Network Working Group
Request for Comments: 1948
Category: Informational

S. Bellovin
AT&T Research

May 1996

# Defending Against Sequence Number Attacks

## Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

## Abstract

IP spoofing attacks based on sequence number spoofing have become a serious threat on the Internet (CERT Advisory CA-95:01). While ubiquitous crypgraphic authentication is the right answer, we propose a simple modification to TCP implementations that should be a very substantial block to the current wave of attacks.

## Overview and Rational

In 1985, Morris [1] described a form of attack based on guessing what sequence numbers TCP [2] will use for new connections. Briefly, the attacker gags a host trusted by the target, impersonates the IP address of the trusted host when talking to the target, and completes the 3-way handshake based on its guess at the next initial sequence number to be used. An ordinary connection to the target is used to gather sequence number state information. This entire sequence, coupled with address-based authentication, allows the attacker to execute commands on the target host.

Clearly, the proper solution is cryptographic authentication [3,4]. But it will quite a long time before that is deployed. It has therefore been necessary for many sites to restrict use of protocols that rely on address-based authentication, such as rlogin and rsh. Unfortunately, the prevalence of "sniffer attacks" -- network eavesdropping (CERT Advisory CA-94:01) -- has rendered ordinary TELNET [5] very dangerous as well. The Internet is thus left without a safe, secure mechanism for remote login.

We propose a simple change to TCP implementations that will block most sequence number guessing attacks. More precisely, such attacks will remain possible if and only if the Bad Guy already has the ability to launch even more devastating attacks.

## Details of the Attack

In order to understand the particular case of sequence number guessing, one must look at the 3-way handshake used in the TCP open sequence [2]. Suppose client machine A wants to talk to rsh server B. It sends the following message:

```
A->B: SYN, ISNa
```

That is, it sends a packet with the SYN ("synchronize sequence number") bit set and an initial sequence number ISNa.

B replies with

```
B->A: SYN, ISNb, ACK(ISNa)
```

In addition to sending its own initial sequence number, it acknowledges A's. Note that the actual numeric value ISNa must appear in the message.

A concludes the handshake by sending

```
A->B: ACK(ISNb)
```

The initial sequence numbers are intended to be more or less random. More precisely, RFC 793 specifies that the 32-bit counter be incremented by 1 in the low-order position about every 4 microseconds. Instead, Berkeley-derived kernels increment it by a constant every second, and by another constant for each new connection. Thus, if you open a connection to a machine, you know to a very high degree of confidence what sequence number it will use for its next connection. And therein lies the attack.

The attacker X first opens a real connection to its target B -- say, to the mail port or the TCP echo port. This gives ISNb. It then impersonates A and sends

```
Ax->B: SYN, ISNx
```

where "Ax" denotes a packet sent by X pretending to be A.

B's response to X's original SYN (so to speak)

```
B->A: SYN, ISNb', ACK(ISNx)
```

goes to the legitimate A, about which more anon. X never sees that message but can still send

```
Ax->B: ACK(ISNb')
```

using the predicted value for ISNb'. If the guess is right -- and usually it will be -- B's rsh server thinks it has a legitimate connection with A, when in fact X is sending the packets. X can't see the output from this session, but it can execute commands as more or less any user -- and in that case, the game is over and X has won.

There is a minor difficulty here. If A sees B's message, it will realize that B is acknowledging something it never sent, and will send a RST packet in response to tear down the connection. There are a variety of ways to prevent this; the easiest is to wait until the real A is down (possibly as a result of enemy action, of course). In actual practice, X can gag A by exploiting a very common implementation bug; this is described below.

## The Fix

The choice of initial sequence numbers for a connection is not random. Rather, it must be chosen so as to minimize the probability of old stale packets being accepted by new incarnations of the same connection [6, Appendix A]. Furthermore, implementations of TCP derived from 4.2BSD contain special code to deal with such reincarnations when the server end of the original connection is still in TIMEWAIT state [7, pp. 945]. Accordingly, simple randomization, as suggested in [8], will not work well.

But duplicate packets, and hence the restrictions on the initial sequence number for reincarnations, are peculiar to individual connections. That is, there is no connection, syntactic or semantic, between the sequence numbers used for two different connections. We can prevent sequence number guessing attacks by giving each connection -- that is, each 4-tuple of <localhost, localport, remotehost, remoteport> -- a separate sequence number space. Within each space, the initial sequence number is incremented according to [2]; however, there is no obvious relationship between the numbering in different spaces.

The obvious way to do this is to maintain state for dead connections, and the easiest way to do that is to change the TCP state transition diagram so that both ends of all connections go to TIMEWAIT state. That would work, but it's inelegant and consumes storage space. Instead, we use the current 4 microsecond timer M and set

```
ISN = M + F(localhost, localport, remotehost, remoteport).
```

It is vital that F not be computable from the outside, or an attacker could still guess at sequence numbers from the initial sequence number used for some other connection. We therefore suggest that F be a cryptographic hash function of the connection-id and some secret data. MD5 [9] is a good choice, since the code is widely available. The secret data can either be a true random number [10], or it can be the combination of some per-host secret and the boot time of the machine. The boot time is included to ensure that the secret is changed on occasion. Other data, such as the host's IP address and name, may be included in the hash as well; this eases administration by permitting a network of workstations to share the same secret data while still giving them separate sequence number spaces. Our recommendation, in fact, is to use all three of these items: as random a number as the hardware can generate, an administratively-installed pass phrase, and the machine's IP address. This allows for local choice on how secure the secret is.

Note that the secret cannot easily be changed on a live machine. Doing so would change the initial sequence numbers used for reincarnated connections; to maintain safety, either dead connection state must be kept or a quiet time observed for two maximum segment lifetimes after such a change.

## A Common TCP Bug

As mentioned earlier, attackers using sequence number guessing have to "gag" the trusted machine first. While a number of strategies are possible, most of the attacks detected thus far rely on an implementation bug.

When SYN packets are received for a connection, the receiving system creates a new TCB in SYN-RCVD state. To avoid overconsumption of resources, 4.2BSD-derived systems permit only a limited number of TCBs in this state per connection. Once this limit is reached, future SYN packets for new connections are discarded; it is assumed that the client will retransmit them as needed.

When a packet is received, the first thing that must be done is a search for the TCB for that connection. If no TCB is found, the kernel searches for a "wild card" TCB used by servers to accept connections from all clients. Unfortunately, in many kernels this code is invoked for any incoming packets, not just for initial SYN packets. If the SYN-RCVD queue is full for the wildcard TCB, any new packets specifying just that host and port number will be discarded, even if they aren't SYN packets.

To gag a host, then, the attacker sends a few dozen SYN packets to the rlogin port from different port numbers on some non-existent machine. This fills up the SYN-RCVD queue, while the SYN+ACK packets go off to the bit bucket. The attack on the target machine then appears to come from the rlogin port on the trusted machine. The replies -- the SYN+ACKs from the target -- will be perceived as packets belonging to a full queue, and will be dropped silently. This could be avoided if the full queue code checked for the ACK bit, which cannot legally be on for legitimate open requests. If it is on, RST should be sent in reply.

## Security Considerations

Good sequence numbers are not a replacement for cryptographic authentication. At best, they're a palliative measure.

An eavesdropper who can observe the initial messages for a connection can determine its sequence number state, and may still be able to launch sequence number guessing attacks by impersonating that connection. However, such an eavesdropper can also hijack existing connections [11], so the incremental threat isn't that high. Still, since the offset between a fake connection and a given real connection will be more or less constant for the lifetime of the secret, it is important to ensure that attackers can never capture such packets. Typical attacks that could disclose them include both eavesdropping and the variety of routing attacks discussed in [8].

If random numbers are used as the sole source of the secret, they MUST be chosen in accordance with the recommendations given in [10].

## Acknowledgments

Matt Blaze and Jim Ellis contributed some crucial ideas to this RFC. Frank Kastenholz contributed constructive comments to this memo.

## References

[1] R.T. Morris, "A Weakness in the 4.2BSD UNIX TCP/IP Software", CSTR 117, 1985, AT&T Bell Laboratories, Murray Hill, NJ.

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

[3] Kohl, J., and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[4] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 1825, August 1995.

[5] Postel, J., and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[6] Jacobson, V., Braden, R., and L. Zhang, "TCP Extension for High-Speed Paths", RFC 1885, October 1990.

[7] G.R. Wright, W. R. Stevens, "TCP/IP Illustrated, Volume 2", 1995. Addison-Wesley.

[8] S. Bellovin, "Security Problems in the TCP/IP Protocol Suite", April 1989, Computer Communications Review, vol. 19, no. 2, pp. 32-48.

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

[10] Eastlake, D., Crocker, S., and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[11] L. Joncheray, "A Simple Active Attack Against TCP, 1995, Proc. Fifth Usenix UNIX Security Symposium.

Author’s Address

Steven M. Bellovin
AT&T Research
600 Mountain Avenue
Murray Hill, NJ 07974

Phone: (908) 582-5886
EMail: smb@research.att.com