Remote Denial of Service Attacks and Countermeasures

David Karig and Ruby Lee

Princeton University Department of Electrical Engineering Technical Report CE-L2001-002, October 2001.

1. Introduction

As evinced by a series of high profile attacks, denial of service (DoS), or prevention of legitimate access to resources, is a threat that demands attention. Of particular concern are distributed attacks, in which an adversary recruits several computers to aid in the attack. The first major distributed denial of service (DDoS) attack brought down the University of Minnesota's network for three days in August 1999. About six months later, the attack by a Canadian teenager on several major sights including Yahoo, Amazon, eBay, CNN, and Buy.com made headlines. In accord with the underlying principle that destruction is simpler than construction, denial of service attacks come in many forms and are easy to carry out, but preventing them can sometimes be tricky. Although there is no panacea for all flavors of denial of service, there are several countermeasures that focus on either making the attacks more difficult or on making the attacker accountable via logging and tracing.

2. Background

A basic understanding of the features of common protocols, DNS, ARP, and buffer overflow are necessary. Readers already familiar with these subjects may proceed to Section 3.

2.1 Network Layers

The International Standards Organization (ISO) devised a standardized model for networking protocols referred to as the OSI (Open Systems Interconnection) model. The lowest level is the physical level, which is responsible for delivering bits through the network. The next layer, the data link layer, aggregates these bits into frames and thus defines the format of data on the network. The third layer is the network layer, which handles routing, the directing of datagrams from one node to another. Next is the transport layer, which divides user-buffer-sized datagrams into network-buffer-sized datagrams and enforces desired transmission control, such sequencing and retransmission. The session layer, to which protocols such as RPC belong, defines a control structure for remote communication sessions. The presentation layer handles the conversion between the local form of representing data and the canonical form defined for the network. Finally, the application layer includes protocols such as HTTP and FTP and provides network services to end-users [16].

2.2 IP, UDP, ICMP, and TCP Basics

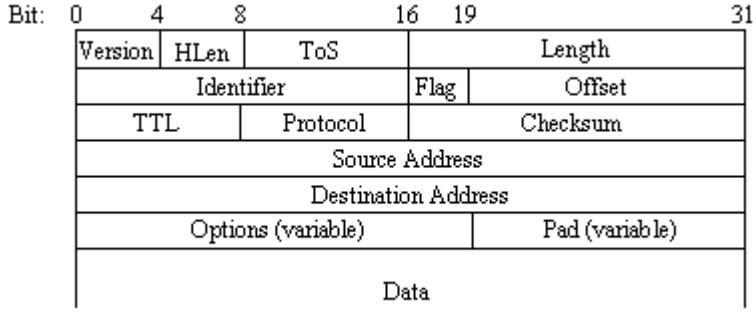
The Internet Protocol (IP) is a network layer entity that provides a means for hosts on different networks to communicate, regardless of the fact that the types of networks may be different. The format of an IP packet is shown in Figure 1. The Version field holds the version of IP used, which for the purposes of this paper shall be assumed to be 4. The Hlen field specifies the length of the header in 32-bit words. ToS, or type of service, indicates how packets

should be treated based on application needs (how they should be queued, etc.). Since the Length field, which specifies the length of the entire datagram in bytes, is 16 bits, the maximum length of an IP packet is $2^{16} - 1$ bytes [19].

The Identifier field, the Offset field, and one of the flags are used for handling fragmentation and reassembly. Different networks have different MTU's (maximum transmission units), meaning that maximum allowable packet sizes differ from network to network. Thus, a large packet from a network with a large MTU may need to be broken into fragments as it enters a network with a smaller MTU. The end host is responsible for reassembling the fragments. Fragments that will be reassembled into a single packet are given the same identifier. The first fragment has an offset of zero, and for the following fragments, the Offset field contains the offset in bytes from the first fragment. A flag is set in a packet to indicate that more fragments will follow [19].

Time to live (TTL) provides a way to prevent packets from floating around indefinitely in transit to their destinations. At each hop, this field is typically decremented, and the packet is discarded if the TTL field reaches zero. Since several protocols such as TCP and UDP ride on top of IP, the Protocol field is used to specify the higher level protocol to which the IP packet belongs. A checksum is performed to provide a means for determining whether or not the header has been corrupted. The source address and destination address of the packet are specified following the checksum. Options, which are not frequently used, are included at the end of the header. The Options field is of variable length, and the length is determined from the Hlen field. Padding is added to make the header align on the 32-bit boundary. Following the header is the data payload [19].

Figure 1 – IP Packet Format

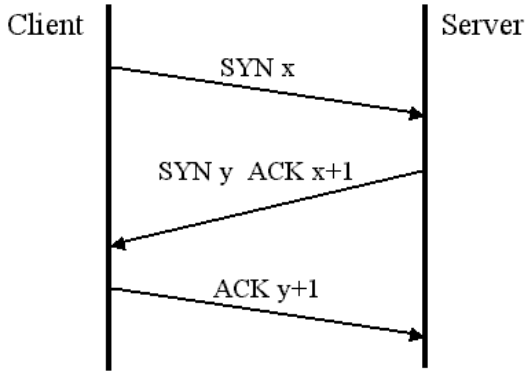


The User Datagram Protocol (UDP) is a simple, connectionless protocol. The header simple consists of a source port field, a destination port field, a header checksum, and a length field that specifies the length of the entire packet in bytes. Port numbers are used to determine the application to which the packet data should be directed. Like UDP, the Internet Control Message Protocol (ICMP) is connectionless. This protocol is commonly used for troubleshooting and error reporting.

By contrast, TCP is a reliable, connection-oriented protocol, and a three-way handshake is used to establish a connection. A client initiates a connection to a server by sending a SYN (one of the six control flags in TCP) and a sequence number (x in Figure 2). The server responds with a packet that has both the SYN flag and the ACK flag raised. This packet has its own sequence number y, and the acknowledgement field of the packet header specifies ($x + 1$), the

next expected sequence number from the client. If the client then responds with an ACK, the connection is established. Since packets can reach the destination out of order, the 32-bit sequence numbers are necessary to tell the destination how to organize packets once they are received. The initial sequence number *y* of a server is generally incremented each time a new connection is made and is also increased as time progresses [21].

Figure 2 – Three-way Connection Handshake



2.3 Domain Name System

The Domain Name System provides the service of resolving human memorable host names into numerical IP addresses and vice-versa. DNS implements a hierarchical tree structure. Directly under the root node in this tree structure are labels such as com, org, mil, gov, etc. Names continue to get increasingly specific with each level. The three major components of DNS are the distributed database, name servers, and clients. The database consists of the Domain Name Space and Resource Records (RRs), which define the domain names within the Domain Name Space. Name servers have zones of authority within the Domain Name Space and are responsible for maintaining RRs for some portion of the Domain Name Space and for servicing client queries. Each zone may actually have multiple name servers, but only one serves as a primary server, where actual changes to the data for a zone take place. Resource Records have the following fields: NAME, TYPE, CLASS, TTL, RD Length, and RDATA. The DNS name to which the RR belongs is specified in the NAME field. The TYPE field specifies the type of RR, and the three types most pertinent to the discussion in this paper are record types A (address records), NS (name server records), and CNAME (canonical name records). For this discussion, CLASS is IN, which stands for Internet. Name servers can cache RRs, and TTL (time to live) indicates the length of time for which a cached RR is valid. RD Length specifies the length of the RDATA field, which in turn holds resource data for a given NAME entry [4].

DNS has a message protocol to handle client queries and name server responses, and this protocol runs over UPD. Clients are configured with a list of a few name servers, out of which one is chosen to query. In the context of resolving host names to addresses, the name server may contact other name servers using NS record entries until one of the servers finds a matching address record entry. This is referred to as recursive requesting. Once the query is resolved, the answer is passed back through the name servers to the client. The name servers that did not have the matching address entry may cache the answer [4].

2.4 Buffer Overflow

One extremely common security hole is buffer overflow vulnerability. A buffer overflow vulnerability can be exploited by attackers to crash processes or possibly to break into a system. Stack overflows are a particularly rampant security problem. A simple example of a stack overflow vulnerability in C is shown below:

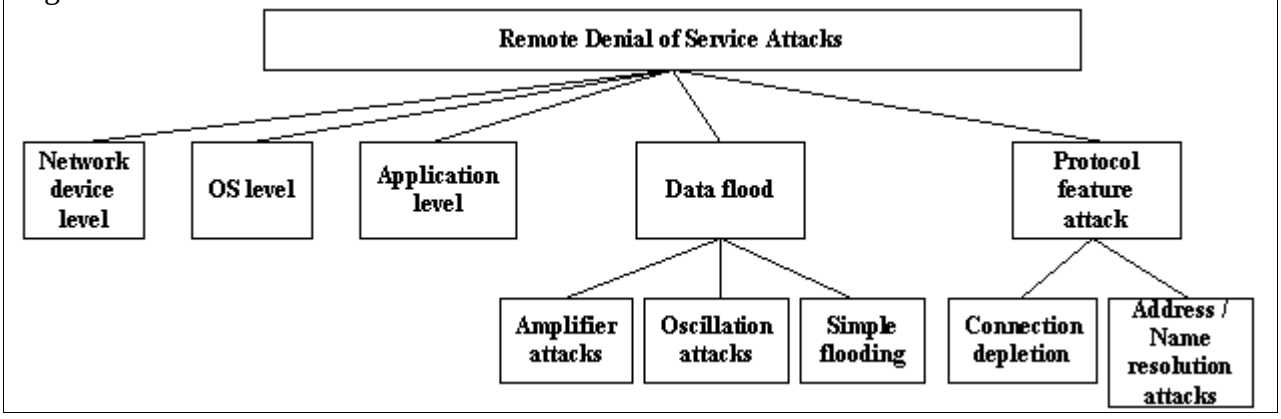
```
void function(void)  {
    char buffer[16];
    gets(buffer);
}
```

When the local function is called, state data including the return address is pushed onto the stack, and space is allocated on the stack for the local variable `buffer`. The problem is that gets() will read in any number of characters from the user input without regard to the amount of allocated space. If a long string is input, then the string will extend beyond its allocated space and overwrite the state data. A malicious user could enter a string that consists of a certain amount of padding, exploit code, and an address that overwrites the return address of gets(). This new return address would point to exploit code so that, once the function finishes executing, the exploit code will be run. If the program with the buffer overflow vulnerability runs with root privileges, then this exploit code would also be run with root privileges. Alternatively, instead of trying to run exploit code, the malicious user could input a long random string in an attempt to cause processes to crash.

3. Classification of Remote Denial of Service Attacks

Due to the wide variety of attacks, it is helpful to classify them in order to clarify the process of defending against DoS. Attacks can take advantage of bugs or software weaknesses of routers and other network devices. In addition, vulnerabilities in the way operating systems implement protocols as well as in applications running on the victim machines may be exploited. Alternatively, attackers may bombard victims with data, so that either little bandwidth is available to legitimate users or hardware resources of hosts or network devices are tied up in trying to process all of the data. Certain standard features of protocols can be taken advantage of in attacks as well. These categories of attacks are illustrated in Figure 3.

Figure 3 – Classification of Remote Denial of Service Attacks



3.1 Network Device Level

Suppose a corporate end network is supplied access to the Internet through three or so routers. If all of these routers are the same model, and if an exploit or weakness is found in this model, an attacker would be capable of denying the corporate network Internet access. One example of a network device exploit is Ascend Kill II. Ascend routers provided a GUI tool that allowed network administrators to remotely reconfigure the routers. This tool was capable of locating other Ascend routers on the network by broadcasting a specially formatted UDP packet on port 9, to which other Ascend routers would respond. However, Ascend routers crashed upon receipt of a "magic packet," a specially formatted UDP port 9 probe packet [23]. Another example is that, due to a buffer overrun error in the password checking routine, certain Cisco 7xx routers could be crashed by connecting to the routers via telnet and entering extremely long passwords.

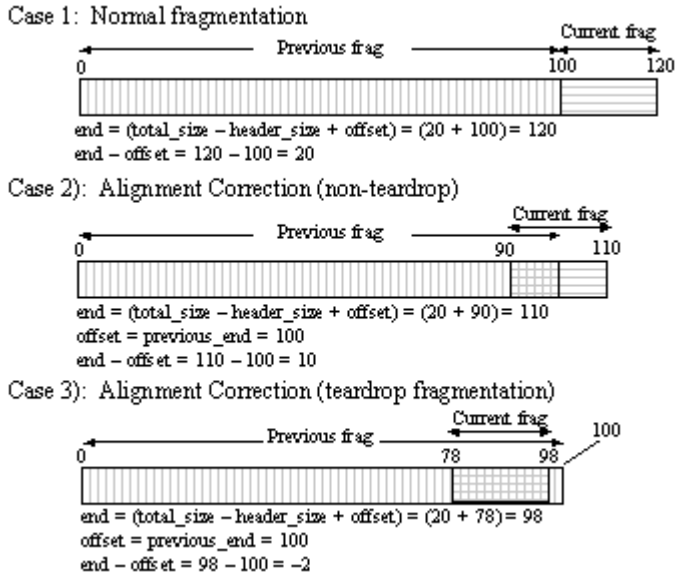
Rather than taking advantage of bugs or weaknesses in software, an attacker may try to exhaust the hardware resources of network devices. For instance, if a router or firewall runs a certain set of packet filtering and logging rules, an attacker could bombard the device with packets that the router must filter out and log in an attempt to bog down the device. A good, properly configured router, however, should be able to handle such an attack. An old form of attack that nowadays is a comparatively weak form of DoS involves bombarding routers with small packets having all options activated. These packets are sometimes called "Christmas tree packets," "kamikaze packets," "nastygrams", and "lamp test segments." Most attacks of this nature can be classified as data floods, which will be discussed later.

3.2 OS Level

Many attacks take advantage of the ways operating systems implement protocols. One example is the Ping of Death attack. In this attack, ICMP echo requests (pings) having total data sizes greater than the maximum IP standard size (2^{16}-1 bytes) are sent to the targeted victim. When such an oversized ping is sent, it is fragmented and later reassembled at the destination host. However, before OS vendors released patches for Ping of Death, this attack often had the effect of crashing the victim's machine. Many operating systems failed to allocate enough memory for oversized reassembled ICMP packets, and the buffer was overrun [18].

Teardrop, one of several attacks based on packet fragmentation and reassembly, is another example of how attackers can take advantage of the way certain operating systems implement protocols. Many operating systems were vulnerable to this attack, but the following explanation is derived from route|daemon9's description from the Linux perspective [22]. Upon receipt of a fragment, the OS calculates an *end* pointer by adding total size of the fragment to the offset and subtracting the length of the header. An *offset* pointer is normally set to the offset of the current fragment. However, if the offset of the fragment points inside the previous fragment, the OS tries to perform alignment correction and sets the pointer to the end pointer of the previous fragment. The fragment is copied into a new buffer of size *end – offset*. This procedure is illustrated in the figure below. Essentially, the length of the portion of the current fragment that "sticks out" from the previous determines the size argument in the memcpy() function used to copy the fragment into a new buffer. In the case of teardrop fragmentation, this is a negative value, which is treated as a huge positive value. This can cause the OS to crash upon receipt of a few teardrop-fragmented packets.

Figure 4 – Memory Allocation for Normal vs. Teardrop Fragmentation



3.3 Application-based Attacks

A vast number of attacks render a particular service or an entire machine inoperable by either taking advantage of bugs in network applications running on the target host or by using such applications to exhaust resources. One example of an application-based attack is the finger bomb. If the argument to finger is `username@host1@host2@…@hostn`, the finger request will be sent to host*n* and forwarded through all specified hosts from right to left until hostname1 is reached. This is referred to as redirection. A malicious user could specify `username@hostname1@hostname1@…@hostname1` as the argument, which would cause the finger routine to be recursively executed on the hostname, potentially exhausting the resources of the host [12]. Another example of a possible application-based attack involves the Windows NT version of RealServer G2 6.0. This version of RealServer has a buffer overflow problem in the username/password checking routine. By entering long username/password pairs, an attacker could possibly crash a machine running RealServer [20].

3.4 Data Flooding

By bombarding a target with massive quantities of data, an attacker may attempt to either use up the bandwidth available to a network, host, or device, or to bog down the resources of a host or device by causing it to process exceptionally large amounts of data. Three categories of this type of attack are amplification attacks, oscillation attacks, and simple flooding. The most commonly known type of amplifier attack is the smurf attack. In a smurf attack, the attacker forges the source address of a ping packet to that of the victim's host and specifies a broadcast address (x.x.x.0 or x.x.x.255) as the destination. The network that the ping request is sent to is referred to as the "bounce" or "amplifier" network, since all computers on the network send a ping reply to the victim. Thus, the victim is flooded with pings from every computer on the amplifier network.

A common type of oscillation attack makes use of the UDP chargen and echo services. When a packet is sent to a port that the chargen service is running on, which is usually port 19, the server generates a string of characters in reply. The echo service simply returns back data that is sent to the echo port, which is normally port 7. An attacker could send a packet to the chargen port of a host with the source port set to 7 and the source address spoofed to the address of a machine offering the echo service. Thus, data will be constantly bounced back and forth between the host responding to the chargen requests and the host echoing back the chargen strings [25].

An attacker could attempt to use up the available bandwidth of a network by simply bombarding the targeted victim with normal, but meaningless packets with spoofed source addresses. An example is flood pinging. Some ping tools have a "flood ping" option, which sends ping packets at the maximum rate that the server can receive them. By sending flood pings from multiple hosts, an attacker can choke out legitimate traffic or possible exhaust server resources. Two factors make simple flooding one of the most difficult forms of DoS to prevent. One factor is the ease by which IP addresses may be spoofed. The other factor is that many Internet services are meant to be available to anonymous users, and simple flood packets can be difficult to distinguish from legitimate traffic, aside from the fact that the IP addresses may be spoofed. Simple flooding is commonly seen in the form of distributed denial of service attacks, which will be discussed later.

3.5 Attacks based on protocol features

Denial of service attacks may take advantage of certain standard protocol features. Several attacks capitalize on the fact that IP source addresses can be spoofed. In addition, connection depletion attacks take advantage of the fact that many connection-oriented protocols require servers to maintain state information after a connection request is made but before the connection is fully established. The most common connection depletion attack is SYN flooding. If a client sends a SYN to a server but never completes the third step of the handshake, the "half-open" connection occupies some of the host's memory until the connection times out, which is often 75 seconds. Since half-open connections do take up some memory, operating systems must place a limit on the number of half-open connections. This limit is usually referred to as the backlog. A TCP SYN flood attack consists of an adversary sending SYN packets to a victim without ever completing the connections. The source addresses of the packets are spoofed to addresses of unreachable hosts. Thus, the victim responds with SYN/ACKs, but since the victim is sending these packets to an unreachable host, they are silently dropped, and the connections are left half open. The idea is to keep the backlog full so that no one can connect to the victim. In order to make such attacks harder to recognize, the SYN's are usually sent to random ports, and the spoofed addresses are varied. The reason that the addresses must be spoofed to addresses of unreachable hosts is that if the hosts were reachable, they would be able respond to the victim's SYN/ACK's. Since these hosts didn't actually send out the SYN's, their responses to the SYN/ACK's would be RST (reset) signals, which cause the server to tear down the connections [21].

Several types of past denial of service attacks have focused on DNS, and many of these attacks involved poisoning DNS cache on name servers. One problem is that earlier versions of BIND, the most commonly used implementation of DNS, did not check to see if the responses to queries were truly answers to the query questions. A rogue or broken name server could respond to a query with bogus information, which could be cached on the name server receiving the

response to the query. An attacker who owns a name server may coerce a victim name server into caching false records by querying the victim about the attacker's own site. A vulnerable victim name server would then refer to the rogue server and cache the answer [4].

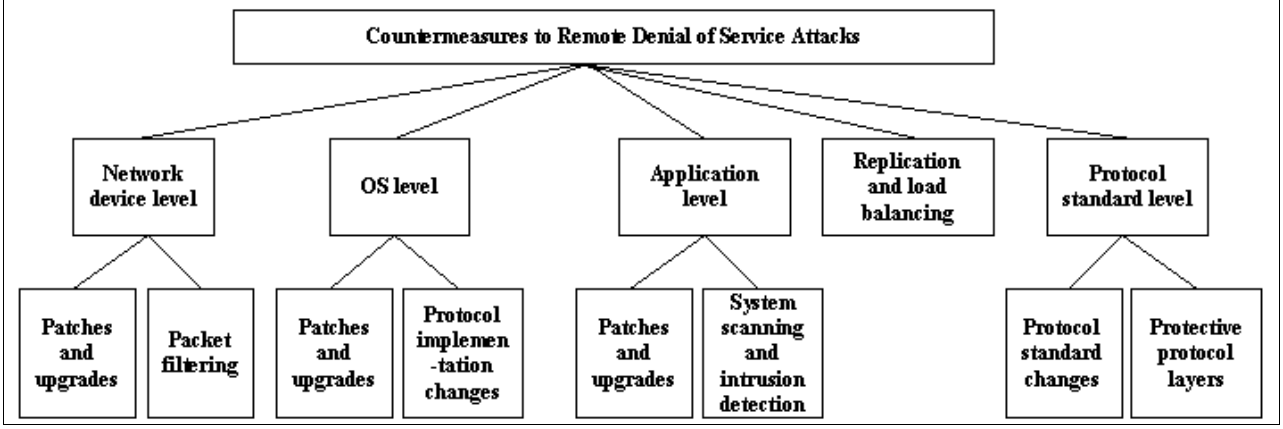
A similar form of DNS attack, which does not require the attacker to be able to directly place bad entries in the local name server, involves spoofing DNS responses. The only challenge to spoofing DNS responses involves predicting the Query ID, which is a number set by the query initiator in order to distinguish responses to different queries that may have been sent out within the same general frame of time. Suppose an attacking host (attacker.somenetwork.com) wants to cause the name server ns.target.com to return a bogus IP (33.33.33.33) when users query this targeted name server asking to resolve the IP address of blah.denied.com. The attacker must first check to see if an entry for blah.denied.com is already cached on ns.target.com. This is done by sending a non-recursive query to ns.target.com asking it to resolve the IP address of blah.denied.com. If it is cached already, the attack will not succeed. If it is not cached, the attacker proceeds to sniff packets sent to ns.somenetwork.com, the attacker's local network. The next step is to query ns.target.com asking it to resolve nonexistent.somenetwork.com, a host that does not exist on the attacker's network. The target name server thus queries ns.somenetwork.com, and the sniffer can then pick up the Query ID. If the Query ID is incremented sequentially, it should be easy to predict future query ID's within a limited time frame. Next, a query is sent to ns.target.com asking it resolve blah.denied.com, and the attacker immediately spoofs a response from ns.denied.com. If the Query ID is successfully predicted, and if the spoofed response arrives before the authentic one, ns.target.com will cache the incorrect entry that maps 33.33.33.33 to blah.denied.com [14].

4. Countermeasures

Countermeasures can be classified in a manner that mostly reflects the classification of attacks. Whether at the network device level, OS level, or application level, many attacks based on software exploits or bugs can be solved with simple patches and upgrades. In addition, routers can be used to filter out packets in order to hinder IP spoofing as well as a few specific DoS attacks. At the OS level, different protocol implementation features may be added or changed to deter attacks. Intrusion detection software can be used to detect certain known hacking activities based on fingerprints. System scanning applications that operate like anti-virus software may be used in hopes of detecting malware installed on a breached system that could be used to launch attacks. In addition, new security features may be added to protocols, or a protocol layer may be used to verify the legitimacy of users before allowing execution of protocols susceptible to DoS.

As shown in Figure 5, patches and upgrades can resolve many of the denial of service problems that are due to software vulnerabilities at the network device level, the OS level, and the application level. Fixing buffer overflow problems resolves the Cisco 7xx router, Ping of Death, and WinNT RealServer G2 6.0 problems. The Ascend Kill problem was eventually fixed in an upgrade, and bounds checking on memory copies can prevent teardrop fragmentation from crashing operating systems. A few problems, such as the finger bomb attack, are difficult to fix without slightly modifying the service. One possibility is to disallow redirection. Another is to have the finger tool ignore finger arguments that are not deemed as reasonable. Specifically, finger would not process an argument containing several references to the same host (i.e. `finger host1@host1@host1@...` or `finger host1@host2@host1@host2@...`).

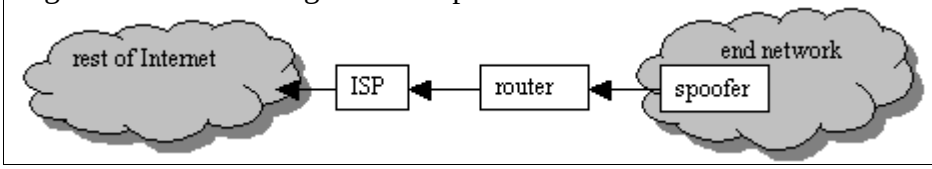
Figure 5 – Classification of Denial of Service Countermeasures



4.1 Network Device Level

Besides patches and updates to fix known software problems, packet filtering is a common DoS countermeasure. For example, to hinder smurf attacks, it may be wise for a network to either disallow broadcast ping requests to enter the network from outside, or to set up a firewall that rejects all incoming echo request packets. Also, the use of IP address spoofing is a common thread to the majority of DoS attacks. Ingress and egress filtering can be used to help keep spoofed packets from reaching their destination. Suppose that the ISP in Figure 6 owns the router shown that provides the end network with Internet access and that the ISP knows the valid address space of packets trying to enter through this router. If a packet is trying to enter the ISP and purports to be from an address that is outside of this valid address space, then it is clearly spoofed. This packet can be filtered out, and the spoof attempt can be logged [11].

Figure 6 – Network Ingress Example



Egress filtering involves the same concept from the end network's prospective. With egress filtering, the network's border router checks packets as they leave the network. Packets purporting to come from addresses outside of the network's address space are filtered out and logged. This makes the network less appealing for hackers to use as a base from which to launch attacks. However, this method does not prevent someone from spoofing an address to a different address within the network's address space. Ingress and egress filtering cannot be used in all networks. For example, a major ISP cannot use egress filtering because of its need to forward traffic that is not part of its own immediate addressing space [10].

4.2 OS Level

Connection depletion is particularly dangerous because of its asymmetric nature, meaning that an attacker with a low bandwidth connection disable a much more powerful

machine with a faster connection. One OS protocol implementation solution to TCP SYN flooding is the use of SYN cookies, an idea developed by Dan Bernstein and Eric Schenk. The underlying concept is to avoid allocating resources until the connection is complete, i.e. when the server has received the client's ACK. SYN cookies are based on the observation that most of the state that is normally stored upon the reception of a SYN request can be gleaned from the last ACK. The main exception is the MSS option, which must be encoded in the cookie. In the SYN cookie approach, when the backlog is filled, the initial server sequence number *y* is set as a cookie. There are various possibilities for calculating the cookie, one of which is shown below:

$$z = (t \,\%\, 2^5)(2^{27}) + (m)(2^{24}) + \mathrm{MD5}_{24\ \mathrm{bits}}(\text{secret}, t, \text{source addr., source port, dest. addr., dest. port, secret})$$

$$y = x + z$$

An intermediate value *z* is calculated. The five most significant bits consist of the lower five bits of a 32 bit counter *t* that increases every minute. This prevents the initial sequence numbers from increasing too quickly. The next three bits are an encoding of the MSS. The lower 24 bits are a secure, secret hash (such as MD5) of the counter value and the source and destination addresses and ports. This introduces randomness into the cookie, making it difficult for an attacker to guess. Finally, the client's initial sequence number is added to *z* to give the cookie *y*. This ensures that the server's initial sequence numbers increase at least as quickly as the client's does [1].

The ACK returned by the client to complete the connection must correctly specify *y*+1 in the acknowledgement field of the packet. To recover the cookie, the server subtracts one from the value in the ACK and also subtracts *x*. The server calculates a secret hash based on the information provided in the ACK packet as well as the current value of *t*. Hashes are also calculated based on the past few values of *t*. If one of these calculated hashes agrees with the cookie recovered from the ACK packet, then memory is allocated to the connection.

Two other proposed methods for prevention of denial of service due to connection depletion attacks are to shorten the timeout period or to randomly drop a connection when the backlog is full. The problem with these methods is that there is a risk of denying a legitimate client access, and decreasing the timeout may penalize users with slow connections.

Lazy receiver processing, another example of an OS method, can help make servers more robust when faced with data flooding attacks. Lazy receiver processing, developed by Druschel and Banga, focuses on reworking the network subsystem to improve server performance under high load conditions. Arrival of a packet signals a hardware interrupt. Many traditional operating systems place the packet in an IP queue and post a software interrupt. In the context of this software interrupt, IP processing is performed, protocol processing for protocols riding on IP is performed, and the packet is then placed in the appropriate socket queue based on its destination port. If a receiver is overloaded, packets may be dropped from socket queues to shed load. The downside to this is that resources have already been wasted on the packets by the time they reach the socket queue. Arrival of new packets will interrupt application processes to do processing on the newly arrived packets. Protocol processing is done without regard to which applications the newly arrived packets are bound. This can allow traffic destined for one application to choke out traffic destined for other applications. This type of network subsystem can lead to receiver livelock in high load situations, since the system may devote all of its resources to processing incoming packets, leaving none left to devote to the receiving applications [9].

In the lazy receiver processing system, packets are initially passed to socket queues instead of a shared IP queue. Incoming packets are scheduled for protocol processing at the priority of the application that receives the packets, and protocol processing does not occur until the application requests the packets. This diminishes the need for interrupt-driven processing. Load shedding is done early so that resources are not wasted on packets that will eventually be dropped anyway. When a socket's receive queue fills, the network interface discards additional packets destined for that socket [9].

4.3 Application Level

Numerous vendors offer intrusion detection systems (IDS) to help detect and deter illicit network activity. Such systems often keep statistics of file activity, user logins, disk activity, etc. Alarms are set off when significant deviation from normal activity occurs. Most intrusion detection systems also try to match network traffic with known attack fingerprints. For example, the system may look for teardrop-fragmented packets or Ping of Death packets. Some possible "alarms" set off when a potential attack is detected include logging the event, notifying the system administrator via e-mail or a pager, and launching programs designed to defend from the particular attack detected. It should be noted that an intrusion detection system can run either on an end host or on a dedicated machine on the network. Some switches incorporate IDS features as well [15]. Systems may also be scanned for configuration weaknesses (such as + + in .rhosts) and for the presence of DoS attack programs that an intruder may have installed.

4.4 Replication and Load Balancing

One DoS countermeasure is to replicate a potentially targeted service. To some extent, this is essentially a brute force countermeasure that does nothing to actually prevent attacks. However, replication and load balancing hinder attacks from bringing down the whole service. For instance, companies such as Akamai and Digital Island provide content distribution services to numerous companies with heavily trafficked websites. Though primarily intended to speed up access to services by distributing them and placing content closer to end users, distributing content can improve the robustness of services to denial of service attacks. If the targeted content is more widely distributed than the attack network, the chances of success of the distributed attack are greatly reduced.

4.5 Protocol Standard Level

Protocols may be enhanced or extended to support security measures. In general, of all countermeasures, protocol enhancements and extensions are the slowest to be put into practice. A major reason for this is the likely requirement of changes (in routers, operating systems, applications, etc.) necessary for implementation. An example of a protocol extension currently under development by the IETF is itrace. The goal of itrace is to provide a means of tracing the actual source of packet flows, regardless of whether or not the IP addresses have been spoofed. This protocol, which is still a work in progress, requires routers to randomly send special "traceback" messages to the destination with probability on the order of one in 20000. Thus, upon receipt of a traceback message, the recipient would know that the corresponding packet has passed through the router that sent the traceback message. A challenging design aspect is that attackers must be prevented from bombarding victims with fake tracebacks to cause confusion about the true source. One proposed method is to have routers digitally sign each itrace message, but the router overhead may be too significant for this method to be practical [3]. The itrace

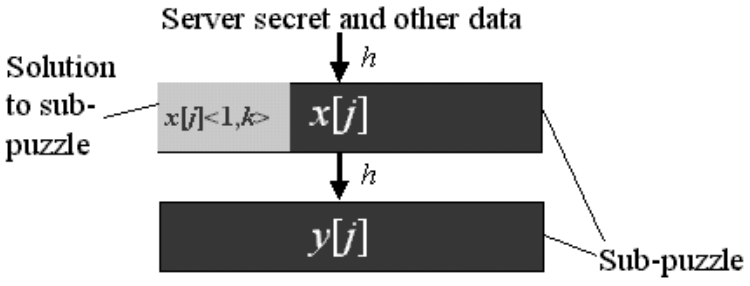
protocol should be able to effectively trace large data flood attacks originating from a small number of sources. Traceback messages would automate the current investigative process of manually contacting ISPs for cooperation in determining actual message sources. This, of course, does not actually prevent attacks, but the accountability factor may deter script kiddies from initiating them.

Another example of protocol enhancements is the set of security extensions to the DNS protocol referred to as the DNSSEC extensions. The fundamental objective of DNSSEC is to provide authentication and integrity to DNS info, and this is accomplished by the use of public key cryptography and authentication hashes. A secure zone computes the hash of each of its resource record sets, and the hash is then encrypted with the zone's private key. The recipient of the resource record set uses the sender's public key to recover the hash and computes its own hash based on the resource record set info. If the decrypted hash matches the newly computed hash, the data origin is authentic, and the data has integrity. New types of resource records were created to support these extensions [4].

Rather than changing or adding enhancements to existing protocols, protective protocol layers may be used. As a countermeasure to connection depletion attacks, Ari Juels and John Brainard of RSA Labs proposed the use of a cryptographic puzzle protocol that could either be integrated into an existing protocol or could be a separate protective layer. The main idea of this method is that if the connection buffer is almost full, clients are required to solve a cryptographic puzzle in order for the server to allocate resources. Under normal operation, the client sends a message to the server asking if there is a puzzle. The server responds that there is no puzzle, and normal protocol execution follows. If the server is under attack, it replies to the client's initial connection request with a puzzle and a timestamp. The client responds with the puzzle solution. The client must also send the original hash inputs other than the server secret so that the server does not have to store anything in memory (besides the secret). If the puzzle is correctly solved within a certain timeout period, then normal protocol execution follows [17].

A puzzle consists of several smaller sub-puzzles. To create a sub-puzzle (see Figure 3), the server first computes a hash $x[j]$ of the current time, the client's connection request message, the sub-puzzle number (i.e., which sub-puzzle of the puzzle it is), and a server secret (128 bits to avoid cryptanalytic attacks). The result $x[j]$ is then hashed to give $y[j]$. The sub-puzzle solution is the first k bits of $x[j]$, and the sub-puzzle itself consists of the remaining bits of $x[j]$ along with $y[j]$. Verification of puzzle solutions requires the k-bit solution to be appended to the remaining bits of $x[j]$ in the puzzle, hashed, and then compared to $y[j]$. Since there are 2^k possible solutions to a sub-puzzle, the client on average will need to guess 2^{k-1} times to determine the solution. A method for creating more efficient puzzles that require less server work is described in the paper as well. One of the nice features of the client puzzle method is that it allows for "graceful degradation," meaning that the server can scale puzzle difficulty with the severity of the attack. The main downside to client puzzles is that special client-side software would be required. However, this could possibly be made available through plug-ins [17].

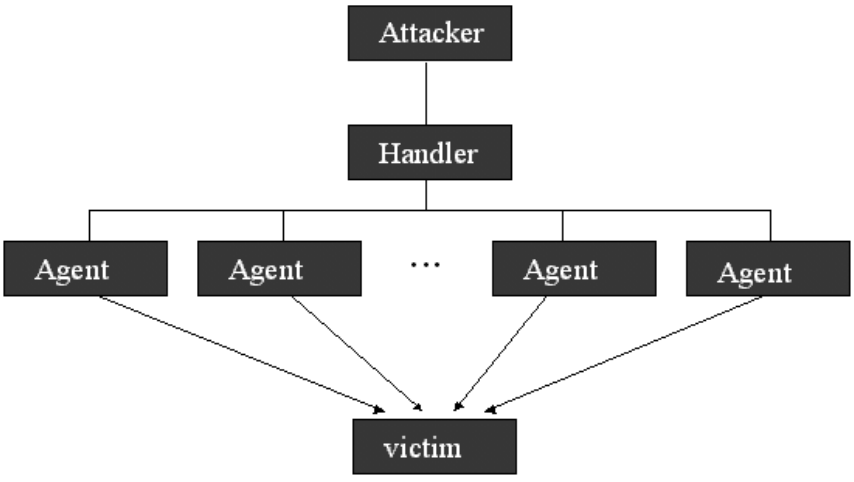
Figure 7 – Sub-puzzle Creation



5. Distributed Denial of Service (DDoS)

The attacks that brought down the University of Minnesota network and the major corporate sites were distributed. The basic model for a distributed attack is shown in Figure 8. There are a number of tools that implement the communication scheme shown in this figure, such as Trinoo, Tribal Flood Network, stacheldraht, shaft, mstream, and carko. First, these tools must be installed on compromised machines. This can be done by using separate programs that scan possibly thousands of hosts for the presence of known vulnerabilities. Vulnerable hosts are compromised, and attack tools are installed on them. According to [24], such programs are capable of compromising a host and installing the tool in under five seconds. Attackers usually recruit one or more "handler" machines, each of which controls the "agents" that actually carry out the attack. As faster "always on" connections such as DSL and cable modems continue to replace dial-up connections, distributed attacks are likely to become more and more of a threat due to the ease by which drone machines can be recruited [3].

Figure 8 – Basic DDoS model



DDoS tools differ in ways such as the protocols used to communicate, the DoS attacks used, and the methods of protecting the distributed network from detection. Stacheldraht can be used to illustrate some of the basics of how these tools work. Stacheldraht is capable of implementing SYN flooding, UDP flooding, and ICMP flooding. Communication between the attacker's client program and the handler is symmetric key encrypted. Handlers and agents communicate using TCP and ICMP, and communication between handlers and agents is Blowfish encrypted. The attacker connects to the handler using a TCP port and enters a password. If the correct password is entered, the attacker can enter a number of commands that perform functions such as pinging agents to see if they are active, starting a particular form of attack against specified IP addresses, setting a timer for attack duration, and killing all agents. Upon starting up, an agent file reads configuration file to determine which handlers control it. The agent proceeds to send an ICMP echo reply packet to each handler with the 666 in the ID field, and the string "skillz" in the data field. The master replies with an echo reply packet with 667 in the ID field and the string "ficken" in the data field [6].

Clearly, it is important for users and particularly for network administrators to keep up-to-date on the latest security issues and system patches. This can at least help prevent systems from being used to implement the attack. Also, intrusion detection systems can use signatures to detect the use of known DDoS programs. For instance, to detect stacheldraht agents, an IDS could monitor for ICMP packets with 666 in the ID field and "skillz" in the data field. The problem with this is that an attacker could simply change default values and strings to avoid detection. Disks can be scanned to see if embedded strings in a program closely match those known to exist in particular DDoS programs [6].

6. Design Considerations

Based on the denial of service attacks discussed, there are several concepts that should be kept in mind when designing network devices, operating systems, or applications. First of all, software designers must consider possible scenarios that would not make sense for normal, intended use of the software. One example is that attackers may cause loops by sending data or entering arguments that would not occur during normal use. For instance, the Land attack involves an attacker sending SYN packets to a victim with the both the source and the destination addresses spoofed to the victim's address and the source port equal to the destination port. This causes vulnerable operating systems to go into a loop as the host attempts to resolve a connection with itself. The fix is rather simple and involves dropping received packets that purport to be sent from the host to itself. Another example of a loop-causing attack involves a form of DNS cache poisoning to which early versions of BIND were susceptible. Suppose an attacker causes a name server to cache a CNAME record that has the NAME field set to the same address as the RDATA field. CNAME records map alias names into canonical names, so querying the name server and causing it to access this self-referential record would cause it to enter a loop and crash [4].

Bounds checking would eliminate numerous problems. For instance, checking bounds on memory copies prevents the teardrop attack from succeeding. Also, buffer overflow problems demand special attention, since they are extremely common and are used to cause services to crash and also to break into machines, which helps attackers set up distributed networks. One tool available to help prevent buffer overflow exploits is the Stackguard compiler, which places a "canary" word in front of return addresses. When a function returns, if the canary has been

altered, then the program halts after logging the buffer overflow exploit attempt [2]. Work has also been done in the area of automated testing for buffer overflow vulnerabilities. Software developers should be wary of using the culprit C functions, and for certain applications, using languages such as Java that perform bounds checking should be considered.

When designing a DoS countermeasure, it is important to make sure that the countermeasure does not hinder one form of denial of service attack at the expense of opening up new possibilities for attack. For instance, if a router offers a feature such as filtering and logging, then it is important that the router is able to withstand repeated executions of the feature. In an attempt to hog the router's resources, an attacker could send a large number of packets that the router must examine, filter out, and log. David Dittrich's analysis of the mstream DDoS tool offers an example. In April 2000, an mstream agent was discovered on a university network. A router implementing egress filtering became non-responsive while trying to filter out large numbers of packets having all 32 bits of the source address spoofed [7].

Many IDS systems open up possibilities for denial of service attacks as well. An example is that the IDS can be bombed with packets that generate false alarms, which have the effect of exhausting computational and/or human resources of network administrators trying to sort through the alarms. To make intrusion detection systems run faster and thus increase marketability, many systems treat packets in a stateless manner. Thus, a single packet can set off an alarm without previous events being used to help verify that the alarm is valid. This makes it easier for an attacker to rapidly cause false alarms [13].

Numerous DoS attacks focus on exhausting server resources, and clever allocation of resources can be helpful in mitigating the success of attacks. For instance, SYN cookies provide a more clever means of allocating resources during a SYN flood attack. Also, lazy receiver processing performs much better than traditional eager receiver processing platforms by introducing a generally better way of processing packets.

7. Conclusion

In order to successfully defend against present types of denial of service attacks and gain insight to future possibilities, it is crucial to develop a clear image of the broad spectrum of existing attacks and countermeasures. Although no "silver bullet" solution to the problem of denial of service attacks currently exists, various countermeasures can make attacks far more difficult to successfully devise and execute. An examination of the diverse range of attacks and countermeasures brings to attention several common threads such as the need to prevent infinite loops, the need to enforce bounds checking, and the need for clever resource allocation.

References

[1] Bernstein, D.J. "Syn Cookies." http://cr.yp.to/syncookies.html

[2] Bulba and Kil3r. "Bypassing StackGuard and StackShield." Phrack Magazine, Volume 10, Issue 56, File 5. http://phrack.infonexus.com

[3] Comerford, Richard. "No Longer in Denial." *IEEE Spectrum*, Jan 2001.

[4] Davidowicz, Diane. "Domain Name System (DNS) Security." 1999. http://www.geocities.com/compsec101/papers/dnssec/

[5] Dingledine, Roger and Fu, Kevin. "Concepts in Network Insecurity." Jan 10, 2001. http://snafu.fooworld.org/~fubob/pubs/sec-concepts-2001.ps

[6] Dittrich, David. "The 'stacheldraht' Distributed Denial of Service Attack Tool." December 1999. http://staff.washington.edu/ditrich/misc/stacheldraht.analysis.txt

[7] Dittrich, David. "The 'mstream' Distributed Denial of Service Attack Tool." May 1999. http://staff.washington.edu/ditrich/misc/mstream.analysis.txt

[8] "DNS #2." Win2000 Archives. http://home.win2000archives.com/2000/bugs/dns__2.html

[9] Druschel, P. and Banga, G. "LRP: A New Network Subsystem Architecture for Server Systems." USENIX Symposium on OSDI. Oct 1996.

[10] "Egress Filtering v 0.2." GIAC Special Notice, SANS Institute Resources. Feb 2000.

[11] Ferguson, Paul and Senie, Daniel. "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." Jan 1998. ftp://ftp.isi.edu/in-notes/rfc2267.txt

[12] "Finger bomb recursive request." http://xforce.iss.net/static/47.php

[13] Giovanni, Coretez. "Fun with Packets: Designing a Stick [draft copy]." http://www.eurocompton.net/stick

[14] Gamma. "DNS ID Prediction and Exploitation." August 1998. http://bak.spc.org/dms/archive/dns_id_attack.txt

[15] Graham, Robert. "FAQ: Network Intrusion Detection Systems." March 21, 2000 . http://www.robertgraham.com/pubs/network-intrusion-detection.html

[16] ISO/OSI Network Model. 28 June 1996. http://www.uwsg.indiana.edu/usail/network/nfs/network_layers.html

[17] Juels, Ari and Brainard, John. "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks." RSA Laboratories.

[18] Kenney, Malachi. "Ping of Death." January 1997. http://www.insecure.org/sploits/ping-o-death.html

[19] Peterson, Larry and Davie, Bruce. Computer Networks – A Systems Approach. 2nd ed. San Franciso: Morgan Kaufmann, 2000.

[20] "RealServer Security Update." http://service.real.com/help/faq/servg260.html

[21] Route/daemon9. "IP-Spoofing Demystified." Phrack Magazine, Volume 7, Issue 48, File 14. http://phrack.infonexus.com

[22] Route/daemon9. "Linux IP Fragment Overlap Bug" Bugtraq e-mail, 13 Nov., 1997.

[23] Secure Networks Inc. "Ascend Router Insecurities." March 1998. http://www.insecure.org/sploits/ascend.router.insecurities.html

[24] "Strategies to Protect Against Distributed Denial of Service (DDoS) Attacks." Feb 2000. http://www.cisco.com/warp/public/707/newsflash.html

[25] "UDP Denial of Services." http://www.netcraft.com/presentations/interop/dos.html