DRDoS

Distributed Reflection Denial of Service

Description and analysis of a potent, increasingly prevalent, and worrisome Internet attack

February 22nd, 2002 by Steve Gibson, Gibson Research Corporation

At 2:00 AM, January 11th, 2002, the GRC.COM site was blasted off the Internet by a new (for us) distributed denial of service attack.

Perhaps the most startling aspect of this attack was that the apparent source was hundreds of the Internet's "core routers", web servers belonging to yahoo.com, and even a machine with an IP resolving to "gary7.nsa.gov". We appeared to be under attack by hundreds of very powerful and well-connected machines.

Once we determined how to block this attack and returned to the Internet, 1,072,519,399 blocked packets were counted before the attack ended.

This page provides a brief tutorial on the operation of the Internet's TCP protocol, followed by explanations of the operation of traditional Internet denial of service (DoS), distributed denial of service (DDoS), and distributed reflection denial of service (DRDoS) attacks.

Bandwidth Consumption

As was true for this January 11th attack, any sort of "distributed" attack is most often a "bandwidth consumption" attack where the combined Internet connection bandwidth of many machines is "focused", or directed, upon one or a few machines. Although the attack's individual Internet packets may be completely harmless, a flood of such packets can overwhelm the target machine's Internet connection or other packet-processing resources. The result is that valid traffic, unable to compete with the malicious flood, has little chance of obtaining useful service.

For additional background on bandwidth consumption DDoS attacks, please see our previous page discussing the multiple DDoS attacks conducted by the malicious 13-year-old "Wicked." Click Here

However, since companies both large and small are now being harassed and even driven out of business as a consequence of malicious packet floods, an understanding and analysis of the exact methodology of a specific attack often suggests effective countermeasures (as was true in this instance). The need for effective forensic analysis is naturally heightened in instances of repeated and relentless attacks.

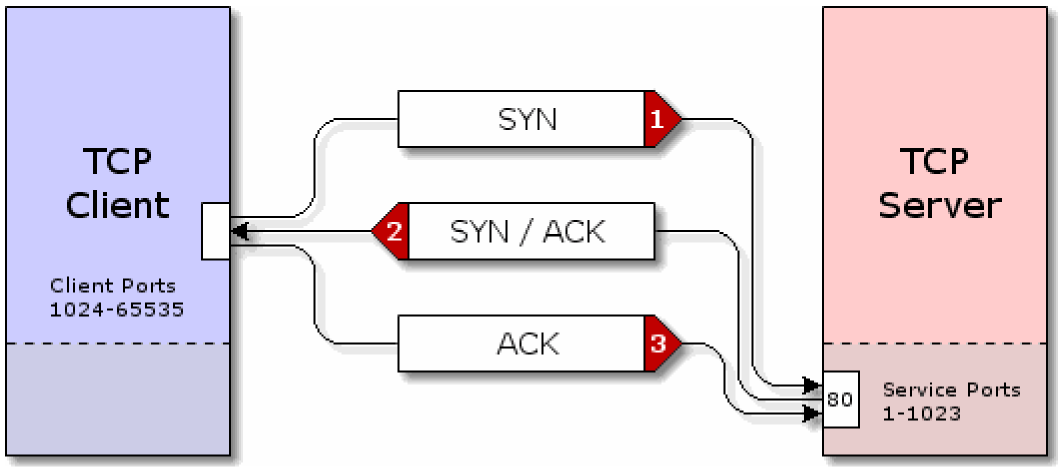
Before we can study and understand the mechanics of the distributed reflection attack, we need some understanding of the operation of TCP — the **T**ransmission **C**ontrol **P**rotocol used to connect remote machines over the Internet.

TCP Connections 101:

I can recall many years ago, well before the Internet "happened", hearing talk of two machines "connecting" to each other over the Internet. As a software-savvy hardware engineer, I remember thinking, "Connecting? How can two machines be 'connected' to each other over a global network?" I later learned that two machines, able to address and send packets of data to each other, negotiated a "connection agreement." The result of their successful negotiation is a "Virtual TCP Connection."

Individual TCP packets contain "flag bits" which specify the contents and purpose of each packet. For example, a packet with the "SYN" (synchronize) flag bit set is initiating a connection from the sender to the recipient. A packet with the "ACK" (acknowledge) flag bit set is acknowledging the receipt of information from the sender. A packet with the "FIN" (finish) bit set is terminating the connection from the sender to the recipient.

The establishment of a TCP connection typically requires the exchange of three Internet packets between two machines in an interchange known as the **TCP Three-Way Handshake.** Here's how it works:



1 SYN: A TCP client (such as a web browser, ftp client, etc.) initiates a connection with a TCP server by sending a "SYN" packet to the server.

As shown in the diagram above, this SYN packet is usually sent from the client's port, numbered between 1024 and 65535, to the server's port, numbered between 1 and 1023. Client programs running on the client machine ask the operating system to "assign them a port" for use in connecting to a remote server. This upper range of ports is known as the "client" or "ephemeral" port range. Similarly, server programs running on the server machine ask the operating system for the privilege of "listening" for incoming traffic on specific port numbers. This lower port range is known as "service ports." For example, a web server program typically listens for incoming packets on port 80 of its machine, and web browsing clients generally send their web packets to port 80 of remote servers.

Note that in addition to source and destination port numbers, each packet also contains the IP address of the machine which originated the packet (the Source IP) and the address of the machine to which the Internet's routers will forward the packet (the Destination IP).

2 SYN/ACK: When a connection-requesting SYN packet is received at an "open" TCP service port, the server's operating system replies with a connection-accepting "SYN/ACK" packet.

Although TCP connections are bi-directional (full duplex), each direction of the connection is set up and managed independently. For this reason, a TCP server replies to the client's connection-requesting SYN packet by **ACK**nowledging the client's packet and sending its own **SYN** to initiate a connection in the returning direction. These two messages are combined into a single "SYN/ACK" response packet.

The SYN/ACK packet is sent to the SYN's sender by exchanging the source and destination IPs from the SYN packet and placing them into the answering SYN/ACK packet. This sets the SYN/ACK packet's destination to the source IP of the SYN, which is exactly what we want.

Note that whereas the client's packet was sent **to** the server's service port — 80 in the example shown above — the server's replying packet is returned **from** the same service port. In other words, just as the source and destination IPs are exchanged in the returning packet, so are the source and destination **ports**.

The client's reception of the server's SYN/ACK packet confirms the server's willingness to accept the client's connection. It also confirms, for the client, that a round-trip path exists between the client and server. If the server had been unable or unwilling to accept the client's TCP connection, it would have replied with a RST/ACK (Reset Acknowledgement) packet, or an ICMP Port Unreachable packet, to inform the client that its connection request had been denied.

3 **ACK: When the client receives the server's acknowledging SYN/ACK packet for the pending connection, it replies with an ACK packet.**

The client **ACK**nowledges the receipt of the SYN portion of the server's answering SYN/ACK by sending an ACK packet back to the server. At this point, from the client's perspective, a new two-way TCP connection has been established between the client and server, and data may now freely flow in either direction between the two TCP endpoints.

The server's reception of the client's ACK packet confirms to the server that its SYN/ACK packet was able to return to the client across the Internet's packet routing system. At this point, the server considers that a new two-way TCP connection has been established between the client and server and data may now flow freely in either direction between the two TCP endpoints.

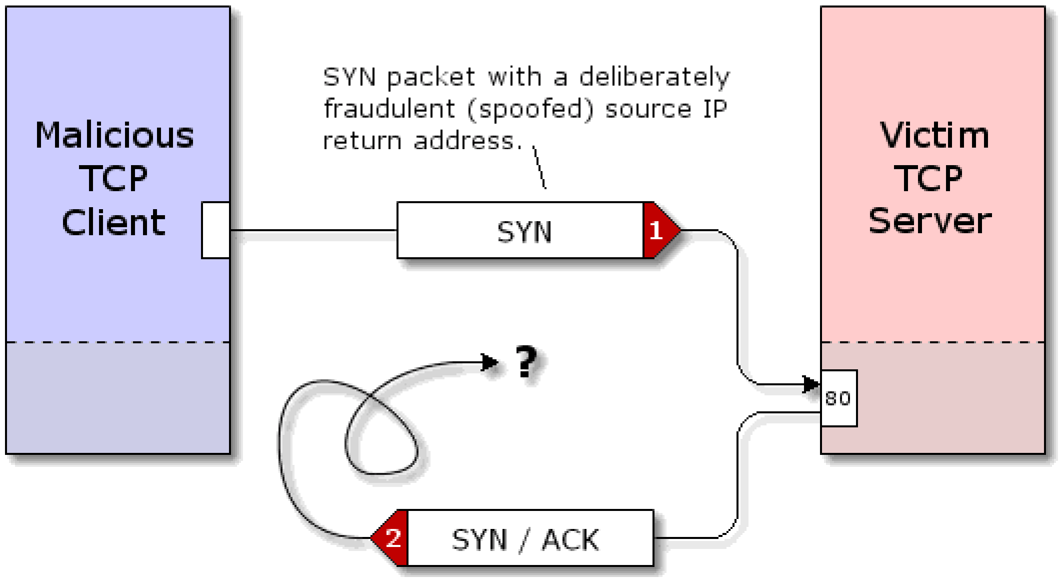
For More Information: The pages describing the design and operation of my custom DoS-proof TCP/IP protocol stack goes into much greater detail which is not relevant to this page's discussion.

Abusing TCP: The Traditional SYN Flood

Several years ago, a weakness in the TCP connection handling of many operating systems was discovered and exploited by malicious Internet hackers.

As shown in the TCP transaction diagram above, the server's receipt of a client's SYN packet causes the server to prepare for a connection. It typically allocates memory buffers for sending and receiving the connection's data, and it records the various details of the client's connection including the client's remote IP and connection port number. In this way, the server will be prepared to accept the client's final connection-opening ACK packet. Also, if the client's ACK packet should fail to arrive, the server will be able to resend its SYN/ACK packet, presuming that it might have been lost or dropped by an intermediate Internet router.

But think about that for a minute. This means that memory and other significant server "connection resources" are allocated as a consequence of the receipt of a single Internet "SYN" packet. Clever but malicious Internet hackers figured that there had to be a limit to the number of "half open" connections a TCP server could handle, and they came up with a simple means for exceeding those limits:


Malicious TCP Client
SYN packet with a deliberately fraudulent (spoofed) source IP return address.
SYN
1
Victim TCP Server
80
?
2
SYN / ACK

Through the use of "Raw Sockets", the packet's "return address" (source IP) can be overridden and falsified. When a SYN packet with a spoofed source IP arrives at the server, it appears as any other valid connection request. The server will allocate the required memory buffers, record the information about the new connection, and send an answering SYN/ACK packet back to the client.

But since the source IP contained in the SYN packet was deliberately falsified (it is often a random number), the SYN/ACK will be sent to a random IP address on the Internet. If the packet were addressed to a valid IP, the machine at that address might reply with a "RST" (reset) packet to let the server know that it did not request a connection. But with over 4 billion Internet addresses, the chances are that there will be no machine at the address and the packet will be discarded.

The problem is, the server has no way of knowing that the malicious client's connection request was fraudulent, so it needs to treat it like any other valid pending connection. It needs to wait for some time for the client to complete the three-way handshake. If the ACK is not received, the server needs to resend the SYN/ACK in the belief that it might have been lost on its way back to the client.

As you can imagine, all of this connection management consumes valuable and limited resources in the server. Meanwhile, the attacking TCP client continues firing additional fraudulent SYN packets at the server, forcing it to accumulate a continuously growing pool of incomplete connections. At some point, the server will be unable to accommodate any more "half-open" connections and even **valid** connections will fail, since the server's ability to accept **any** connections will have been maliciously consumed.

This is NOT bandwidth consumption

Before operating systems' TCP support was enhanced to mitigate the effect of these SYN floods, even a single malicious machine using a slow dial-up connection could fill and consume the "connection queues" of the highest performance Internet server. Although some advances in anti-SYN-spoofing vulnerability have been made, few effective solutions have been created.

It is important to understand that these early spoofed-source-IP SYN attacks were **not** bandwidth consumption attacks. Due to the susceptible nature of most operating systems' TCP/IP protocol handlers, very little inbound bandwidth was required to completely tie-up a TCP server. Rather than consuming the network's "bandwidth resource", the server's "connection resources" were consumed.

Also notice that this Denial of Service (DoS) attack was not "Distributed." It was a "DoS" attack, not any form of "DDoS" attack. A single, malicious, SYN-generating machine, hiding its Internet address and identity behind falsified source IP SYN packets, could tie-up and bring down a large web site.

Solving the SYN spoofing problem

Operating system vendors responded to spoofed SYN packet DoS attacks by strengthening their TCP "protocol stacks" in various ways. Most of these were quantitative improvements to make their systems less vulnerable, but they did not eliminate the problem.

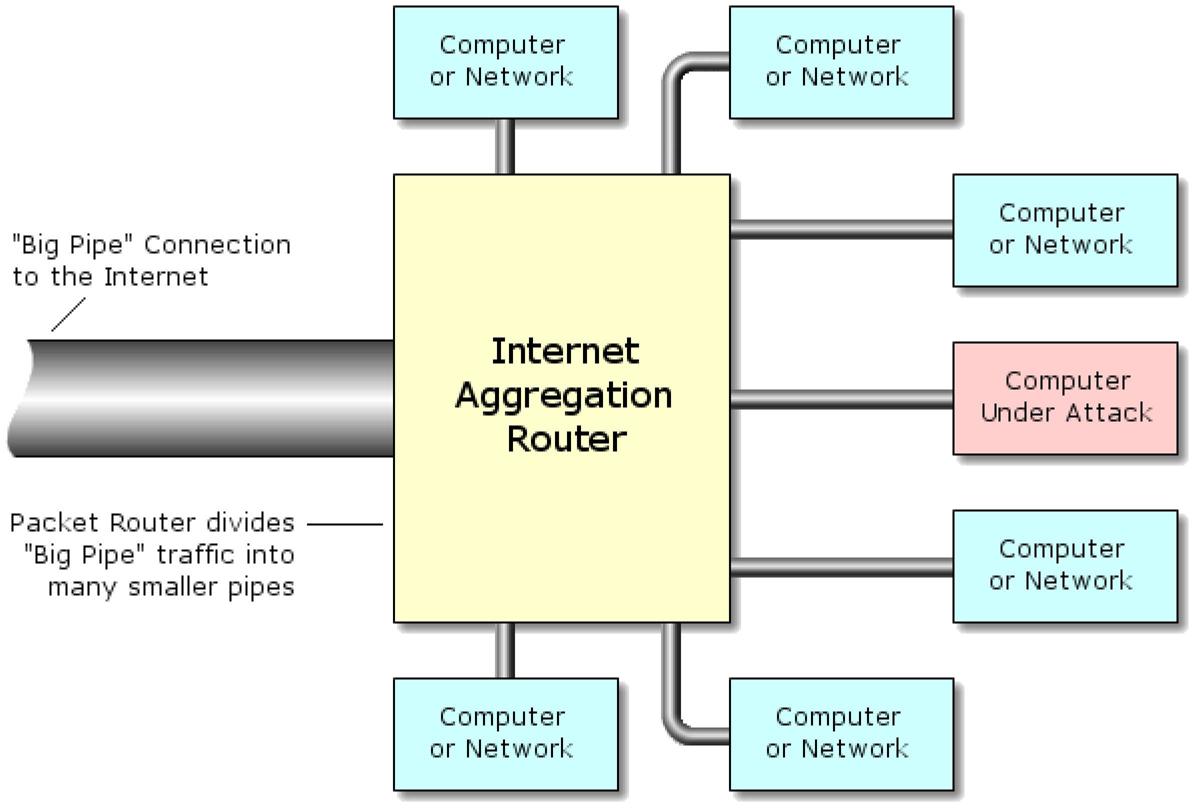
Two complete, robust, and practical solutions were developed: The Unix community invented a clever "stateless" TCP connection system known as "SYN-cookies". Not knowing about SYN-cookies, I independently created and implemented my own different solution which was dubbed "GENESIS". The GENESIS TCP solution has been in continuous service at grc.com since September of 2000. Both of these DoS solutions arrange to stay compatible with all important aspects of the standard TCP protocol. They operate by eliminating **all** allocation of server resources after receiving a SYN packet and generating a SYN/ACK reply.

The Development of Bandwidth Attacks

As the sophistication of malicious hackers has grown, and as the available inventory of insecure and readily compromised Internet-connected host machines has skyrocketed, "bandwidth consumption" **distributed** denial of service (**D**DoS) attacks have become commonplace. As I discovered and documented in May of 2001, powerful, remote Internet attack tools are now in the hands of children who wield their disruptive power with little thought for, or remorse over, the consequences.

What happens during a bandwidth attack?

Unlike a DoS-style attack, in which a low rate of fraudulent SYN packets consumes a vulnerable server's TCP connection resources, a bandwidth attack creates a brute force flood of malicious "nonsense" Internet traffic to swamp and consume the target server's or its network connection bandwidth. This malicious packet flood competes with, and overwhelms, the network's valid traffic so that "good packets" have a low likelihood of surviving the flood. The network's servers become cut off from the rest of the Internet, and their service is denied.



Discarding packets

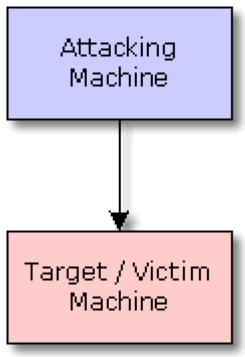
The diagram above helps clarify the consequences of a bandwidth attack. The computers and/or networks shown to the right are serviced by the central "aggregation router." This router is placed at the "customer edge" of the Internet service provider's network to collect and disperse traffic from many smaller customer networks. Thus, many lower-bandwidth Internet connections are "aggregated" into a single high-bandwidth Internet connection for routing to the public Internet.

During normal operation, the traffic coming from the Internet down the "Big Pipe" will be sorted and forwarded to the router's various lower bandwidth client networks. But imagine what happens when the Big Pipe is **filled** by a high volume of packets bound for **just one** of the router's client networks. Faced with the task of squeezing too many packets from the big pipe into the much smaller pipe, the router has no choice but to deliberately drop and discard a large percentage of the packets struggling to get through the smaller pipe.

Valid Internet clients, trying to access the resources on the far side of the smaller pipe, will resend their dropped packets. But these clients will generally give up after a few attempts. The victim's network is effectively blasted off the Internet by the flood of malicious traffic.

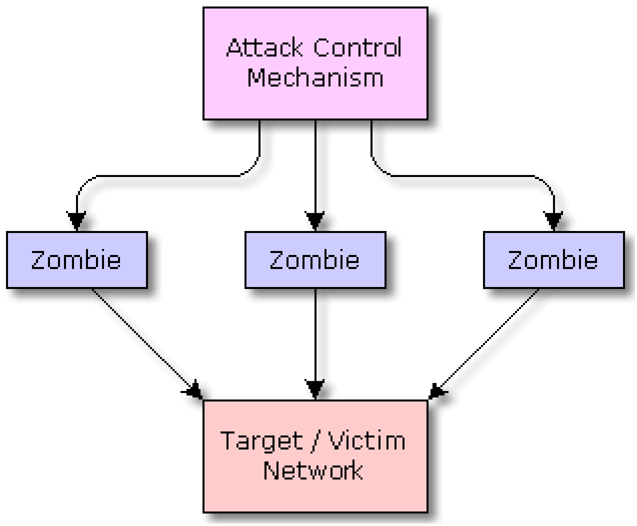
DoS versus DDoS

DoS: The traditional DoS style attack, where a single machine attacks another, can be depicted by this diagram **. . .**



As we saw in the routing diagram above, if the attacking machine enjoys a significantly higher speed Internet connection than the target/victim machine, it could successfully swamp the target's connection bandwidth. Thus, even one well-connected attacking machine can flood a less well-connected target to deny its access to the Internet.

DDoS: Much higher levels of flooding traffic can be generated by focusing the combined bandwidth of multiple machines onto a single target machine or network **. . .**

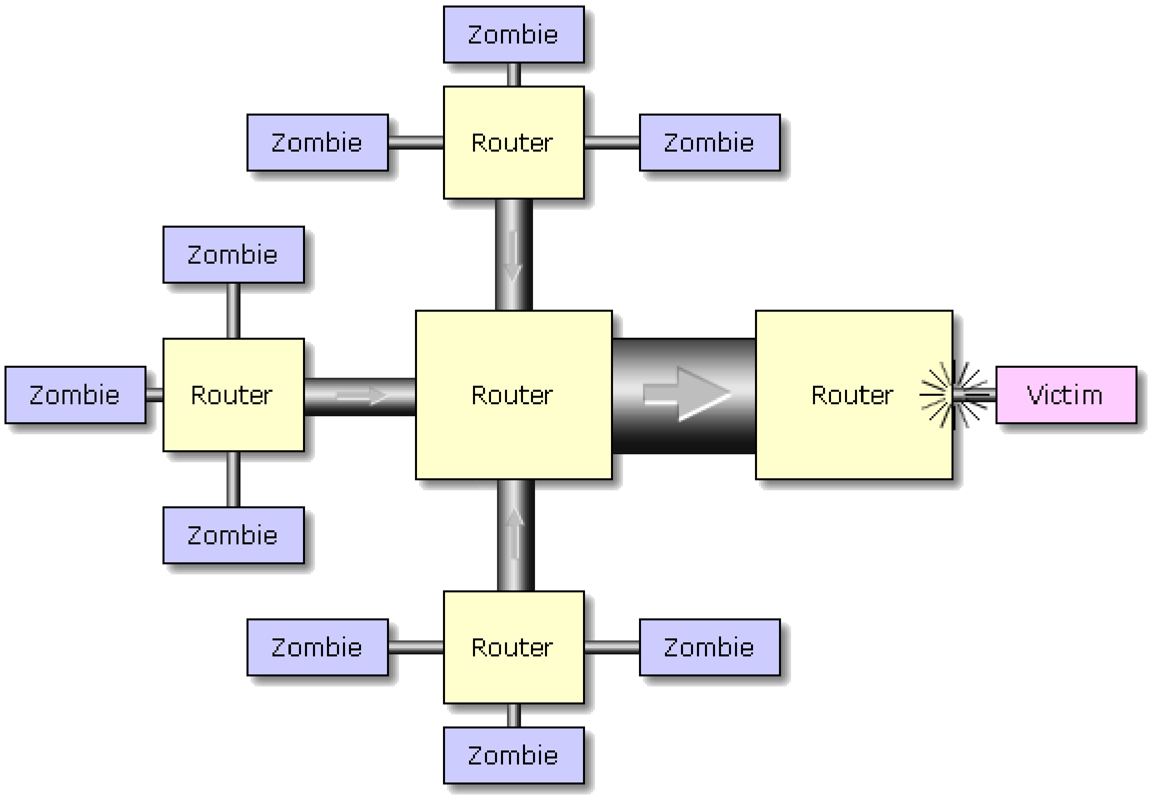


The diagram above shows the architecture commonly used in distributed denial of service attacks. The operation of a network of compromised machines, containing remotely controlled "Zombie" attack programs, is directed and coordinated by a "Zombie Master" central control agency. When the network of Zombies receives instructions from its Master, each individual Zombie begins generating a flood of malicious traffic aimed at a single target/victim machine or network.

This is the organization used by the many popular distributed attack tools, including the Windows-hosted Evilbots driven by the 13-year-old "Wicked" that originally attacked grc.com during May of 2001. The Evilgoat Evilbot employs public IRC servers as their meeting place and attack control mechanism. The Zombie Master joins a secret IRC chat room to issue real-time attack instructions.

Distributed zombie traffic aggregation

As the individual streams of traffic move across the Internet from their many separate sources, they are combined by the Internet's routers to form a single massive flood **. . .**



When the flood finally encounters the target network's service provider aggregation router, most of the network's traffic must be discarded. Since the router cannot differentiate valid from invalid traffic — to the router, all packets are created equal — the network's valid traffic will also be discarded **. . .** effectively cutting the network off from the rest of the Internet.

As if all of the foregoing weren't troublesome enough, now it gets much worse. . .

Distributed Reflection
A Next-Generation DDoS Attack

At 2:00 AM, January 11th 2002, grc.com was blasted off the Internet by a more advanced malicious packet flood. This new style of DDoS attack could be called a **Distributed Reflection Denial of Service** attack — **DRDoS.**

A packet flood mystery

I was awake and working at 2:00 AM when the attack began, so I was able to quickly begin capturing the portion of the flood that Verio's (our service provider) aggregation router was able to squeeze down through our two T1 trunks. Our web server's normal outbound traffic had dropped to zero because valid server requests from the Internet were unable to compete with the flood. We were off the air.

In the past, we had been hit by Evilbot's non-spoofed UDP and ICMP floods. These are easily generated by commandeered, Zombie-infected, Windows machines. We have also been blasted by many sorts of classic spoofed source IP SYN floods. So my eyebrows jumped a bit when the attack's packet capture revealed that we were being flooded by SYN/ACK packets.

Of course, **that** fact in itself wasn't a big deal. As we saw earlier on this page, a SYN/ACK packet is just a SYN packet with its "ACK" flag bit turned on. Anyone with access to a "raw socket" capable machine can generate any sort of packet — benign or malicious — that they desire. So an attacker can turn on whatever "TCP flag bits" they like.

The real surprise came when I looked into the source IPs of the flooding packets:

Source IP	Machine Name
129.250. 28. 1	ge-6-2-0.r03.sttlwa01.us.bb.verio.net
129.250. 28. 3	ge-1-0-0.a07.sttlwa01.us.ra.verio.net
129.250. 28. 20	ge-0-1-0.a12.sttlwa01.us.ra.verio.net
129.250. 28. 33	ge-0-0-0.r00.bcrtfl01.us.bb.verio.net
129.250. 28. 49	ge-1-1-0.r01.bcrtfl01.us.bb.verio.net
129.250. 28. 98	ge-1-2-0.r00.sfldmi01.us.bb.verio.net
129.250. 28. 99	ge-1-0-0.a00.sfldmi01.us.ra.verio.net
129.250. 28.100	ge-1-1-0.a01.sfldmi01.us.ra.verio.net
129.250. 28.113	ge-1-2-0.r01.sfldmi01.us.bb.verio.net
129.250. 28.116	ge-1-1-0.a00.sfldmi01.us.ra.verio.net
129.250. 28.117	ge-1-0-0.a01.sfldmi01.us.ra.verio.net
129.250. 28.131	ge-0-3-0.a00.scrmca01.us.ra.verio.net
129.250. 28.142	ge-0-2-0.r00.scrmca01.us.bb.verio.net
129.250. 28.147	ge-1-2-0.a00.scrmca01.us.ra.verio.net
129.250. 28.158	ge-0-2-0.r01.scrmca01.us.bb.verio.net
129.250. 28.164	ge-1-0-0.a10.dllstx01.us.ra.verio.net
129.250. 28.165	ge-1-0-0.a11.dllstx01.us.ra.verio.net
129.250. 28.190	ge-6-0-0.r01.dllstx01.us.bb.verio.net
129.250. 28.200	ge-0-2-0.a00.snjsca03.us.ra.verio.net
129.250. 28.201	ge-0-2-0.a01.snjsca03.us.ra.verio.net
129.250. 28.221	ge-2-1-0.r04.snjsca03.us.bb.verio.net
129.250. 28.230	ge-1-1-0.a00.snjsca03.us.ra.verio.net
129.250. 28.231	ge-1-1-0.a01.snjsca03.us.ra.verio.net
129.250. 28.254	ge-2-1-0.r01.snjsca03.us.bb.verio.net
205.171. 31. 1	iah-core-01.inet.qwest.net
205.171. 31. 2	iah-core-02.inet.qwest.net
205.171. 31. 5	iah-core-01.inet.qwest.net
205.171. 31. 6	iah-core-03.inet.qwest.net
205.171. 31. 9	iah-core-01.inet.qwest.net
205.171. 31. 13	iah-core-01.inet.qwest.net
205.171. 31. 17	iah-core-01.inet.qwest.net
205.171. 31. 21	iah-core-01.inet.qwest.net
205.171. 31. 25	iah-core-02.inet.qwest.net
205.171. 31. 33	iah-core-01.inet.qwest.net
205.171. 31. 37	iah-core-01.inet.qwest.net
205.171. 31. 41	iah-core-02.inet.qwest.net
205.171. 31. 53	iah-core-02.inet.qwest.net
205.171. 31. 57	iah-core-03.inet.qwest.net
205.171. 31. 61	iah-core-02.inet.qwest.net
205.171. 31. 81	iah-core-03.inet.qwest.net
206. 79. 9. 2	globalcrossing-px.exodus.net
206. 79. 9.114	exds-wlhm.gblx.net
206. 79. 9.210	telefonica-px.exodus.net
208.184.232. 13	core1-atl4-oc48-2.atl2.above.net
208.184.232. 17	core2-atl4-oc48.atl2.above.net
208.184.232. 21	core1-atl4-oc48.atl2.above.net
208.184.232. 25	core2-core1-oc48.atl2.above.net

```
208.184.232. 45          core1-core2-oc192.sfo1.above.net
208.184.232. 46          core2-core1-oc192.sfo1.above.net
208.184.232. 54           sfo1-sjc2-oc48-2.sfo1.above.net
208.184.232. 57           ord2-sea1-oc48-2.ord2.above.net
208.184.232. 58           sea1-ord2-oc48-2.sea1.above.net
208.184.232. 97             bos2-dca2-oc48.bos2.above.net
208.184.232. 98             dca2-bos2-oc48.dca2.above.net
208.184.232.101           bos2-dca2-oc48-2.bos2.above.net
208.184.232.102           dca2-bos2-oc48-2.dca2.above.net
208.184.232.109            core1-dfw3-oc48.dfw2.above.net
208.184.232.110            core1-dfw2-oc48.dfw3.above.net
208.184.232.113            core2-dfw3-oc48.dfw2.above.net
208.184.232.114            core2-dfw2-oc48.dfw3.above.net
208.184.232.118            core1-dfw1-oc48.dfw2.above.net
208.184.232.126             sfo1-sjc2-oc48.sfo1.above.net
208.184.232.133           dca2-dfw2-oc48-2.dca2.above.net
208.184.232.134           dfw2-dca2-oc48-2.dfw2.above.net
208.184.232.145             ord2-bos2-oc48.ord2.above.net
208.184.232.146             bos2-ord2-oc48.bos2.above.net
208.184.232.149             lga1-ord2-oc48.lga1.above.net
208.184.232.150             ord2-lga1-oc48.ord2.above.net
208.184.232.157             atl2-lga2-oc48.atl2.above.net
208.184.232.158             lga2-atl2-oc48.lga2.above.net
208.184.232.165           atl2-lga2-oc48-2.atl2.above.net
208.184.232.166           lga2-atl2-oc48-2.lga2.above.net
208.184.232.177                  sjc3-pao1-oc12.above.net
208.184.232.189             bos2-lga2-oc48.bos2.above.net
208.184.232.190             lga2-bos2-oc48.lga2.above.net
208.184.232.193           bos2-lga2-oc48-2.bos2.above.net
208.184.232.194           lga2-bos2-oc48-2.lga2.above.net
208.184.232.197           core2-lga2-oc192.lga1.above.net
208.184.232.198           core2-lga1-oc192.lga2.above.net
208.184.233. 46             ord2-sjc2-oc48.ord2.above.net
208.184.233. 50            core2-sjc2-oc48.sjc3.above.net
208.184.233. 61          iad1-lga1-oc192-2.iad1.above.net
208.184.233. 62          lga1-iad1-oc192-2.lga1.above.net
208.184.233. 65            iad1-lga1-oc192.iad1.above.net
208.184.233. 66            lga1-iad1-oc192.lga1.above.net
208.184.233. 81     core1-main1colo56-oc48.sea2.above.net
208.184.233. 85     core1-main2colo56-oc48.sea2.above.net
208.184.233. 89     core2-main1colo56-oc48.sea2.above.net
208.184.233. 93     core2-main2colo56-oc48.sea2.above.net
208.184.233.101          core1-core2-oc192.sea2.above.net
208.184.233.102          core2-core1-oc192.sea2.above.net
208.184.233.105           core2-sea2-oc192.sea1.above.net
208.184.233.106         core2-sea1-oc192-2.sea2.above.net
208.184.233.121          core1-core2-oc192.dca2.above.net
208.184.233.126            iad1-dca2-oc192.iad1.above.net
208.184.233.129            dca2-iad1-oc192.dca2.above.net
208.184.233.130            iad1-dca2-oc192.iad1.above.net
208.184.233.134             dca2-sjc2-oc48.dca2.above.net
208.184.233.150             ord2-dfw2-oc48.ord2.above.net
208.184.233.174         globalcenter-above.iad2.above.net
208.184.233.189             sea1-nrt3-stm1.sea1.above.net
208.184.233.190             nrt3-sea1-stm1.nrt3.above.net
208.184.233.193           sea1-nrt3-stm1-3.sea1.above.net
208.184.233.194           nrt3-sea1-stm1-3.nrt3.above.net
208.184.233.197           core1-main1-oc12.nrt3.above.net
208.184.233.201           core1-main2-oc12.nrt3.above.net
208.184.233.205           core2-main1-oc12.nrt3.above.net
208.184.233.209           core2-main2-oc12.nrt3.above.net
208.184.233.217           core2-core3-oc48.lga1.above.net
208.184.233.225          core2-v6core3-oc3.nrt3.above.net
```

```
208.184.233.237           core1-oc192-core2.bos2.above.net
208.184.233.238           core2-oc192-core1.bos2.above.net
208.185.  0. 25               core5-dlr-oc3.iad1.above.net
208.185.  0.113            core5-main1-oc48.iad1.above.net
208.185.  0.117            core5-main2-oc48.iad1.above.net
208.185.  0.121             core4-iad4-oc48.iad1.above.net
208.185.  0.133             core5-iad4-oc48.iad1.above.net
208.185.  0.138            core4-core1-oc48.iad1.above.net
208.185.  0.142            core4-core3-oc48.iad1.above.net
208.185.  0.146            core5-core1-oc48.iad1.above.net
208.185.  0.150            core5-core3-oc48.iad1.above.net
208.185.  0.153            core4-main1-oc48.iad1.above.net
208.185.  0.157            core4-main2-oc48.iad1.above.net
208.185.  0.165            core1-core2-oc48.lga3.above.net
208.185.  0.166            core2-core1-oc48.lga3.above.net
208.185.  0.169             core1-lga3-oc12.lga1.above.net
208.185.  0.170             core1-lga1-oc12.lga3.above.net
208.185.  0.173           core1-core3-oc3-2.lga3.above.net
208.185.  0.177             core2-core3-oc3.lga3.above.net
208.185.  0.189            core1-core3-oc48.ord2.above.net
208.185.  0.193            core2-core3-oc48.ord2.above.net
208.185.  0.197             core1-ord1-oc48.ord2.above.net
208.185.  0.202             core2-ord1-oc48.ord2.above.net
208.185.  0.221            core1-core3-oc48.atl2.above.net
208.185.  0.225            core2-core3-oc48.atl2.above.net
208.185.  0.229            dca2-atl2-oc48-2.dca2.above.net
208.185.  0.230            atl2-dca2-oc48-2.atl2.above.net
208.185.  0.233           core1-core2-oc192.lga1.above.net
208.185.  0.234           core2-core1-oc192.lga1.above.net
208.185.  0.237            core1-core3-oc48.lga1.above.net
208.185.  0.245            core1-lga2-oc192.lga1.above.net
208.185.  0.246            core1-lga1-oc192.lga2.above.net
208.185.  0.249             core1-dfw2-oc48.atl2.above.net
208.185.  0.250             core1-atl2-oc48.dfw2.above.net
208.185.156.  2            core2-lhr1-stm16.lhr3.above.net
208.185.156. 65            core3-core5-oc48.sjc2.above.net
208.185.156.121          core2-sea2-oc192-2.sea1.above.net
208.185.156.122          core1-sea1-oc192-2.sea2.above.net
208.185.156.157            ord2-lga1-oc48-2.ord2.above.net
208.185.156.158            lga1-ord2-oc48-2.lga1.above.net
208.185.156.189       core3-main1colo7-oc12.sjc2.above.net
208.185.156.193       core4-main2colo7-oc12.sjc2.above.net
208.185.175. 90              ord2-sea1-oc48.ord2.above.net
208.185.175. 93             core3-core4-oc3.sea1.above.net
208.185.175.114              earthlink-above.lax.above.net
208.185.175.145           core1-core2-oc192.sjc3.above.net
208.185.175.146           core2-core1-oc192.sjc3.above.net
208.185.175.149            core2-sjc4-oc192.sjc3.above.net
208.185.175.158             core1-sjc2-oc48.sjc3.above.net
208.185.175.178            core2-core1-oc48.sea1.above.net
208.185.175.182            core3-core1-oc48.sea1.above.net
208.185.175.189      core1-main1colo56-oc48.sjc3.above.net
208.185.175.193      core1-main2colo56-oc48.sjc3.above.net
208.185.175.197      core2-main1colo56-oc48.sjc3.above.net
208.185.175.201      core2-main2colo56-oc48.sjc3.above.net
216.200.127.  9             core4-iad5-oc48.iad1.above.net
216.200.127. 13             core5-iad5-oc48.iad1.above.net
216.200.127. 26              sjc2-iad1-oc48.sjc2.above.net
216.200.127. 29              core4-epe1-oc3.iad1.above.net
216.200.127. 33              core5-epe1-oc3.iad1.above.net
216.200.127. 45              core1-epe1-oc3.lga1.above.net
216.200.127. 49              core2-epe1-oc3.lga1.above.net
216.200.127. 61            iad1-lga1-oc48-2.iad1.above.net
```

```
216.200.127. 62           lga1-iad1-oc48-2.lga1.above.net
216.200.127. 65             lga1-sea1-oc48.lga1.above.net
216.200.127. 66             sea1-lga1-oc48.sea1.above.net
216.200.127. 69           lga1-lhr1-stm4-3.lga1.above.net
216.200.127.118             sea1-sjc2-oc48.sea1.above.net
216.200.127.145         core1-core2-oc192.lga2.above.net
216.200.127.146         core2-core1-oc192.lga2.above.net
216.200.127.149          core1-core3-oc48.lga2.above.net
216.200.127.153     core1-main1colo45-oc48.lga2.above.net
216.200.127.157     core1-main2colo45-oc48.lga2.above.net
216.200.127.161    core1-main1colo678-oc48.lga2.above.net
216.200.127.165    core1-main2colo678-oc48.lga2.above.net
216.200.127.169          core2-core3-oc48.lga2.above.net
216.200.127.173     core2-main1colo45-oc48.lga2.above.net
216.200.127.177     core2-main2colo45-oc48.lga2.above.net
216.200.127.181    core2-main1colo678-oc48.lga2.above.net
216.200.127.185    core2-main2colo678-oc48.lga2.above.net
216.200.127.189          core1-main1-oc48.lga1.above.net
216.200.127.194          core1-main2-oc48.lga1.above.net
216.200.127.197          core2-main1-oc48.lga1.above.net
216.200.127.201          core2-main2-oc48.lga1.above.net
216.200.127.205             dfw2-dca2-oc48.dfw2.above.net
216.200.127.206             dca2-dfw2-oc48.dca2.above.net
216.200.127.209         core1-core2-oc192.dfw2.above.net
216.200.127.210         core2-core1-oc192.dfw2.above.net
216.200.127.213          core1-core3-oc48.dfw2.above.net
216.200.127.217          core2-core3-oc48.dfw2.above.net
216.200.127.225             atl2-dfw2-oc48.atl2.above.net
216.200.127.226            dfw2-atl2-oc48.dfw2.above.net
```

We appeared to be under attack by more than TWO HUNDRED of the Internet's core infrastructure routers.

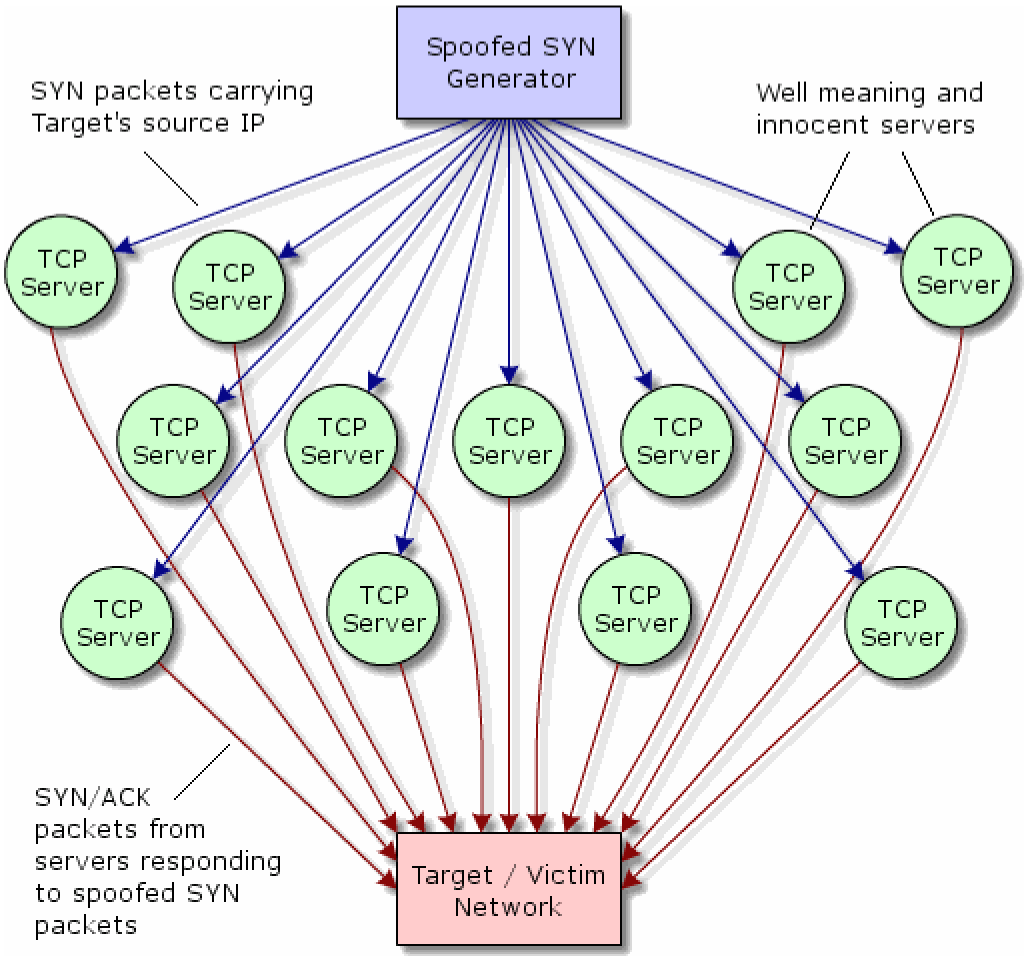
What was going on?

Looking at the individual packets flooding us from Verio, Qwest, and Above.net, I saw that they appeared to be completely legitimate SYN/ACK connection response packets showing a TCP source port of 179. In other words, just as a web server's packets will return from the HTTP port 80, these packets were returning from the "BGP" port 179.

BGP is the "Border Gateway Protocol" supported by intermediate routers. Routers use BGP to communicate with their immediate neighbors to exchange their "routing tables" in order to inform each other about which IP ranges the router can forward.

The details of BGP are unimportant. What **is** important is the fact that virtually all of the Internet's extremely well-connected (high bandwidth) intermediate routers will accept TCP connections on their port 179. In other words, a SYN packet arriving at port 179 of an Internet router will elicit a responding SYN/ACK packet.

I suddenly knew what must be happening . . .


Spoofed SYN Generator
SYN packets carrying Target's source IP
Well meaning and innocent servers
TCP Server
SYN/ACK packets from servers responding to spoofed SYN packets
Target / Victim Network

There was **no way** that all, or probably any, of those hundreds of routers had been compromised or infected by any sort of Zombie. I realized that they were just ordinary, innocent, TCP servers doing their jobs. They were sending SYN/ACK packets to grc.com in the well-meaning belief that WE wanted to open a TCP connection with their built-in BGP servers.

In other words, a malicious hacker located somewhere else on the Internet, was SYN FLOODING INTERNET ROUTERS with TCP connection-requesting SYN packets. Those SYN packets carried the fraudulent (spoofed) source IP belonging to grc.com. Therefore, the routers believed that the SYN packets were coming from us, and they were replying with SYN/ACK packets as the second phase of the standard TCP three-way connection handshake.

Malicious SYN packets were being "Reflected" off innocent bystanding TCP servers. Their SYN/ACK responses were being used to flood and attack our bandwidth.

A new type of attack

Fascinating as this was from a theoretical standpoint, grc.com was being actively blasted off the Internet by hundreds of Internet routers. Once forensic packet captures were underway, my priority was to get us back on the Net.

I have no idea why we were the target of this attack. Perhaps this was a test of a potent new weapon **. . .**

"Hey, let's blast Gibson and see what he does."

As we will see below, mounting evidence indicates that after the initial attack on us, one or more of these weapons has been in frequent use. As we will also see below, there are a number of reasons for malicious hackers to much prefer this "distributed reflection attack" mechanism over traditional distributed direct flooding approaches.

I didn't want to bother Verio, my service provider, if the attack was going to end by the time I had an engineer on the phone. So I watched the attack and waited for two hours. By 4:00 AM Pacific time, the attack showed no sign of letting up. The East coast of the United States would be waking up soon, so I made the call to Verio's 24/7 network trouble center. I summarized the problem and convinced the operator that we really were under attack and needed immediate engineering assistance. The problem ticket bounced around within Verio for two more hours and I was delighted when a sleepy-eyed Verio Security engineer returned my call.

Blocking the reflection attack

The good news was, this attack appeared to be relatively easy to block. Since we are not an Internet service provider ourselves, we never have any need to connect with remote BGP-equipped routers. Therefore, I asked Verio to block any inbound traffic originating from the BGP service port 179. Since the malicious hacker's SYN packets were aimed at the intermediate routers' port 179, any reflected packets would be originating from that port.

Verio's engineer added a "filter" to the aggregation router servicing our Internet connection to block (drop) any packets inbound to us from port 179. The flood of packets coming in from port 179 immediately stopped.

But we did NOT return to the Internet.

Whoops. A fresh packet capture revealed that we were **now** being actively flooded by an entirely new set of Internet servers. Since this second set of traffic appeared only after the port 179 router traffic had been blocked, it appeared that this second wave of reflection traffic had been unable to compete with the routers' flood. (You know you're in trouble when packet floods are competing to flood you.)

With the routers traffic blocked, we were now being flooded by a SYN/ACK packets pouring in from ports 22 (Secure Shell), 23 (Telnet), 53 (DNS), and 80 (HTTP/Web). There were also some packets coming from port 4001 (a proxy server port) and 6668 (IRC chat).

I am annoyed with myself because I was so surprised by this unexpected second tier of flooding packets that I failed to capture and retain a complete and accurate sampling of the non-BGP flooding traffic. However, one of my earlier capture logs did reveal some non-BGP SYN/ACK traffic that had slipped through during the original BGP flood. So I have some record of it:

A small sampling of the web servers that were flooding us with SYN/ACK traffic from their HTTP (web) port 80:

Source IP	Machine Name
64.152. 4. 80	www.wwfsuperstars.com
128.121.223.161	veriowebsites.com
131.103.248.119	www.cc.rapidsite.net

```
164.109. 18.251          whalenstoddard.com
171. 64. 14.238           www4.Stanford.EDU
205.205.134.  1       shell1.novalinktech.net
206.222.179.216             forsale.txic.net
208. 47.125. 33                gary7.nsa.gov
216. 34. 13.245    channelserver.namezero.com
216.111.239.132                 www.jeah.net
216.115.102. 75             w3.snv.yahoo.com
216.115.102. 76             w4.snv.yahoo.com
216.115.102. 77             w5.snv.yahoo.com
216.115.102. 78             w6.snv.yahoo.com
216.115.102. 79             w7.snv.yahoo.com
216.115.102. 80             w8.snv.yahoo.com
216.115.102. 82           w10.snv.yahoo.com
```

The partial list of port 80 SYN/ACK flooders above has a few interesting members. The servers are mostly a scattered collection, so it appears that the malicious hacker went around deliberately collecting the IP addresses of well known and presumably well-connected web servers for use in the packet reflection attack. The seven web servers belonging to YAHOO.COM are interesting, as is the one machine **gary7.nsa.gov**. ("Gary7" was the name of a human from the future who appeared in the original Star Trek series.)

The wide array of additional SYN/ACK flooding traffic, and the use of many Internet servers other than Internet routers, demonstrated that the malicious hackers were well aware that **any** general purpose TCP connection-accepting Internet server could be used as a packet reflection server. Once I saw that we needed to do more than simply block packets inbound from the BGP port 179, I developed a more comprehensive solution for this sort of attack. This is discussed below, after the analysis of the attack's probable construction and consequences.

Once our reflection attack filters were in place, we immediately popped back onto the Internet (after a relatively brief four-hour attack outage). Although I was unable to monitor the subsequent conclusion of the attack from behind our filters, it is somewhat chilling to note that **. . .**

By the time the attack ended, Verio's router had discarded more than one billion (1,072,519,399) malicious SYN / ACK packets.

I received this sobering number from Verio when I contacted them after realizing that I had failed to capture the second, non-BGP, wave of the attack. I wanted to reconfigure our defenses to deliberately put us back into denial of service so that I could collect that data. But by then the attack had stopped.

Reflecting on the reflection attack

Judging from the evidence collected during the 1/11 attack, and other subsequent evidence, someone, some group of people, or multiple groups of people, have accumulated — and, as we'll see below, are actively accumulating — a growing list of well-connected (high bandwidth) Internet servers and corresponding TCP service port numbers. Among these are several hundred routers serving the BGP (port 179) protocol and many other servers listening for connections on other common service ports including SSH, TELNET, DNS, HTTP, and IRC.

Building and maintaining the "reflection server" list

Since any publicly accessible Internet server qualifies as a "reflection server", a list of servers is easily generated, grown, and maintained. For example, the common Internet "Trace Route" command provides the IP address of every Internet router between the tracer and any other remote address — even a nonexistent address. As we have seen, there is a very good chance that an Internet router will be exposing its BGP server to the public, and that it will have a very high bandwidth connection. A simple script can be used to collect an arbitrarily large number of Internet router IPs. Huge web server farms, such as Yahoo.com, are all publicly available. Simple port scans through high-bandwidth IP regions will reveal thousands, if not millions, of publicly available waiting TCP servers. Any Internet search engine will produce hundreds of thousands of potential web site domains whose IP addresses can be looked up and cataloged.

The resulting massive list of "reflection servers" can be easily and continuously maintained by bouncing a valid, non-spoofed SYN packet off the machine. The answering SYN/ACK will confirm the machine's presence and its willingness to unwittingly participate in future reflection attacks.

Using the reflection server list

Any raw socket capable host (Unix, Linux, Windows 2000, and now the mass-market Windows XP) can trivially be used as a platform for the generation of reflection attack SYN floods. The number of SYN-generating hosts required for an attack would be determined by the total flooding bandwidth required to sufficiently bury the target host computer or network. (See "Bandwidth Multiplication" below for additional important details about this.)

Driven by a large list of innocent TCP packet reflection servers, each SYN spoofing host "**sprays**" its SYN packets evenly across every reflection server on its list. A fraudulent SYN packet would bounce off one of each server's open TCP ports, with the reflected packet aimed at the intended victim network.

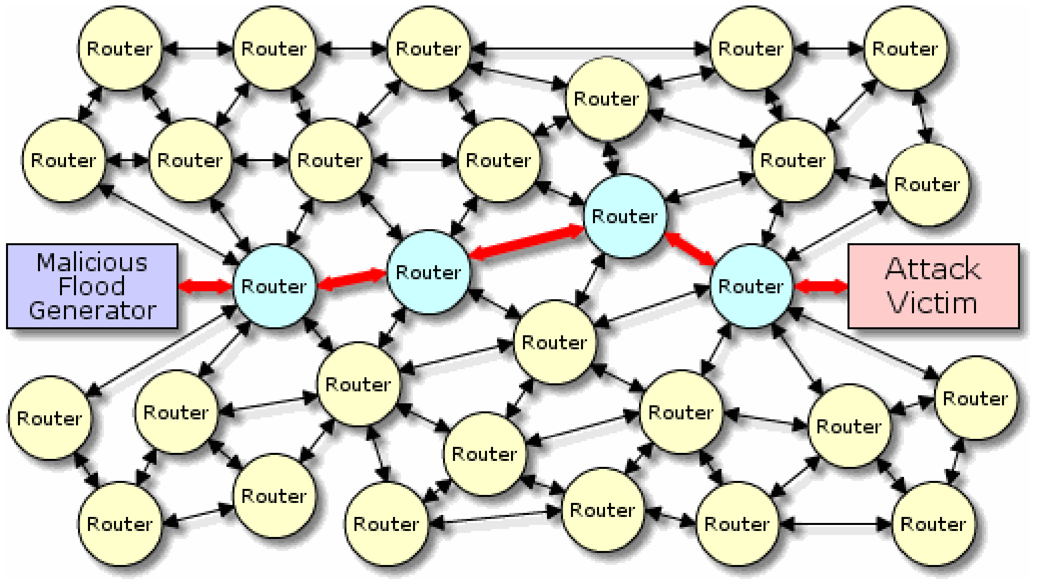
Since the SYN flooding machine is "spraying" its packets across a huge number of intermediate reflection servers, none of the servers participating in the attack will, themselves, be unduly inconvenienced by their participation in the attack. Each reflection server will experience a low-level "SYN flux" rather than a high-level SYN flood.

Why should we worry?

Why is a reflection attack superior to simply having the malicious hosts directly flood and attack their victim?

Packet path diffusion

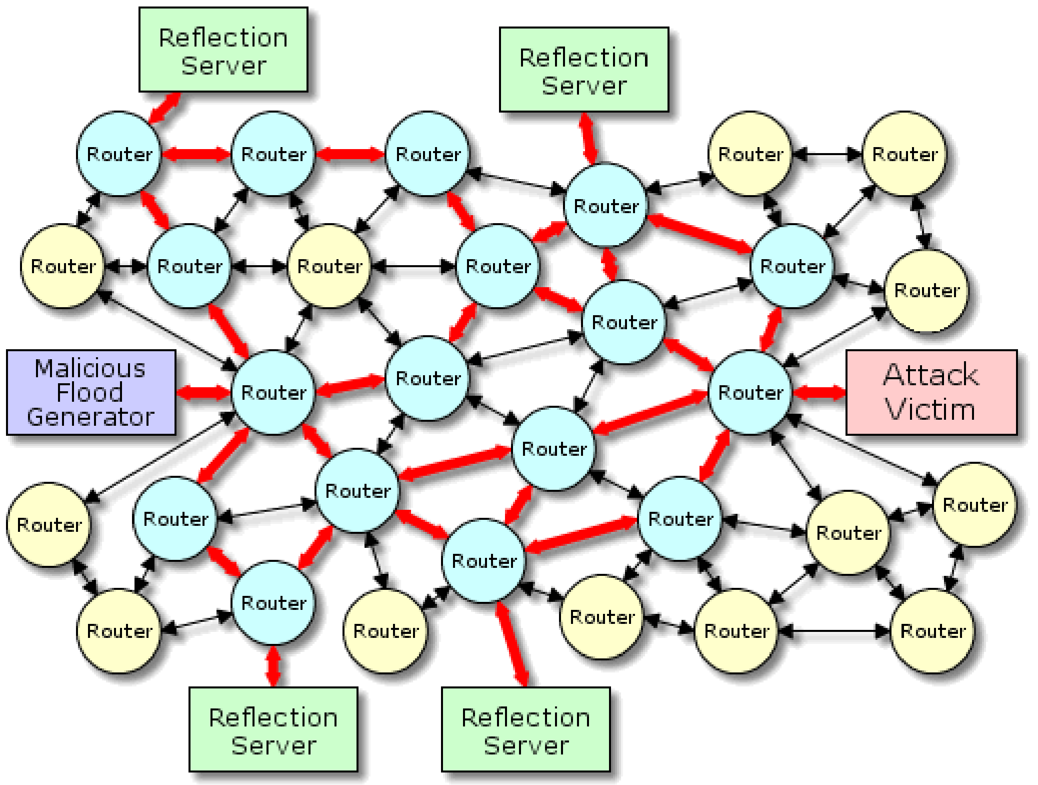
The big win for the attacker is the extreme degree of "packet path diffusion" made possible when attack traffic can be bounced off a large number of intermediate TCP servers. This diagram is a representation of the path of traffic between a single attacker and victim:



The Internet is essentially a large network of individual, interconnected, packet routers. The specific path taken by individual packets moving between any two Internet endpoints may change as a result of local network outages, congested routers, and other factors. However, over a short period of time, most packets will travel along the "best path" — which rarely changes from one packet to the next.

Since Internet routers do not retain any record of previously routed packets, "backtracking" an attack from the victim to the attacker, relies upon the feasibility of manually following a packet flood "upstream" from one router to the previous.

Diffusing the path: Even in the presence of a solid and powerful packet flow, packet path backtracking is a difficult and time consuming manual process. So, imagine what happens when a large number of widely spread packet refection servers are added to the system **. . .**



As this second traffic-routing diagram demonstrates, the addition of innocent reflection servers substantially transforms the attack. Upon leaving an attacking machine, the malicious SYN packets immediately fan out. No longer aimed at the victim, these attack packets are instead being sent to widely spread TCP servers. As we know, these servers are potentially located throughout the entire Internet. Just a few "router hops" away from the attacker, the heavy packet flow will no longer be discernible because it will have diffused into neighboring routers rather than following a single path.

The situation on the receiving end is also significantly changed. Rather than being assaulted by discrete packet flows, one from each attacking machine, the victim is now under a SYN/ACK flood from hundreds, thousands, or even millions of individual, innocent, servers. Although each of those packet flows would be individually harmless to the victim (just as each one is harmless to its individual reflection server), the convergence of these packets arriving from everywhere across the entire Internet, creates an attack that will swamp the victim.

What can the victim do?

What **can** the victim do? Faced with unwanted packet traffic flowing in from potentially hundreds of thousands or millions of servers, each one occasionally sending an individually valid SYN/ACK packet, the target network is in serious trouble. And **. . .**

With a bit more cleverness from the attacker, it gets worse still:

Reflector usage phasing

As if the situation outlined above were not already bad enough, the addition of a little more cleverness from the attacker can make things much worse. We have already seen how easily a vast inventory of well-connected Internet reflection servers can be obtained and managed. Armed with a sufficiently large reflection server inventory, a continuous and steady stream of flooding traffic can be focused upon the victim network **while cycling through a subset of the total reflection server inventory.** At any time, only a small portion of the total reflection server inventory would be in use.

This "reflector usage phasing" would have the effect of continuously changing all of the many attack paths, phasing them in and out of use, while never relenting upon the target network. Due to the manual nature of attack backtracking, following an individual trickle is much more difficult than following a flood. But nothing is more difficult than backtracking an intermittent trickle. In fact, with current router technology, it probably borders on impossible.

Reflector diffusion

Another significant consequence of the use of a large reflection server inventory is "reflector diffusion." With the attacker(s) outbound SYN packet flood being "sprayed" and spread out across a large number of reflection servers, no single reflection server will be receiving a troubling number of SYN packets. Since half-open connections will be discarded after a minute or two (after the server grows tired of resending unanswered SYN/ACK packets — see next section), the number of dangling, never completed, "half-open" TCP connections will never grow excessive. Since the server will simply see these as client-aborted connections — which can and do occur routinely between clients and servers — a low level of these maliciously generated reflections will probably never raise any automated alarms on most servers. Although an attentive administrator might wonder about a persistent collection of ever-changing half-open connections, it would probably be shrugged off since it would not be hurting the server in any way.

(As we will see below, a few attentive administrators HAVE already spotted what is very likely this abuse of their TCP servers.)

Bandwidth multiplication

A clever and significant characteristic of any TCP reflection attack is the emission of several times more SYN/ACK attack traffic from the reflection servers than the triggering SYN traffic they receive. Due to TCP's automatic lost packet re-sending, the reflection servers will generate several times more outbound SYN/ACK flooding traffic than they receive from the SYN generating hosts.

For every one SYN packet received by a TCP reflection server, up to four SYN/ACK packets will generally be sent. This will occur because the victim's aggregation router will be discarding the majority of the inbound flooding traffic. A reflection server that receives no reply to its SYN/ACK packet response will believe that its packet was lost in transit (as indeed it was, but in this case by design) and will resend the "lost" SYN/ACK packet several times (typically three retries) before giving up and discarding the aborted connection.

This phenomenon has the interesting consequence of spreading "attack intent" both physically across the Internet and temporally a few minutes into the future.

Improved manageability

Traditional DDoS attacks have used large networks of attacking machines for two purposes: To generate large aggregate flooding volume, and to diffuse the source of their attacks. For a given target, the combination of bandwidth multiplication and the extreme levels of path diffusion offered by distributed reflection attacks significantly reduces the number of hosts required to provide the required flooding bandwidth and sufficient backtracking path obfuscation. The result is an easier to assemble and maintain network of compromised attack hosts and much less opportunity for discovering their true location and nature.

Stealth

A stealth aspect of this attack architecture, is the lack of any generated "backscatter" traffic. Reflection attacks will not appear on the "radar screens" of organizations monitoring traditional spoofed source IP SYN floods and other similar "fraudulent IP" attacks. Unlike traditional spoofed source IP attacks, which typically generate source IPs at random, every IP occurring during the different aspects of a reflection attack refers to a valid machine — either a reflection server or the attack target. Consequently, this attack does not generate any monitorable attack "backscatter." (However, as noted below, "reflection honeypots" are easily established and monitored.)

For all of these reasons, distributed reflection denial of service attacks (DRDoS attacks) can be easily engineered and maintained. They are difficult or impossible to backtrack, and they are disturbingly potent.

Additional evidence

While I was assembling this page, a little over one month following the 1/11 reflection attack, a posting to the "Bugtraq" mailing list caught my eye:

> In the last few days I've been seeing what *looks* like a SYN flood attack on port 80 across all IP addresses on my network. However, if it's a flood, it's not a very strong one. Modest hardware is able to keep up with the incoming packets without a problem, but the steady flow of SYN packets is still a steady flow. (On a given system, the number of connections in a SYN_RECVD-ish state numbers 50-100.) The source IP addresses stay constant for a minute or two and then cease, sometimes as another IP address starts sending its own stream of SYN packets, though occasionally more than one host will be sending traffic at a time. Source addresses are in a variety of networks, but seem to be consistently dialup or similar type connections.
>
> It "feels" like an attempt at a denial-of-service attack, but why spread it out over so many destination IP addresses (many of which have no Internet presence), and why

> would the flood be so weak as not to actually affect anything? [...]

If you have been following this discussion, you'll notice that this is PRECISELY what a sharp administrator of reflection servers would see while their servers were participating in a reflection attack. He misinterpreted his evidence a bit, because he didn't have the advantage of knowing that his servers were being used as a reflector. He saw a gentle "flux" (not a flood) of SYN packets coming in from a given IP, which might suddenly change and appear to be sourced by another IP. We know this was due to a change of the attack's target.

From the perspective of the reflection server, the apparent source of those SYN packets was actually the target of the attack, to which his servers were sending a gentle "flux" of SYN/ACK packets. His outbound bandwidth to the victim was probably about four times more than the inbound SYN flux, as demonstrated by the fact that those connections were in their "SYN_RECVD" state. This means that they were waiting for the client's answer (which was unlikely to arrive) and were therefore resending SYN/ACKs, delivering attack bandwidth multiplication.

Upon seeing this posting, I immediately replied to the list with my interpretation of what he was seeing. Before my reply was broadcast to the mailing list some other sharp-eyed network administrators chimed-in:

> Yes, we are seeing the same thing over here... It appears to be most effective when the attack is pointed at a subnet with a shared web server with many IP's bound to the same interface. This also could be an attempt to use these system's as a reflector to flood a particular IP address out on the web...

> We saw a similar event a few days ago. Turned out to be an attack against an IRC server company. Forged source address, and used our public web addresses as reflectors. Took the IRC server company off the air. Whomever was the source sprayed at least a class B with this stuff.

> Yes, we see them here too. It's all very strange.
>
> We started noticing them in early February, but a check of our raw header logs shows some activity as early as January 15. (That's the oldest log left)
>
> We've noticed that they seem to be coming from a limited range of IP's, maybe about a dozen. Many of them seem to be coming from universities. I've notified several of them and received feedback that they would block the IP's and investigate further.
>
> We also see the sequence numbers being identical for a large number of packets. We saw traffic from one IP in Korea that sent over 1,000,000 packets in a one-hour period, but most are coming in at a rate of 10,000-30,000 per hour.
>
> We also notice one incident where the source and destination ports were both #23.
>
> We've called it the "Stuttering SYN" attack. Your observation is the first that I've seen, and I've been looking for about a week now. It's gratifying that others are seeing it too.

Remember that since this administrator was seeing reflection attack SYN traffic, the reference to *"traffic coming from a limited range of IP's, maybe about a dozen. Many of them seem to be coming from universities."* actually meant that these were the targets of the reflection attack to which his servers were unwittingly contributing.

> I have two web servers on different networks that have been receiving this type of traffic for the last 2 or 3 weeks. The same source IP's hit both hosts at about the same time. This is low rate traffic and generates ACK's back to the target. I have been logging this activity for about two weeks and have captured some of the packets. I suspect that more than one machine have the same reflector host list based on the varying times of day when activity occurs. [...]

That's great feedback. The fact that two different web servers located on different networks were simultaneously receiving SYN packets from the same spoofed source IP indicates that machines within different IP address regions were both on the same "reflection server inventory" and were innocently participating in the same attack.

> I noticed this traffic on my machine last november, it wasn't until a few weeks ago that I had time figure out it was some sort of SYN flood. I'm glad someone finally mentioned this, as I thought I had pissed someone off. :)

So we have some evidence that tools to perpetrate these sorts of attacks may have started appearing at least as early as November, 2001. We won't have any real sense for how prevalent and widely spread these reflection attacks are until administrators start looking for the telltale signs of low-rate SYN floods into their servers, or until victims of SYN/ACK floods make their attacks known.

A "reflection attack honeypot"

A "honeypot" is the term used within the security industry to refer to a machine that has been deliberately set up to "sting" an attacker by giving them something sweet to attack.

A recent correspondent of mine, who was interested in this sort of attack, established a network to appear as a large block of web servers on a never before used IP space. The next day, this newly created network of web servers had been "discovered" by a scan across their IP space. Before long, those machines were participating in various distributed reflection denial of service attacks.

For a thorough description and "how-to" for a virtual Linux honeypot, see this off-site link: Click Here

Reflection attack defense & prevention

There's conditional good news, and some rather bad news.

As we saw near the top of this page, machines communicating over the Internet can generally be divided into the roles of client and server. These roles may change depending upon circumstance (a web serving machine might be a client of an eMail server), but most individual TCP connections generally imply a client and server relationship. Clients generally initiate connections from high-numbered (client) ports to servers that are listening for incoming client connections on low-numbered (service) ports.

Since any reflected SYN/ACK packets **must** have bounced off a TCP server, and since almost all common service ports fall within the range from 1 to 1023, blocking all inbound packets originating from the service port range will block most of the traffic being innocently generated by reflection servers. However, there are several problems with doing this **. . .**

Protecting a server

First, the attack against grc.com contained SYN/ACK packets from ports 4001 and 6668. So those (and perhaps other) high-numbered service port exceptions might also need to be blocked if their SYN/ACK traffic was sufficiently high. The problem with blocking inbound traffic from high-numbered ports is that legitimate clients of the protected server could be generating connections from those blocked ports. The reflection attack filters would block those packets and prevent the valid client connections.

The second problem with simply blocking all inbound traffic originating from ports below 1024, is that a server behind such a blanket filter would be unable to transact with other servers when acting as their clients. The example above, of a web server acting as a client of an SMTP (eMail) server, would be complicated because the remote SMTP server's packets, attempting to return from the SMTP server's port 25, would be blocked by the reflection attack filter. To get around this problem, exception "holes" would need to be created through the reflection filter to allow "friendly" remote server traffic to pass through the filter.

Protecting a client

Here's the rather bad news. Client-profile machines, like that of the typical end user, can not be protected. Since most clients spend all of their time connecting to remote servers all over the Internet — to the very servers that might be inadvertently attacking them — they require access to data coming back from many of the most common low-numbered service ports.

Consequently, a characteristic of reflection attacks is that no sort of upstream filtering, short of full inbound connection proxying or ISP-resident NAT routing, can protect users who require access to remote servers.

Preventing reflection server exploitation

I can imagine that the savvy network administrators who posted to the Bugtraq mailing list might have felt some sense of chagrin when they realized that their servers or server farms were, even innocently, involved in adding a bit of their bandwidth to the task of burying someone else's bandwidth. This raises the question of whether there's anything "the man in the middle" (or the server in the middle) can do to prevent the use of their servers for reflection attacks.

In theory, it's simple: The trick to doing this would be to recognize a SYN source IP that never completes its connections. Since a number of failed connections occurring within a short time span would be highly unusual, the target of any reflection attack could be readily determined. Dynamic reflection attack prevention would simply "black list" any SYN packets arriving from any such IPs until none had been received for some period of time, perhaps an hour.

A reflection server exploitation prevention system could be easily built into a server-resident firewall application. However, expecting (or relying upon) every server on the Internet to be running such an altruistic application is probably unrealistic. Asking, or requiring, ISP's to provide spoofed packet network egress filtering would seem to be far more feasible **. . .**

The ISP's responsibility

The generation of traffic for a reflection attack depends upon source IP address spoofing. If ISPs would begin adopting the practice of preventing the escape of fraudulently addressed packets from within their controlled networks, this potent attack, and its many cousins, would die overnight. In addition to being the right thing to do by helping to prevent abuses by their customers upon those outside the network, egress filtering also enhances the security for an ISP's own customers because malicious hackers would soon learn that their spoofing attack tools would not function within an egress filtered ISP network.

The attacking platform's responsibility

I imagine that anyone reading this page is already well aware of my feelings regarding the deliberate and unnecessary inclusion of the raw socket API in a mass market consumer desktop PC. I am referring, of course, to the absolute insanity of Microsoft's inclusion — and subsequent defense of — the raw socket API in Windows XP.

While pedantic network experts, and Microsoft themselves, correctly argue that there are other ways to produce malicious Internet traffic, there is no easier way than through the use of raw sockets. The best way to earn users' trust is to deserve it. But deliberately incorporating this unnecessary facility into every Windows XP machine — and essentially enabling it, by design, to become a malicious reflection attack generator — makes a mockery of Microsoft's recent "Trustworthy Computing" rhetoric. We can always hope, as I fervently do, that Microsoft will recognize that it is not too late, and will remove raw sockets from XP during one of the product's continuous flow of patches and Windows Updates.

History and future

Most Internet attacks exploit fundamental and original design weaknesses in the Internet's protocols, or in the operation of its many infrastructure components. Consequently, "reflection attacks" of many sorts have always been possible. Many have been known and used before, and they are not generally new. The simple TCP reflection attack discussed on this page is no exception.

The crucial fact is that the world is changing rapidly and the world's Internet of today and tomorrow is not the Internet of yesterday. As business and commerce plunge headlong onto the Internet, as the cost of Internet-bandwidth and connected machines plummets, and as the techniques for assaulting the Internet's growing resources percolate throughout the huge "script kiddie" population, that which was once old, becomes new again.

New motivations, new capabilities, new victims, and new freely available tools are resulting in the irresponsible assault and exploitation of an old network technology that was never designed for the future it faces. We ignore these realities at our own peril.

In conclusion

If you have managed to read this entire page, you will have a strong grasp of the essential facts and operation of this significantly worrisome and apparently increasingly prevalent "Distributed Reflection Denial of Service" (DRDoS) Internet attack.

If you started into this page without a detailed understanding of the functioning of the Internet, the operation of the TCP protocol, and the relationship of clients, servers, and port numbers, I hope you have had an enjoyable and edifying read. This page was created for you.

All the best,

Steve

Last Edit: Feb 22, 2002 at 09:41 (0.03 days ago) Viewed <too new> times per day

Home | Purchasing | Tech Support | Mailing List | Projects | Free Stuff | Discussions

The contents of this page are Copyright (c) 2002 by Gibson Research Corporation.