs to ensure the success of a TAO connection, but is not in a position where s/he can sniff on a local network, they should simply choose a large value for the spoofed CC. The chances that one given T/TCP host will burn through even half the tcp_ccgen space with another given host is highly unlikely. Simple statistics tells us that the larger the chosen tcp_ccgen is, the greater the odds that the TAO test will succeed. When in doubt, aim high.

T/TCP and SYN flooding

TCP SYN flooding hasn't changed much under TCP for Transactions. The actual attack is the same; a series of TCP SYNs spoofed from unreachable IP addresses. However, there are 2 major considerations to keep in mind when the target host supports T/TCP:

1) SYN cookie invalidation: A host supporting T/TCP cannot, at the same time, implement SYN cookies. TCP SYN cookies are a SYN flood defense technique that works by sending a secure cookie as the sequence number in the second packet of the 3-way handshake, then discarding all state for that connection. Any TCP options sent would be lost. If the final ACK comes in, only then will the host create the kernel socket data structures. TAO obviously cannot be used with SYN cookies.

2) Failed TAO processing result in queued data: If the TAO test fails, any data included with that packet will be queued pending the completion of the connection processing (the 3-way handshake). During a SYN flood, this can make the attack more severe as memory buffers fill up holding this data. In this case, the attacker would want to ensure the failure of the TAO test for each spoofed packet.

In a previous Phrack Magazine article, the author erroneously reported that T/TCP would help to alleviate SYN flood vulnerability. This obviously incorrect statement was made before copious T/TCP research was done and is hereby rescinded. My bad.

T/TCP and trust relationships

An old attack with a new twist. The attack paradigm is still the same, (readers unfamiliar with trust relationship exploitation are directed to P48-14) this time, however, it is easier to wage. Under T/TCP, there is no need to attempt to predict TCP sequence numbers. Previously, this attack required the attacker to predict the return sequence number in order to complete the connection establishment processing and move the connection into the established state. With T/TCP, a packet's data will be accepted by the application as soon as the TAO test succeeds. All the attacker needs to do is ensure that the TAO test will succeed. Consider the figure below.

```
     Attacker                     Server                          Trusted
    -----------------------------------------------------------------------
    0         -spoofed-TAO->
    1                        TAO test succeeds
T   2                        data to application
i   3                                          ---TAO-response->
m   4                                                      no open socket
e   5                                          <------RST-------
    6                        tears down connection

                            [ fig 2 ]
```

The attacker first sends a spoofed connection request TAO packet to the server. The data portion of this packet presumably contains the tried and true non-interactive backdooring command `echo + + > .rhosts`. At (1) the TAO test succeeds and the data is accepted (2) and passed to application (where it is processed). The server then sends its T/TCP response to the trusted host (3). The trusted host, of course, has no open socket (4) for this connection, and responds with the expected RST segment (5). This RST will teardown the attacker's spoofed connection (6) on the server. If everything went according to plan, and the process executing the command in question didn't take too long to run, the attacker may now log directly into the server.

To deal with (5) the attacker can, of course, wage some sort of denial of service attack on the trusted host to keep it from responding to the unwarranted connection.

T/TCP and duplicate message delivery

Ignoring all the other weaknesses of the protocol, there is one major flaw that causes the T/TCP to degrade and behave decidedly NONTCP-like, therefore breaking the protocol entirely. The problem is within the TAO mechanism. Certain conditions can cause T/TCP to deliver duplicate data to the application layer. Consider the timeline in figure 3 below:

```
      Client                                Server
    ----------------------------------------------------------------------
    0                  --TAO-(data)--->
    1                                       TAO test succeeds
T   2                                       accept data ---> (to application)
i   3                                       *crash*     (reboot)
m   4  timeout (resends) --TAO-(data)--->
e   5                                       TAO test fails (data is queued)
    6  established       <-SYN-ACK(SYN)--  fallback to 3WHS
    7                    --ACK(SYN)----->  established (data --> application)
```

[fig 3]

At time 0 the client sends its TAO encapsulated data to the server (for this example, consider that both hosts have had recent communication, and the server has defined CC values for the client). The TAO test succeeds (1) and the server passes the data to the application layer for processing (2). Before the server can send its response however (presumably an ACK) it crashes (3). The client receives no acknowledgement from the server, so it times out and resends its packet (4). After the server reboots it receives this retransmission, this time, however, the TAO test fails and the server queues the data (5). The TAO test failed and forced a 3-way handshake (6) because the servers CC cache was invalidated when it rebooted. After completing the 3-way handshake and establishing a connection, the server then passes the queued data to the application layer, for a second time. The server cannot tell that it has already accepted this data because it maintains no state after a reboot. This violates the basic premise of T/TCP that it must remain completely backward compatible with TCP.

In closing

T/TCP is a good idea that just wasn't implemented properly. TCP was not designed to support a connectionless-like paradigm while still maintaining reliability and security (TCP wasn't even designed with security in mind at all). T/TCP brings out too many problems and discrete bugs in TCP to be anything more then a novelty.

Appendix A: Internet hosts supporting RFC 1644

This information is ganked from Richard Steven's T/TCP homepage (http://www.kohala.com/~rstevens/ttcp.html). It is not verfied to be correct.

- www.ansp.br
- www.elite.net
- www.iqm.unicamp.br
- www.neosoft.com
- www.sbq.org.br
- www.uidaho.edu
- www.yahoo.com

Appendix B: Bibliography

1) Braden, R. T. 1994 "T/TCP - TCP Extensions for Transactions...", 38 p
2) Braden, R. T. 1992 "Extending TCP for Transactions - Concepts...", 38 p
3) Stevens, W. Richard. 1996 "TCP Illustrated volume III", 328 p
4) Smith, Mark. 1996, "Formal verification of Communication...", 15 p

EOF