Internet Vulnerabilities Related to TCP/IP and T/TCP

Marco de Vivo mdevivo@reacciun.ve Apartado Postal 68274 Caracas, Venezuela.

Gabriela O. de Vivo gdevivo@reacciun.ve

Roberto Koeneke rkoeneke@compuserve.com

Germinal Isern isern@reacciun.ve

LACORE U.C.V.

Abstract*

The Internet put the rest of the world at the reach of our computers. In the same way it also made our computers reachable by the rest of the world. Good news and bad news! Over the last decade, the Internet has been subject to widespread security attacks. Besides the classical terms, new ones had to be found in order to designate a large collection of threats: *Worms*, *break-ins*, *hackers*, *crackers*, *hijacking*, *phrackers*, *spoofing*, *man-in-the-middle*, *password-sniffing*, *denial-of-service*, and so on.

Since the Internet was born of academic efforts to share information, it never strove for high security measures. In fact in some of its components, security was consciously traded for easiness in sharing. Although the advent of *electronic commerce* has pushed for "real security" in the Internet, there are still a large number of users (including computer scientists) that are very vulnerable to attacks, mostly because they are not aware of the nature (and ease) of the attacks and still believe that a "good" password is all they need to be concerned about.

Aiming for a better understanding of the subject, we wrote a first paper **[1]** in which we discussed several threats and attacks related to **TCP/IP**. The present work is an extension of the first one, and its main goal is to include **T/TCP** in the discussion. Additionally, in an effort to make this paper more comprehensive, we included some sections from the former.

Besides the description of each attack (the *what*), we also discuss the way they are carried out (the *how*) and, when possible, the related means of prevention, detection and/or defense.

Keywords: TCP/IP, T/TCP, SYN Attack, Sniffing, Spoofing, Denial of Service.

* This work was partially supported by C.D.C.H (U.C.V) under Grant No. 03-13-4051-97

1. The TCP/IP Scenario.

1.1. IP Spoofing.

The term *trusted host* was coined by the developers of the UNIX networking software. If one host extends trust to another host, then any user who has the same *username* (login name) on both machines can log in (or execute *remote* commands) from the trusted host to the trusting one **without presenting a password**. Trust can also be extended to different users from selected hosts, and eventually to **any** user from **any** host! The trusted usernames and hostnames are maintained in two types of special files (UNIX): ***.rhosts*** and ***hosts.equiv***. Any authorized user of a host **A**, can create in **her own** directory (on **A**) a file named .rhosts containing the combinations of username and hostname that may connect to **its** account on **A**. By contrast, only **one** hosts.equiv file (/etc/hosts.equiv) may exist on each host. If present, it contains a list of trusted hosts. Any user of any host in the list may access an account with the same username on the trusting host without presenting a password.

It is worth noting that besides ***rlogin*** (the login command from a **remote** host), several other commands use the *trusted host* scheme (e.g., ***rcp***, ***rdist***, ***rsh***.) They are known collectively as ***r**** commands.

Although *trusted host* is a very useful and convenient scheme, its actual implementation is quite vulnerable to attacks because an authentication mechanism based solely on IP addresses is used. In fact, when any of the r* commands is invoked from a remote machine, the receiving host checks if the IP address of the sender matches an authorized (trusted) host. If so, the command is executed. Otherwise, either permission for execution is denied or a **password** (and a **login** if the originating user is not equivalent to the remote user) is (are) prompted for on the remote machine.

IP Spoofing attacks exploit this weak form of authentication. In this type of attack, an intruder masquerades her host **too.evil.com** as **friend.ok.com**, a machine (usually internal) trusted by the host **target.ok.com**. The intruder does this by substituting the IP address of the trusted machine, **friend.ok.com**, for the IP address of her host, **too.evil.com**, in all of the outgoing packets.

The machine being attacked, **target.ok.com**, then believes that the intruder is, in fact, the machine that it trusts, **friend.ok.com**, and gives it access.
Please note that the above is just a generic explanation. IP spoofing attacks can be mounted in several ways on diverse scenarios and each case must usually be addressed accordingly.
Thorough discussions on IP spoofing can be found in **[1]** and **[2]**. For a complete and detailed guide to TCP/IP, please consult **[3]**.

1.2. TCP Connection Spoofing Blind Attacks.

A ***TCP connection spoofing attack*** is a very complex (*IP spoofing*-based) 'blind' attack. The possibility of an attack of this type, was first discussed by R. Morris **[4]**. The following scenario outlines a typical TCP connection spoofing attack:
<u>The Hosts</u>: **Target** is a server trusting the host **Friend**, **Evil** is the attacker's machine and **Unreach** is an unreachable host.
<u>The Events</u>: **Evil** (impersonating **Friend**) starts a TCP connection with **Target** which, in turn, sends a reply (related to the connection establishment protocol) to the <u>real</u> **Friend** because neither DNS spoofing nor source routing is being used.
At this point, **Evil** is facing two problems:
1) It doesn't know what the answer from **Target** was, so it cannot be sure about the exact contents of the message that must be sent to **Target** to continue with the connection establishment protocol.
2) It must block the delivery (to **Friend**) of messages being sent by **Target**, otherwise these unexpected messages would induce **Friend** to ask **Target** to abort the connection being established (which would frustrate the attack).
Since the attacker has been gathering enough statistics as to predict what should be answered to **Target** to continue with the connection establishment, a new message containing a guessed reply is sent from **Evil** (impersonating **Friend**) to **Target**. If the attacker is correct in her prediction, the connection is established, and **Target** is compromised. Generally, after compromise, the attacker will use r* commands to insert a *backdoor*.
Meanwhile, to deal with the second problem, **Evil** (this time impersonating **Unreach**) has been flooding **Friend** with TCP connection requests directed to the TCP port it desires disabled. Since the reply messages that are being sent from **Friend** to **Unreach** will never be acknowledged, **Friend**'s queue of incomplete connections will keep increasing until a limit is reached after which all incoming packets related to connection establishment are silently discarded by TCP (including the ones coming from **Target**).
Now, to better understand the above scenario, some related details will be explained:

• *What is attacked?*: The attack is usually directed either to port TCP 513 (*rlogin*) or to port TCP 514 (*rsh*). Often, the command 'echo "+ +" >> ~/.rhosts' (used in UNIX to extend trust to *any* user from *any* host) is executed to install a backdoor.

• *Why is it called a 'blind' attack?*: Because **Evil**'s TCP software never 'sees' any message from **Target**'s TCP (which is sending to **Friend** all the datagrams related to the fake connection). Hence **Evil** must rely exclusively on guessing.

• *What is guessed?*: TCP is a connection-oriented, reliable transport protocol. Connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data. Reliability is provided in TCP by the use of checksums, timers, data sequencing and acknowledgments. By assigning a sequence number to **every** byte transferred, and requiring an acknowledgment from the other end upon receipt, TCP can guarantee reliable delivery. Sequence numbers are used to ensure proper ordering of the data and to eliminate duplicate data bytes. Note that in a TCP session there are usually two streams of data (every end point is receiving from one of them and sending through the other). So an ***ISN*** (initial sequence number) must be assigned to <u>each</u> stream when the connection is being established. To see how it is done, let's suppose that **C** is a client wishing to connect to the server **S**, and analyze the connection establishment process (often called the *three-way handshake*):

1	**C**	---SYN XX--->	**S**
2	**C**	<---SYN YY/ ACK XX+1---	**S**
3	**C**	---ACK YY+1--->	**S**

1. C sends a TCP message (known as SYN request) to **S** with the special flag SYN (SYNchronize sequence numbers) set to ON. A SYN request specifies the port number of the server that the client wants to connect to, and the client's ISN (XX in this example).
2. The server responds with its own SYN message containing **S**'s ISN (YY) and acknowledging **C**'s SYN by specifying that the **next** byte expected from **C** is the byte numbered XX+1.
3. C acknowledges the SYN message from **S**, and data transfer may take place.

The problem with the blind attack outlined in the scenario is that **Evil** never sees the second message, which is in fact sent to **Friend**. So what is to be guessed by the attacker is YY (in order to have Evil sending the ACK YY+1 message). However, this is not an easy task because each TCP maintains a 32-bit ISN counter that is incremented by 64,000 every half-second, and, additionally, by 64,000 each time a connection is established.
To get an idea of where in the 32-bit sequence number space **Target**'s TCP is, the attacker establishes several connections to a TCP port on **Target**, and stores the final ISN received. Besides, the attacker calculates the average RTT (round-trip time) from **Evil** to **Target** to **Evil** (most likely by using ICMP Ping messages). Now the attacker has the baseline (the last received ISN) and a good idea of how long it will take an IP datagram to travel across the Internet to reach **Target** (approximately half the average RTT, as most times the routes are symmetrical). So, the attacker immediately proceeds to initiate the three-way handshake (each interim connection would increment the ISN by 64,000) and finally the spoofed segment with the predicted ACK is sent to **Target**. As said before, if the guess is correct, **Target** is compromised.

• *What must be disabled on* ***Friend****?*: In the SYN request sent by **Evil** (impersonating **Friend**) to **Target**, not only the destination port number is specified, but also the source port number is included (as in every TCP segment). So when **Target** answers by sending to **Fiend** the second segment in the three-way handshake protocol (SYN/ACK) it will use as destination port number, the same one he received as source port number in the SYN request. Therefore, what must be disabled on **Friend** is precisely this port. In the outlined scenario a ***Denial of Service*** attack known as ***TCP SYN flooding*** attack (or just SYN attack) is used to disable the port. *SYN attacks* are discussed in section **1.3.**
• *How can this attack be prevented?*: This attack can be prevented by disabling all the r* commands, or (if trust is extended only to local hosts) by having the router(s) deny any packet coming from outside with a source IP address corresponding to a local host. *RARP* queries could be used to detect attacks coming from machines on the same physical network as the target server. Finally, sequence number attacks can be prevented by using TCP implementations that randomize the ISN of each connection as well as the value of the increment that is added every half-second (e.g., BSD/OS 2.0 and 4.4BSD-Lite2 implementations).
An exhaustive discussion on TCP connection spoofing attacks can be found in **[5]**.

1.3. Clogging.

The implementation of the three-way handshake protocol used by TCP to establish connections can be abused to mount a very nasty denial of service attack known as ***TCP SYN flooding*** attack (or just SYN attack). The potential for abuse arises at the point where the server system has sent an acknowledgment (SYN/ACK) back to client but has not yet received the ACK message. This is what is known as a *pending* connection or (somewhat improperly) a *half-open* connection. The server has built in its system memory a queue containing all pending connections. This queue is of finite size, and it can be made to overflow by intentionally creating too many partially open connections. This can be accomplished by flooding a server port with spoofed SYN messages. As the packets are spoofed to impersonate unreachable clients, the protocol will never be completed, and thus the half-open queue will eventually fill causing the server to be unable to accept any new incoming connections.
Although the half-open connections will eventually expire, the attacking system can just keep sending spoofed SYN requirements faster than the victim system can expire the pending connections.
TCP SYN flooding has been used to mount major attacks on Internet Service Providers. The services themselves are not harmed, just the ability to provide them is impaired.
Blocking an ongoing attack by having a router deny any packet coming from a specific IP address is not a solution because usually each of the source addresses in the spoofed packets is randomly chosen, by the attacker's software, from an array of unreachable IP addresses.
Several patches have been released to make UNIX kernels SYN-attack resistant. If a patch is not available, and your UNIX kernel is configurable at run time (e.g., SOLARIS using the *ndd* utility, and BSDI using *sysctl*), then, at least, the number of half-open connections allowed should be increased and the amount of time that a connection is allowed to stay in a half-open state should be reduced.
Additional details on TCP SYN flooding can be found in **[6]**.

2. The T/TCP Scenario.

2.1. Background.

Unlike UDP, TCP was designed to provide a reliable transport mechanism between interconnected computers. As this goal was fully achieved, TCP became the *de facto* standard over which the Internet and most of

the actual **intranets** and **extranets** rest.
Most of the popularity of TCP is due to the fact that it showed itself as an excellent support to build applications following the *client-server* paradigm. The whole Internet grew following this paradigm and nowadays more and more modern applications are being designed based not only on the classic client-server model but on the **client-server transaction** model as well. Please note that in this context, the term *transaction* just means the following sequence:

- A client sends a *request* to a server.
- The server sends back a *reply*.

So, **no** actions associated with database transactions are implied (*two-phase commit*, *locking*, and so on).
Now, in spite of its reliability (usually a necessary condition for transactions) the problem with using TCP for transactions is that the normal sequence it follows to establish and terminate connections, imposes an excessive number of network packets (as well as some intolerable delay constraints) to the simple request-reply sequence. Several protocols (mostly modifications or extensions to TCP/IP) **[7]** have been proposed to address this problem: **IRTP**, **VMTP** and **T/TCP**.
T/TCP [8], *TCP for transactions*, a backward compatible extension to TCP, has been implemented and ported into several UNIX systems, and so far it seems the best suited protocol to handle client-server transactions. T/TCP was designed to be both: reliable as TCP and fast as UDP. To achieve these goals, the designers had to overcome two major problems related to the use of TCP for client-server transactions:

- The three-way handshake protocol, used to establish connections, adds an extra **RTT** (round-trip time) to the client's measured transaction time.
- Since the client sends the first **FIN** (i.e., does the *active close*), its end of the connection must remain in the **TIME_WAIT** state for **240** seconds after receiving the server's **FIN**. Since during this time, the connection's pair of sockets cannot be reused (with some exceptions), only **64,512** *ephemeral* ports are available to the client every 240 seconds, a very demanding client application (more than 268 transactions per second) would quickly run out of ports.

T/TCP solves these two problems by bypassing the three-way handshake, and by shortening the duration of the TIME_WAIT state. This is achieved by extending TCP as follows:

1) Three new TCP options are included to be used in T/TCP headers: **CC**, **CCnew**, and **CCecho**. Observe that T/TCP headers differ from regular TCP headers only by the possible values of the *OPTIONS* field.
Besides, as T/TCP requires some new information to be maintained by the kernel, several new variables are added: a global kernel variable, along with three variables in a per-host (for each contacted host) cache, and three control variables for each open connection.
These new options and variables are used basically to maintain on each host a ***Connection Count*** (**CC**) that is incremented by one for **every** connection established by the host (actively or passively). So, every time a client establishes a valid T/TCP connection with a server, the server *caches* the CC sent by the client in its first (SYN) segment. Later, when a new connection is being initiated by the same client, the server compares the cached client's CC with the new one contained in the SYN segment just received from the client. **If** the latter is greater than the former, then the connection is considered **ESTABLISHED** and all the data contained in the same client's first segment is <u>immediately</u> passed to the server process. **Else**, the server just continues the normal three-way handshake and sends the second segment.
The new form of connection establishment is known as ***TAO***, *TCP accelerated open*. The associated test is called the ***TAO test***.

2) The use of **CC**s, not only eases the task of dealing with old duplicated segments but allows a new way to handle the problem of *lost* final **ACKS** as well. So for short connection times (less than the **MSL**, i.e., less than 120 seconds) T/TCP allows the TIME_WAIT state to be truncated, shortening its duration from 240 seconds to 12 seconds (or to 8 times the retransmission timeout, **RTO**). Further optimization can be achieved by making small changes to the client process. Note that shorter times mean not only a continual availability of ports, but also a faster release of resources.

When the client and the server both have state information about each other, the minimal T/TCP transaction of three segments occurs. The segments are exchanged as follows:
i. From <u>client to server</u> (*active open*): The **first** segment, containing: client's SYN, client's data (*request*), client's FIN, and client's CC. When this segment is received by the server, a Tao test is performed. If the test succeeds, the data is processed and passed to the server process.
ii. From <u>server to client</u>: The **second** segment, containing: server's SYN, server's data (*reply*), server's FIN, server's cumulative ACK (acknowledging client's SYN, data, and FIN), server's CC, the CCecho option echoing the client's CC and informing the client that the server understands T/TCP. As soon as the receiving client processes the ACK to its FIN, the server's *reply* is passed to the client process.

iii. From client to server: The **third** segment, which includes the client's cumulative ACK. When the server processes the ACK to its FIN, the connection is moved to the **CLOSED** state.
The book referenced in **[7]** contains an excellent and thorough description of T/TCP.

2.2. T/TCP and Spoofing Attacks.

T/TCP connection spoofing attacks are not only easier to mount than in TCP, but also their chance to succeed is much higher. The attacker has two ways to deceive a target server by impersonating a known host:
i. By using packet *sniffing* to obtain the current CC value in use for a given connection, or
ii. By choosing a **large** value for the **spoofed** CC, hoping (a reasonable assumption) that the value cached for the legitimate host is still quite far from 4,294,967,295 the maximum possible value (CC is an unsigned 32-bit number).
Whichever the method, the goal is the same: to succeed the TAO test. It is worth noting that if the attacker indeed succeeds, chances are high that the next TAO test for the legitimate client will fail.
Since a successful TAO test is a sufficient condition for a server to consider a T/TCP connection *established*, intruders can easily exploit UNIX *trust relationships* to compromise hosts (no more prediction of TCP sequence numbers is needed). Sending a spoofed SYN segment with an acceptable CC is basically the only action needed to mount the attack. Additionally, as the attack is usually directed either to port TCP 513 (*rlogin*) or to port TCP 514 (*rsh*), the *backdooring* command 'echo + + >> ~/.rhosts' (used in UNIX to extend trust to *any* user from *any* host) is most probably included in the data portion of the spoofed segment. Remember that if the TAO test finally succeeds, the data portion (including the *echo* command) is passed with no delay to the server process. Nevertheless, there is still a potential weakness in the attack: when the server sends the **second** segment (unexpectedly) to the **legitimate** client, the latter responds immediately with a RST segment which of course tears the spoofed connection down. Usually the backdooring command is executed long before the teardown, however in those (rare) situations in which the server process takes too long to execute the command, the intruder could still mount some sort of Denial of Service attack on the trusted (legitimate) host to keep it from responding to the unwanted connection.
Preventing these kind of attacks under T/TCP is even harder that in the case of TCP. TAO mechanism, as seen above, allows intruders to establish connections without the need of TCP sequence numbers prediction. This fact, besides strongly easing the attack, makes absolutely worthless the best available technique to prevent TCP connection spoofing attacks: the randomization of the ISN of each connection and of the increment that is added every half-second.
The remaining defenses are the same as in TCP: disabling all the r* commands, or (if trust is extended only to local hosts) having the router(s) deny any packet coming from outside with a source IP address corresponding to a local host.

2.3. T/TCP and SYN Flooding.

Under T/TCP the SYN attack is carried out in exactly the same way as under TCP: a flood of SYN segments spoofed from unreachable IP addresses. Two new circumstances, however, make the attack even nastier:
i. When a TAO test fails, the data (the *request*) carried by the first (SYN) segment must be queued pending the completion of a regular three-way handshake. An attacker could then drastically increase the effect of a SYN flooding attack by ensuring the failure of the TAO test for each spoofed segment. In this way, buffers will be exhausted very quickly.
ii. Promising SYN flooding defense techniques, like ***TCP SYN* cookies [9]**, designed to work with the regular three-way handshake, cannot be used with TAO.

References.

[1] M. de Vivo, G. de Vivo, G. Isern, **Internet Security Attacks at the Basic Levels**. Operating Systems Review, Vol. 32, No. 2. SIGOPS, ACM, April **1998**.

[2] D. Atkins *et al*., **Internet Security**. New Riders, Second Edition, **1997**, pp. 173-232.

[3] W. Stevens, **TCP/IP Illustrated, Vol. 1**. Addison-Wesley, **1994**.

[4] R. Morris, **A Weakness in the 4.2 BSD UNIX TCP/IP Software**. Computing Science Technical Report 117. AT&T Bell Laboratories, **1985**.

[5] **IP-spoofing Demystified**. By route@infonexus.com Guild Productions, **June 1996**.

[6] **TCP SYN Flooding and IP Spoofing Attacks**. CERT Advisory CA-96.21. Available at http://www.cert.org and at ftp://info.cert.org/pub/cert_advisories/ **1996**.

[7] W.R. Stevens, **TCP/IP Illustrated, Vol. 3**. Addison-Wesley, **1996**, pp. 24-25.

[8] R.T. Braden, **T/TCP--TCP Extensions for Transactions**. RFC 1644, **1994**.

[9] **syncookies**, ftp://koobera.math.uic.edu/ at pub/docs/syncookies-archive.