Understanding and preventing TCP attacks

HIJACK PREVENTION

It is quite easy to take a TCP connection down using a RST attack, and this risk increases with applications that need long-term connections, such as VPNs, DNS zone transfers, and BGP. We'll describe how a TCP attack can happen, and we'll show you some simple techniques for protecting your network.

BY CHRISTOPH WEGENER AND WILHELM DOLLE

Since 1985, experts have known that the Transmission Control Protocol (TCP) is unsafe. Attackers can take down, corrupt, or even hijack existing TCP connections armed with just a few basic details of the connection: the source IP address, the target address, and a valid sequence number. If an attacker is able to sniff the connection, the battle is over before it begins. If the attacker can't do so, because they don't control a machine in the path between the client and the server, things become a little more complicated. However, people have definitely overestimated the effort required to undermine a connection, and tricks like TCP windowing make remote manipulation even easier.

One of the most difficult issues, is that of guessing the right sequence numbers. This is the only way to convince the target machine that the injected IP datagrams really belong to the current TCP connection. If an attacker has the right values, there is nothing to stop him from injecting data into the existing connection, thus gaining unauthorized access to information or taking down the connection by transmitting a packet with the Reset flag (RST) set.

In many cases a reset is just a minor irritation; this is particularly true if you are just surfing the web. In other situations, it can cause major problems: repeated interruptions to a BGP (Border Gateway Protocol) connection between two core routers on the Internet could be expensive.

But a DoS attack could potentially be a major threat to smaller enterprise networks as well. Imagine malevolent hackers taking down an online shop for an extended period; the resulting revenues losses could be considerable [1].

Background

We need a bit of background information on TCP to understand the vulnerability. The protocol was specified in RFC 793 [2], which was published in 1981. Each TCP segment starts with a header, which contains the source and target ports (both 16-bit values between 0 and 65 535) and other important parameters, such as the sequence and acknowledgment numbers, both 32 bits. Add a smattering of control flags (SYN, ACK, PSH,

THE AUTHOR

Christoph Wegener has a PhD. in physics and is head of Business Development at gits AG. He has been a freelance consultant for Linux and IT security for many years.

THE AUTHOR

Wilhelm Dolle is a CISSP certified and a BSI licensed IT security auditor. As a member of the Interactive Systems GmbH (iAS) management team, he is responsible for Information Technology and IT Security.

Table 1: Initial Window Size

Operating system	Window Size
Linux Kernel 2.4	5840 Bytes
Linux Kernel 2.6	5840 Bytes
OpenBSD 3.6	16,384 Bytes
Windows 2000 SP1, SP2	17,520 Bytes
Windows 2000 SP3, SP4	65,535 Bytes
Windows 2000 MS05-019	17,520 Bytes
Windows XP Professional, SP2	65,535 Bytes

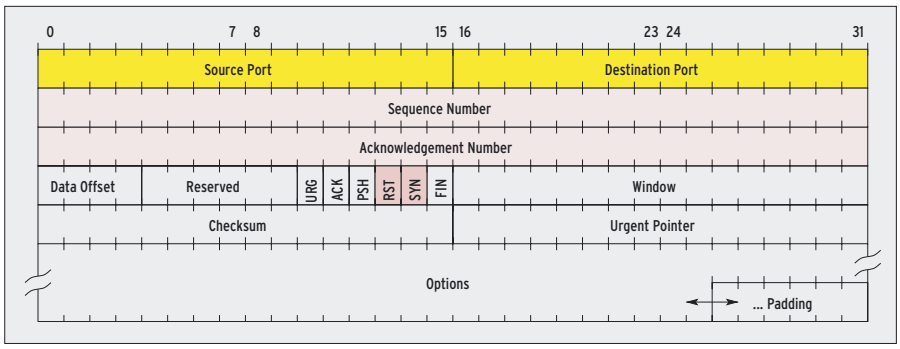

Figure 1: Every TCP segment starts with this header, followed by the payload. In combination with the IP addresses, the port numbers uniquely identify the connection, and the sequence number identifies the location of the packet within the data stream.

URG, RST und FIN) and the size of the receive window. The latter is critical to the exploit we will be discussing (Figure 1).

An established connection is uniquely identified by a quadruple (group of four) consisting of the source IP and port, and the target IP and port. The sequence number identifies the position of the segment in the data stream belonging to the connection to a byte. Anyone who has the details of the quadruple and can guess a valid sequence number has all the information he or she needs to gatecrash a connection. No matter where the attacker may be, he or she can just spoof a packet and send it on its way.

Random Numbers

In principle, a sequence number can have one of 2^{32} possible values. The odds on guessing the right number, using it to craft a packet, and injecting the packet into the connection at the right time, are very slim. However, the odds can change if the sender and receiver do not use random initial sequence numbers (ISN) when setting up the TCP connection (3 way handshake). As the connection progresses, the sequence number is just incremented for each byte transmitted.

Whereas the target port number is typically dictated by the application and the service listening on that port, the source port can have any value between 0 and 65,535. The fact that the first 1024 ports on Unix are reserved for privileged processes is not important to this evaluation. For a long time, people just assumed that an attacker would need to try 2^{32} sequence numbers, multiplied by 2^{16} possible source port numbers, to remotely attack a TCP connection without sniffing the connection. Unfortunately, most operating systems do not select the source port at random; but more of that later.

The biggest problem with TCP has to do with the windowing mechanism. Packets can overtake each other on the Internet. The receive window allows the receiving end to put the individual segments back together in the right order, and to confirm reception of a group of segments.

The RFC 793 glossary explains the window mechanism as follows: "This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND – 1 carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded."

Closing the Window

If the sequence number of a packet is within the receive window, TCP will accept and process the packet. This considerably reduces the number of guesses an attacker would need; in the case of sequence numbers, the number of guesses drops drastically from 2^{32} to 2^{32}/window size.

Depending on the operating system, the window size can be between 65,535 bytes (Windows XP Professional with SP2) and 5840 bytes (Linux kernel 2.4 and 2.6). Table 1 has more values for the initial window size. The window size also varies depending on the application. Whereas Cisco Telnet uses a size of 4192 bytes, the value for Cisco BGP is 16,384 bytes.

No matter how you look at it, the receive window will reduce the number of sequence numbers an attacker needs to investigate. If you take a Windows XP system, the numbers drop to 65,000. In other words, an attacker would just need to generate 65,000 attack packets to inject a valid RST packet and thus take down the connection. This is a terrifyingly small number. Good intrusion detection systems (IDS) will trigger due to the large number of RST packets, but hard working networks without this fea-

History

1981: The Transmission Control Protocol (TCP) is specified in Request for Comment (RFC) 793 [2].

1985: Bob Morris points out vulnerabilities in TCP [14].

1994: The first publicized manipulation of TCP vulnerabilities occurs when Kevin Mitnick uses the so-called Christmas Day attack to hit security expert Tsutumo Shimomura [19].

1995: Paul Watson posts an advisory on TCP vulnerabilities on Usenet. The advisory receives a significant attention. A number of investigations follow, especially in the field of sequence number generators.

1995: Laurent Joncheray presents a paper on the topic "Simple Active Attack against TCP" at the Usenix conference [15].

2001: Cert describes a vulnerability in various TCP/IP sequence number generators and points to the window size issue [16].

2003: Paul Watson shows that attacks are quite easy to perform even using a simple DSL connection.

2004: The IETF (Internet Engineering Task Force) publishes an initial version of the Internet draft "Improving TCP's Robustness to Blind In-Window Attacks" [10].

ture would not even notice the traffic from these packets.

Highly Scalable

Things get even worse if the parties in a connection support window scaling. This TCP extension (RFC 1323, [3]) increases the odds of finding a matching sequence number within a short time. Window scaling is designed for connections that need more than the standard window size due to high bandwidth or latency. To allow everyone to transmit without interruptions, and without waiting for acknowledgments, this technique scales the receive window size by up to 14 bits (Microsoft Windows). This is a factor of 16,384. The receive window is only restricted to a value of 65,535 bytes as specified in RFC 793 if neither end of the connection uses window scaling.

The attacker still has to overcome one more obstacle: the source and target IP address/port quad. The IP addresses are no problem – attackers know who they are gunning for – and the target port is just as easy. It is slightly more difficult to guess the source port, which could be anywhere between 0 and 65,535 in theory. The range is smaller in practical applications with ports below 1024 and above a certain operating system specific threshold reserved for special tasks.

A Linux system (with kernel 2.4 or 2.6) and at least 128MB RAM uses source ports between 32,768 and 61,000 (or less than this, if the system does not have 128MB memory). The kernel typically uses the ports above 61,000 for masquerading. You can check out these values in */proc/sys/net/ipv4/ip_local_range*, and change then using sysctl, for example: *sysctl -w net.ipv4.ip_local_range = "32768 61000"*.

No Need to Guess

Much to the delight of attackers, the remaining 28,232 options are by no means randomly distributed; the kernel assigns them based on a specific scheme. This is one revelation in Paul Watson's [4] advisory. Attackers should have no trouble predicting source ports. There are only a few exceptions, such as OpenBSD, that actually bother randomly assigning source ports. For example, Windows XP starts with source port 1050 for the first connection, and increments this number for every consecutive connection. Linux (Fedora Core 3 with kernel 2.6.9 in this case) starts at 32,768 and again increments the numbers.

Figure 2a shows a Linux system with kernel 2.6 (ports 32,771 through 32,777.) Compare this with Figure 2b, which shows OpenBSD 3.6 with random source port assignments. Cisco systems increment the port by 512 for each new connection, but that doesn't make the mechanism any safer.

Attackers do not need to guess the source port if they know the current connection number on the victim's machine. All an attacker typically needs to do is start with the known initial value, and try, say, 50 ports. Inquisitive hackers should have no trouble at all fingerprinting the operating system. So, in fact, predicting source ports is not really an obstacle.

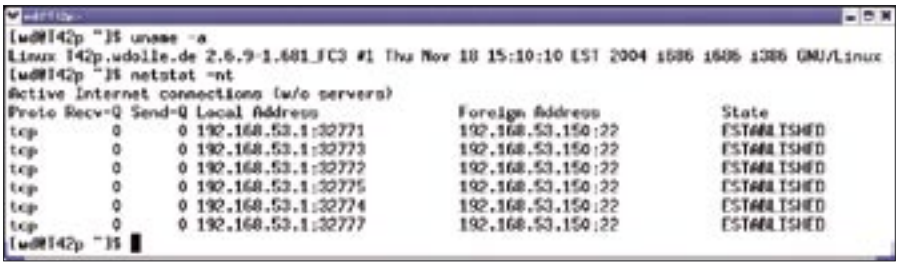

Figure 2a: The netstat -nt command lists the TCP connections for the local machine. The source port number follows a simple scheme. The kernel increments the number for each connection.

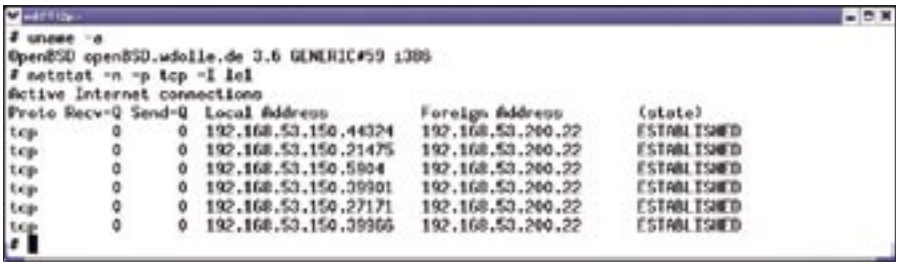

Figure 2b: OpenBSD goes to the trouble of making things difficult for attackers. This includes assigning a random source port for each connection, which forces the attacker to guess the number rather than predicting it.

Attack Techniques

Many attacks on TCP connections rely on the vulnerabilities discussed so far. One exploit involves an attacker injecting RST (Reset) control bits. According to RFC 793, this flag tells the target to drop the connection without any further interaction. The target evaluates the sequence number and possibly the acknowledgment number to decide whether to honor or ignore the reset command. The target is not permitted to send an RST in reply.

The important thing about the specification is that the target always has to verify the RST based on the validity of the sequence number. The target only closes the connection if the sequence number is within the receive window. Although the target could evaluate the acknowledgment number, there is no need for it to do so. Security expert Paul Watson (see the box titled "History") investigated a large number of TCP stacks for this and discovered that most of them simply ignore the acknowledgment number [4].

An RST packet received within the permitted window, and with data that match the connection, will always lead to the connection being terminated. Long-term connections, such as BGP connections between routers, are particularly vulnerable to reset attacks. For one thing, an attacker has enough time to insert a carefully crafted packet, on the other hand, the damage that DoS could cause is extreme. The routers need to reconfigure their neighbor tables, and this could take a few minutes under real-life conditions.

Synchronous Demise

What is less obvious is the fact that a SYN flag is also capable of taking a connection down. RFC 793 specifies that when the SYN flag is set at the start of a connection, the sequence number field must hold the starting value for the sequence numbers to be used later. If a

SYN packet arrives later on in the connection, RFC 793 states that this is an error. As a consequence, the receiver is forced to cancel the connection immediately by transmitting a reset packet.

This is TCP's way of avoiding the same connection being initiated twice, when one of the sides reboots, for example. Setting a RST or SYN flag in an IP datagram with a valid sequence number has the same effect: it closes the connection. In contrast to a RST packet, the host will respond to the SYN packet by transmitting a RST. There is a technical term for this behavior: we say that the receiver reflects the packet. And this opens the way to another DoS attack. The attacker can use up a victim's bandwidth. This technique is particularly successful in case of ADSL lines. The victim receives the data faster than it can transmit the responses.

Whereas RST and SYN attacks do not use a payload in the IP datagram, a third technique injects data into an existing connection. The attacker could inject arbitrary data that corrupts the connection, or the attacker might craft the data carefully to provoke an error condition. The victim might not even notice the manipulation.

Practical Applications

To test his theories [4] under real conditions, Paul Watson developed the *reset_tcp.c* tool, which he published in May 2004 [12]. What Paul noticed was that a simple DSL connection was all it took to guess a sequence number that would take down a connection within eleven minutes, given a window size of 65,535 bytes and 50 source ports to investigate. With a receive window size of only 16,384 bytes, the process would take 45 minutes.

The program requires the "Libnet Packet Construction Library" [13] by Mike D. Schiffman. You need to specify your own MAC address and the MAC address of the target interface before building the program. The latter is typically your default gateway or the interface on the victim machine, if this happens to be on the local network.

When called, the program expects a few parameters: *reset_tcp interface sourceIP sourceport targetIP targetport windowsize*. The interface is the network card via which the RST packets will be leaving the attacking machine.

The first practical test was to have the tool take down an SSH connection between two Linux computers (Figure 3). Both machines use T-Com T-DSL 1000 to connect to the Internet (upstream: 128Kbps). For this simple test, we will assume that the attacker already knows the source port. The RST packet size is 40 bytes (for the IP and TCP headers), or 320 bits if you prefer.

Let's assume that the window size is 5840 in both directions. Based on the theory, we can work out how long it will take at the most to take down the connection: the maximum value of the sequence number, divided by the window size, multiplied by the size of a packet, divided by the transfer rate. Adding the values gives us: 4294967296 / 5840 * 320 bits / 128000 bps, which comes to 1840 seconds, or 30 minutes and 40 seconds.

If we assume that all attempts have the same odds of succeeding, the attacker will actually only take half this time, that is, 15 minutes and 20 seconds. Our test results confirm this assumption:

Important Terms

Acknowledgment Number: This is a 32-bit element of the TCP segment header containing the sequence number that the sender of the data segment expects in the TCP segment header of the next IP datagram.

BGP: The Border Gateway Protocol describes how routers notify each other of the availability of communication routes. The strength of the BGP protocol is that it can merge various optional routing paths to form a single routing table.

Control Bits: These flags are part of the TCP segment header. There are six control bits:

- SYN: Synchronization request at the start of a connection.
- ACK: This packet confirms that the sender has received all the packets whose sequence number is smaller than the value specified in the acknowledgment number field.
- FIN: All data have been sent (finish; regular connection termination).
- RST: Cancel/Reset connection.
- PSH: A push flag tells the TCP stack to flush all buffers and transmit any pending data immediately, or forward that data to the appropriate application.
- URG: If the URG flag is set, the content of the Urgent header field points to the data that the receiver would like to expedite.

ICMP: The Internet Control Message Protocol (defined in RFC 792) is mainly used for troubleshooting and information exchanges.

Looking Glass: This service allows users to ascertain if a server is available and how good the connection is [6]. This is done by querying the BGP routers involved. The Looking Glass service gives users a clear overview of the connection quality. Ping and traceroute can provide additional information on the intermediate systems.

MD5 Signature: MD5 calculates a unique hash for an arbitrary data set. If a password is used to calculate the hash, MD5 can generate a signature (keyed hash).

Sequence Number: This is a 32 bit TCP segment header element which specifies the number of the first octet (byte) in the data segment. The receiver uses the sequence number to check the order and validity of the incoming packets. This protects the receiver against replay or injection attacks. However, this feature is designed to counteract random error and does not give the receiver much protection against manipulated packets.

TCP: The Transmission Control Protocol (defined in RFC 793) controls the data transfer between the sender and receiver. In contrast to the User Datagram Protocol (UDP, defined in RFC 768), TCP is connection oriented and ensures that all the data arrives uncorrupted and in the right order.

Window Size: This is a 16 bit TCP segment header element that specifies the number of data octets (bytes) that the sender of the TCP segment will accept as valid.

Window Scaling: An approach to optimizing the performance of high bandwidth connections or connections with high latency times. Window Scaling involves increasing the size of the receive window to allow entities to process packets that arrive late, and to transmit more data without waiting for an acknowledgment.

we recorded an average value of 932 seconds (15 minutes and 32 seconds) for a series of 20 tests that we performed. Let's say you needed to test 50 source ports (on a computer with just a few network connections); the time required to perform the attack would be about 13 hours. This is a lot of effort, even for a long-term SSH connection.

Variable Windows

Most operating systems modify the window size in an active connection to reflect the traffic volumes. For example, Linux increases the window size for an SSH connection which is just transferring the output from the *top* command to a value of more than 16,000. If the attacker knows that the victim uses the connection to transfer larger volumes of data, the attacker can leverage the larger window size. We repeated the tests for our example of a user running *top* over an SSH connection and measured a time of 5 minutes and 45 seconds for a window size of 16,000, assuming a known source port.

We based another set of experiments on Watson's example of a BGP connection: a Linux computer with (kernel 2.4, initial window size 5,840) using BGP to talk to a Cisco router at the other end (initial window size 16,384). The window size changes, as expected, when traffic is exchanged. At the start of the connection, BGP needs to transmit fairly high volumes of data; in our scenario, the window size grew in both directions to 16,616 within a few minutes before stabilizing at that level. The theoretical average time for an attack would thus be 4294967296 / 16616 * 320 Bit / 128.000 Bit/s / 2 = 5 minutes and 23 seconds.

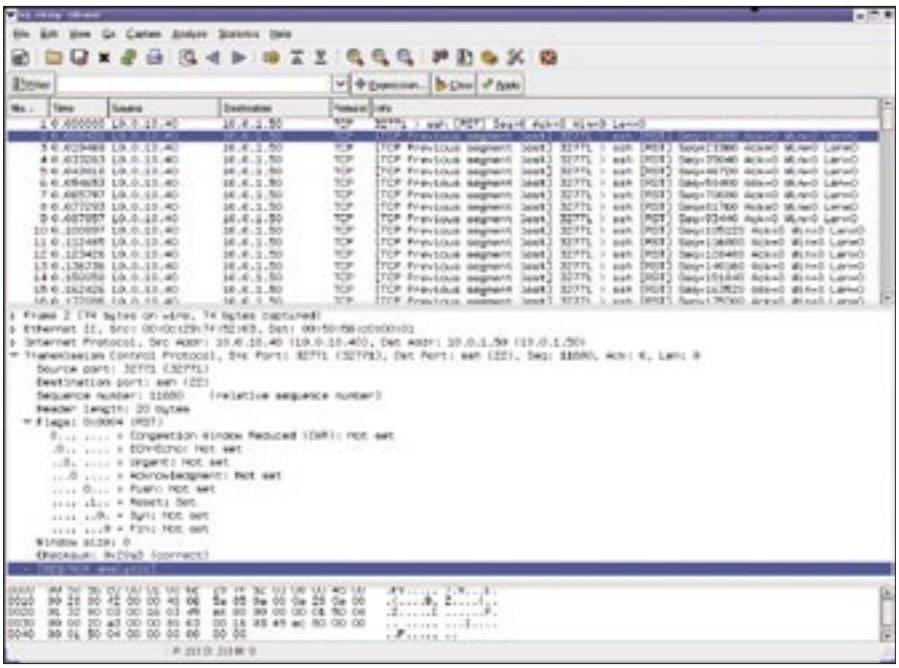

Figure 3: Ethereal monitoring a TCP reset attack over an SSH connection. The attacker transmits a large number of TCP segments with rising sequence numbers (top window, near the end of the line: Seq=....) The detail view (center) clearly shows that the reset flag has been set.

The value of 5 minutes and 39 seconds that we recorded for this test confirms the theory. BGP connections typically stay up for weeks, or even months; it typically takes over a minute to establish a connection and BGP routers do not open many network connections when left to their own devices. This makes them easy targets for attackers who have plenty of time to check just a few source ports.

Proactive Protection

Due to the high level of exposure and the low risk for the attacker, it is important to take preventive measures. There are a few approaches to mitigating the effect of the issues we referred to just earlier. As a general rule, you will want to avoid publishing information about your connection and network configuration as much as possible. Looking glasses, for example, are far too promiscuous (BGP; see the "Important Terms" box and [6]) as are some DNS entries.

Many operating systems allow admins to set the receive window size (see the "Changing the Window Size" box); a small value makes the system harder to sabotage. You should do without window scaling if possible. As a result, fewer sequence numbers fit in the receive window, and the attacker needs to be more precise, which in turn costs time and bandwidth.

However, there are limits to this kind of tuning. If the values you select are too low, the network performance will suffer. TCP runs slower because the protocol software has to wait for acknowledgments rather than transmitting. And it has to transmit more acknowledgments (ACKs); the overhead therefore increases. Let's look at an extreme example: if you have a window size of 10, a 40 byte ACK packet is required for 10 bytes of data, (20 bytes of IP header, 20 bytes of TCP header.)

Well Filtered

Filter rules give you more granular protection against reset attacks on border gateways. Routers should only accept incoming and outgoing traffic with IP

Changing the Window Size

The TCP receive window size is not fixed. Administrators can modify the default and maximum size values for most operating systems.

Cisco IOS: In enable mode, the window size can be set by entering *ip tcp window-size windowsize*.

Linux kernel 2.4 and 2.6 with IPv4: Modify the values of */proc/sys/net/ipv4/tcp_rmem* and */proc/sys/net/ipv4/wmem*, or enter values for *tcp_rmem* and *tcp_wmem* in */etc/sysctl.conf*, and then call *sysclt -p*. Refer to [18] for a detailed HOWTO.

Sun Solaris: On Solaris systems the *ndd* command does the trick: *ndd -set /dev/tcp tcp_xmit_hiwat windowsize* and *ndd -set /dev/tcp tcp_recv_hiwat windowsize*.

Windows 2000 and XP: Modify the values for *GlobalMaxTcpWindowSize* (type REG_DWORD) and *TcpWindowSize* (Typ REG_DWORD) in the *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters* registry entry. Refer to [17] for a detailed HOWTO.

source addresses that match the router interface through which the packet arrived. This reduces the risk of IP spoofing, and it should be a matter of course for any router configuration.

Ingress and egress filters protect the internal network against spoofed packets from the outside that claim to have internal addresses, and they protect the rest of the Internet against spoofers on the internal network. Cautious admins will add more filter rules that only accept BGP traffic from and for known routers. This makes attacks on the BGP connection more or less impossible.

Another kind of protection was introduced in 1998: RFC 2385 describes MD5-based signatures for TCP connections [7]. This approach creates MD5 hashes based on a password and all critical TCP header fields. This allows the receiver to detect spoofed packets. Of course, the passwords must be robust to prevent password crackers such as *bgpcrack* [8] using a dictionary attack to crack the signature. There is a useful overview of this BGP vulnerability, and others, along with possible solutions at [9].

Cautious Response

In April 2004, the IETF published an Internet draft [10] suggesting changes to TCP's response behavior, so that a TCP instance would only react immediately to incoming RST flags if they exactly matched the expected sequence number. If the number was simply within the receive window but not the next number up, the instance would respond by setting an ACK flag and dropping the segment it had just received. The recipient of the ACK packet could then respond by sending a second RST (Figure 4). The idea behind this is that a spoofer would not see the ACK packet and the odds would be no better than for the first attempt. If the RTS packet originated with a genuine sender, the sender could respond to the ACK and close the connection gracefully at the second attempt.

However, this new behavior introduces a new vulnerability: in a so-called ACK war, an attacker could flood a victim with RST packets. If the victim responds to each RST, the connection bandwidth is soon taken up with control traffic, and this would quickly block an ADSL connection. To avoid this, the suggestion was that each host would ACK a maximum of 10 RST packets in a period of 5 seconds.

The draft recommends verifying the validity of an incoming SYN by transmitting an ACK. All of this behavior is compatible with the original RFC 793-based behavior. The draft uses TCP features to combat TCP vulnerabilities. The danger of a DoS flooding attack is still real, however, despite all the improvements.

Linux has another technique for preventing attacks: the GR Security patch [11] ensures that kernels 2.4 and 2.6 assign arbitrary source ports. This feature comes from OpenBSD, like many of the patch's other features. Our experiments confirm that the patch really does prevented remote attacks.

Prevention Needed

The TCP RST and the related TCP SYN attacks are very dangerous. Any number of exploits are now available in the form of scripts and tools, and a simple DSL connection is all an attacker needs to take down semi-permanent connections. Referring to this type of attack, Theo de Raadt, the OpenBSD project maintainer, once said: "Lots of people are saying this is not a problem, but I am sure we will see a worm using it one day." ■

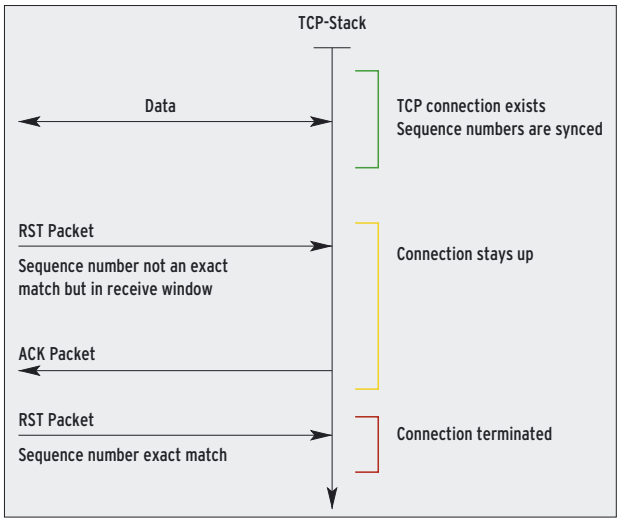

Figure 4: The modified stack based on [10] requires the RST sequence number to be an exact match.

INFO

[1] Computer Security Institute: *http://www.gocsi.com*

[2] RFC 793, "Transmission Control Protocol": *http://rfc.net/rfc793.html*

[3] RFC 1323, "TCP Extensions for High Performance": *http://rfc.net/rfc1323.html*

[4] Open Source Vulnerability Database, Paul (Tony) Watson, "Slipping in the Window: TCP Reset Attacks": *http://www.osvdb.org/reference/SlippingInTheWindow_v1.0.ppt*

[5] Open Source Vulnerability Database, Paul (Tony) Watson, "TCP Reset Spoofing": *http://www.osvdb.org/4030*

[6] IPv4 Looking Glass Sites: *http://www.bgp4.net/lg*

[7] RFC 2385, "Protection of BGP Sessions via the TCP MD5 Signature Option": *http://rfc.net/rfc2385.html*

[8] BGP Crack: *http://www.cisco.com/security_services/ciag/tools/bgpcrack-2.1.tar.gz*

[9] Sean Convery and Matthew Franz, "BGP Vulnerability Testing: Separating Fact from FUD v1.1": *http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-convery-franz-v3.pdf*

[10] Internet Draft, "Improving TCP's Robustness to Blind In-Window Attacks": *http://ietfreport.isoc.org/idref/draft-ietf-tcpm-tcpsecure/*

[11] GR Security: *http://www.grsecurity.net*

[12] TCP Connection Reset Remote Exploit: *http://www.frsirt.com/exploits/04232004.tcp-exploit.php*

[13] Libnet: *http://www.packetfactory.net/projects/libnet/*

[14] Robert T. Morris, "A Weakness in the 4.2 BSD Unix TCP/IP Software": *http://pdos.csail.mit.edu/~rtm/papers/117.pdf*

[15] Laurent Joncheray, "Simple Active Attack Against TCP": *http://www.usenix.org/publications/library/proceedings/security95/full_papers/joncheray.ps*

[16] Cert Advisory CA-2001-09, "Statistical Weaknesses in TCP/IP Initial Sequence Numbers": *http://www.cert.org/advisories/CA-2001-09.html*

[17] Dave MacDonald und Warren Barkley, "Microsoft Windows 2000 TCP/IP Implementation Details": *http://www.microsoft.com/technet/itsolutions/network/deploy/depovg/tcpip2k.mspx*

[18] Oskar Andreasson, "Ipsysctl tutorial – Chapter 3, IPv4 variable reference": *http://ipsysctl-tutorial.frozentux.net/chunkyhtml/tcpvariables.html*

[19] Cert Advisory CA-1995-01, "IP Spoofing Attacks and Hijacked Terminal Connections": *http://www.cert.org/advisories/CA-1995-01.html*