Security Problems in the TCP/IP Protocol Suite

*S.M. Bellovin**
smb@ulysses.att.com

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The TCP/IP protocol suite, which is very widely used today, was developed under the sponsorship of the Department of Defense. Despite that, there are a number of serious security flaws inherent in the protocols, regardless of the correctness of any implementations. We describe a variety of attacks based on these flaws, including sequence number spoofing, routing attacks, source address spoofing, and authentication attacks. We also present defenses against these attacks, and conclude with a discussion of broad-spectrum defenses such as encryption.

1. INTRODUCTION

The TCP/IP protocol suite[1][2], which is very widely used today, was developed under the sponsorship of the Department of Defense. Despite that, there are a number of serious security flaws inherent in the protocols. Some of these flaws exist because hosts rely on IP source address for authentication; the Berkeley "*r*-utilities"[3] are a notable example. Others exist because network control mechanisms, and in particular routing protocols, have minimal or non-existent authentication.

When describing such attacks, our basic assumption is that the attacker has more or less complete control over some machine connected to the Internet. This may be due to flaws in that machine's own protection mechanisms, or it may be because that machine is a microcomputer, and inherently unprotected. Indeed, the attacker may even be a rogue system administrator.

1.1 Exclusions

We are not concerned with flaws in particular implementations of the protocols, such as those used by the Internet "worm"[4][5][6]. Rather, we discuss generic problems with the protocols themselves. As will be seen, careful implementation techniques can alleviate or prevent some of these problems. Some of the protocols we discuss are derived from Berkeley's version of the UNIX→ system; others are generic Internet protocols.

We are also not concerned with classic network attacks, such as physical eavesdropping, or altered or injected messages. We discuss such problems only in so far as they are facilitated or possible because of protocol problems.

For the most part, there is no discussion here of vendor-specific protocols. We do discuss some problems with Berkeley's protocols, since these have become de facto standards for many vendors, and not just for UNIX systems.

2. TCP SEQUENCE NUMBER PREDICTION

One of the more fascinating security holes was first described by Morris[7]. Briefly, he used TCP sequence number prediction to construct a TCP packet sequence without ever receiving any responses from the server. This allowed him to spoof a trusted host on a local network.

---

* Author's address: Room 3C-536B AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974.

Reprinted from Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989.

The normal TCP connection establishment sequence involves a 3-way handshake. The client selects and transmits an initial sequence number ISN_C, the server acknowledges it and sends its own sequence number ISN_S, and the client acknowledges that. Following those three messages, data transmission may take place. The exchange may be shown schematically as follows:

$C \rightarrow S: SYN(ISN_C)$
$S \rightarrow C: SYN(ISN_S), ACK(ISN_C)$
$C \rightarrow S: ACK(ISN_S)$
$C \rightarrow S: data$
and/or
$S \rightarrow C: data$

That is, for a conversation to take place, C must first hear ISN_S, a more or less random number.

Suppose, though, that there was a way for an intruder X to *predict* ISN_S. In that case, it could send the following sequence to impersonate trusted host T:

$X \rightarrow S: SYN(ISN_X), SRC = T$
$S \rightarrow T: SYN(ISN_S), ACK(ISN_X)$
$X \rightarrow S: ACK(ISN_S), SRC = T$
$X \rightarrow S: ACK(ISN_S), SRC = T, nasty - data$

Even though the message $S \rightarrow T$ does not go to X, X was able to know its contents, and hence could send data. If X were to perform this attack on a connection that allows command execution (i.e., the Berkeley *rsh* server), malicious commands could be executed.

How, then, to predict the random *ISN*? In Berkeley systems, the initial sequence number variable is incremented by a constant amount once per second, and by half that amount each time a connection is initiated. Thus, if one initiates a legitimate connection and observes the ISN_S used, one can calculate, with a high degree of confidence, ISN_S' used on the next connection attempt.

Morris points out that the reply message

$S \rightarrow T: SYN(ISN_S), ACK(ISN_X)$

does not in fact vanish down a black hole; rather, the real host T will receive it and attempt to reset the connection. This is not a serious obstacle. Morris found that by impersonating a server port on T, and by flooding that port with apparent connection requests, he could generate queue overflows that would make it likely that the $S \rightarrow T$ message would be lost. Alternatively, one could wait until T was down for routine maintenance or a reboot.

A variant on this TCP sequence number attack, not described by Morris, exploits the *netstat*[8] service. In this attack, the intruder impersonates a host that is down. If *netstat* is available on the target host, it may supply the necessary sequence number information on another port; this eliminates all need to guess[1].

Defenses

Obviously, the key to this attack is the relatively coarse rate of change of the initial sequence number variable on Berkeley systems. The TCP specification requires that this variable be incremented approximately 250,000 times per second; Berkeley is using a much slower rate. However, the critical factor is the granularity, not the average rate. The change from an increment of 128 per second in 4.2BSD to 125,000 per second in 4.3BSD is meaningless, even though the latter is within a factor of two of the specified rate.

---

1. The *netstat* protocol is obsolete, but is still present on some Internet hosts. Security concerns were not behind its elimination.

Let us consider whether a counter that operated at a true 250,000 hz rate would help. For simplicity's sake, we will ignore the problem of other connections occurring, and only consider the fixed rate of change of this counter.

To learn a current sequence number, one must send a SYN packet, and receive a response, as follows:

$$\begin{aligned} X \rightarrow S&: \ SYN(ISN_X) \\ S \rightarrow X&: \ SYN(ISN_S), ACK(ISN_X) \end{aligned} \tag{1}$$

The first spoof packet, which triggers generation of the next sequence number, can immediately follow the server's response to the probe packet:

$$X \rightarrow S: \ SYN(ISN_X), SRC = T \tag{2}$$

The sequence number ISN_S used in the response

$$S \rightarrow T: \ SYN(ISN_S), ACK(ISN_X)$$

is uniquely determined by the time between the origination of message (1) and the receipt at the server of message (1). But this number is precisely the round-trip time between X and S. Thus, if the spoofer can accurately measure (and predict) that time, even a 4 ∝-second clock will not defeat this attack.

How accurately can the trip time be measured? If we assume that stability is good, we can probably bound it within 10 milliseconds or so. Clearly, the Internet does not exhibit such stability over the long-term[9], but it is often good enough over the short term.[2] There is thus an uncertainty of 2500 in the possible value for ISN_S. If each trial takes 5 seconds, to allow time to re-measure the round-trip time, an intruder would have a reasonable likelihood of succeeding in 7500 seconds, and a near-certainty within a day. More predictable (i.e., higher quality) networks, or more accurate measurements, would improve the odds even further in the intruder's favor. Clearly, simply following the letter of the TCP specification is not good enough.

We have thus far tacitly assumed that no processing takes places on the target host. In fact, some processing does take place when a new request comes in; the amount of variability in this processing is critical. On a 6 MIPS machine, one tick — 4 ∝-seconds — is about 25 instructions. There is thus considerable sensitivity to the exact instruction path followed. High-priority interrupts, or a slightly different TCB allocation sequence, will have a comparatively large effect on the actual value of the next sequence number. This randomizing effect is of considerable advantage to the target. It should be noted, though, that faster machines are *more* vulnerable to this attack, since the variability of the instruction path will take less real time, and hence affect the increment less. And of course, CPU speeds are increasing rapidly.

This suggests another solution to sequence number attacks: randomizing the increment. Care must be taken to use sufficient bits; if, say, only the low-order 8 bits were picked randomly, and the granularity of the increment was coarse, the intruder's work factor is only multiplied by 256. A combination of a fine-granularity increment and a small random number generator, or just a 32-bit generator, is better. Note, though, that many pseudo-random number generators are easily invertible[10]. In fact, given that most such generators work via feedback of their output, the enemy could simply compute the next ''random'' number to be picked. Some hybrid techniques have promise — using a 32-bit generator, for example, but only emitting 16 bits of it — but brute-force attacks could succeed at determining the seed. One would need at least 16 bits of random data in each increment, and perhaps more, to defeat probes from the network, but that might leave too few bits to guard against a search for the seed. More research or simulations are needed to determine the proper parameters.

---

2. At the moment, the Internet may not have such stability even over the short-term, especially on long-haul connections. It is not comforting to know that the security of a network relies on its low quality of service.