Denial of Service Attacks

Renaud Bidou

RADWARE
renaudb@radware.com

Abstract.

Denial of service attacks are not new. However, the recent increase of their power and their use by organized criminal organizations make necessary to consider them as one of the major issues IT infrastructures will have to face in the next few years.

Trying to defeat those attacks without understanding their technical aspects is illusory. As such, this document intends to provide as much detail as possible about IT weaknesses, techniques involved in DoS attacks and their impact. This is the first step in building a comprehensive and efficient security infrastructure that will protect the IT against this old but resurging threat.

History of Denial of Service

Genesis

It is always difficult to evaluate the very beginning of such a phenomenon. However some critical attacks have left traces and some techniques are so old that they are part of the History.

The Morris Worm

On November 2 1988 the first logical bomb was launched on the electronic world. That very day Robert Morris Jr. let his proof-of-concept worm invade the Internet. As a result about 15% (about 6.000) of the systems connected to the network were infected and stopped running.

The most surprising part of the story is the fact that this worm was not intended to block systems. Its only purpose was to propagate as far and as fast as possible. However, as it did not include any routine to avoid propagating locally and on an already infected system, it propagated thousands of times on each system, until resources were exhausted, leading to one of the largest breakdowns the Internet has ever known to that point.

SYNFloods

SYNFloods have existed since TCP has existed. They are a direct consequence of TCP specifications. It is therefore possible to say that SYNFloods are part of TCP just as spoofing is part of UDP.

Easy to implement, effective and hard to trace back to the real source, Denial of Service attacks are still appreciated by malicious Internet abusers, managing to launch hundreds of megabits, sometimes more than one gigabit, of SYNs targeted to a single service.

Major advances

It has not been long since the Internet become popular and the rise of new Denial of Service techniques. Many new techniques were discovered and investigated in the end of the 90's. Most of them still remain effective and are heavily exploited.

The Ping of Death

In 1995 the first popular and do-it-yourself anomaly-based Denial of Service appeared. Built on the incapacity of most TCP/IP stacks to properly handle ICMP packets longer than 65.635 bytes this attack showed the world that DoS was a reality that anybody could reach from the command line:

```
C:> ping <target> -l 65.511
```

This attack gave rise to a race for more and more anomalies that TCP/IP stacks were not able to manage. Fragmented UDP packets, weird TCP flag combinations, packets with same layer 3 and 4 sources and destinations and the like quickly followed. One can say that most possible anomalies have been explored during this period of time, and tools such as bo(i)nk, pepsi, land and teardrop are still in memories.

Smurfing

Once anomalies were patched, it became difficult to crash a system with a single packet. In addition Internet links at home relied on slow PSTN connections (14.400 to 33.600 bps) and the power of personal computers was far below that of servers. It became necessary to find leveraging factors in order to increase the effect of DoS attacks.

In 1998 a solution was found and the smurfing attack started to impact networks. This ancestor of reflection techniques relied on loosely protected networks (which were very common at the time) to relay their ICMP Echo traffic to the target. This target, flooded with ICMP Echo Replies from every system in the relay network, could reach very high CPU usage trying to handle each received ICMP packet and see if they match any entry in its tables. As a side effect, Internet links of the target IT infrastructure could be filled up with those ICMP replies.

DDoS

The DDoS onslaught on February 7th and 8th 2000 is a case study. This is the typical result of a static defense facing an active and moving attack line. About one year before the launch some proof-of-concept codes of DDoS elements started to become available. In October 1999 a bugtraq post provided links to the source code and even the CERT/CC issued an advisory in December the same year. Behind their firewalls IT infrastructures felt safe.

However, in just a few minutes yahoo, amazon, e-bay as well as other world major web sites were down on their knees and remained unreachable for as long as the attack lasted. They had been suddenly hit by thousands and thousands of legitimate connections, either crashing the components of the server or filling up the ISP uplink.

Maturity

The end of the 20th century was a period of exploration. A lot of new techniques emerged and proved their efficiency. But it was time to stop experimentation and move to the industrialization process.

CodeRed and Slammer

CodeRed was nothing new. It used blocks of technologies already known and widely exploited. But CodeRed was the first to put them all together and to target a popular application such as Microsoft IIS.

Then, exploiting the trivial UNICODE vulnerability CodeRed not only affected hundreds of thousands of systems but also loaded an agent whose purpose was to behave like an automated DDoS agent which was supposed to target the white house web site on August 2001. This automation and use of a worm to propagate a DDoS agent were quite new at the time.

The scale and speed of the infection were also unknown, including to the author of the worm. When he thought it would take about 3 months to propagate a single week was enough. Then the worm could be captured and reverse engineered. In the mean time IIS users were urged to apply a patch that had been made available for more than 6 months previously. This was the bright side of CodeRed: having people feel the need for massive and quick patch deployment solutions.

Slammer remains in memories as the second worm that heavily impacted the Internet infrastructure on January 25th 2003. This worm propagated by sending hundreds of thousands of small UDP packets. This lead to network congestion and router collapse. The effect found a huge leveraging factor in the choice of date and time of the attack: 6 AM GMT on a Saturday.

Broadband access and WiFi

In the early years of the Internet, home-users as well as SOHO relied on dynamic and slow Internet connections. Botnets then needed to be quite huge and were pretty unreliable. Cable and ADSL technologies changed the situation. Low prices, high bandwidth and permanent connection have propagated broadband access almost everywhere. A compromised university network is no longer necessary to launch a huge DoS attack.

Moreover the power of personal computers has also increased and made it possible to exploit the full capacity of bandwidth offered by broadband connections. Actual computers are able to generate more than 150.000 packets per second. For small packets, similar to the ones used for SYNFloods, this means that they can create around 80 Mbps of traffic.

Last, WiFi popularity lead to a huge number of unprotected or loosely protected access points that can be used to anonymously launch attacks on behalf of the legitimate owner of the Internet connection.

Denial of Service today

Techniques

The time for innovation in Denial of Service techniques is over and we are far from the discoveries of the last decade. However, broadband access, automation and increased power of today's home computers don't make any research necessary. This becomes particularly obvious when we find that old attacks such as *land*, which sends UDP packet with similar source and target IP addresses and ports, and disappeared in late 90's are back. The only improvement provided is the possibility to launch parallel tasks thus increasing the power of the attack in a way that was impossible to a simple 486 processor.

Another interesting point to take into consideration is the fact that IP stacks don't seem to have been properly patched. Computers don't crash with a single packet anymore; however, CPU operations remain high in order to handle such packets. As packet generation was limited at the time the patches went out, it was not that easy to realize. Maybe technology improved too fast. Whatever the reason, those old and obsolete attacks are now back and terribly efficient.

Use of Denial of Service

Denial of Service attacks were first used to "have fun", get some kind of revenge from system operators or make complex attacks possible, such as blind spoofing on r services. IRC servers were also often targeted after one got insulted on a channel. At this time networks and Internet uses were "confidential", and those attacks had very limited impact.

With time and as the Internet gets more and more used as a communication channel, *hacktivism* becomes more and more popular. Geopolitical situations, wars, religious concerns, ecology, any motive is then good to launch attacks on companies, political organization or even national IT infrastructures.

A more recent use of Denial of Service is linked to online gaming. Many servers have been victims of such attacks, generated by unhappy gamers who lost lives or their favorite weapon during game.

But the very use of Denial of Service today is definitely extortion. More and more enterprises rely on their IT infrastructure. Mail, critical data and even phone are handled by the network. Very few companies can survive without their main communication channel. Furthermore the Internet is also a production tool. Search engines and gambling web sites, as an example rely entirely on their connectivity to the network.

Therefore, as business relies, directly or not, on the Internet, old black mailing becomes digital. At first an attack is launched during a short and non-critical timeframe. Then the victim has to pay for "protection", else…

Denial of Service Techniques

Network protocols attacks

These attacks aim at the transmission channel, and therefore target the IP stack which is an entry point for critical resources such as memory and CPU.

SYNFloods

SYNFloods are typical *concept-based* denial of service attacks as they entirely rely on the way TCP connections are established. During the initial 3-way handshake, the server fills up a table, the TCB (Transmission Control Block), which keeps session information in memory. When a server receives the initial SYN packect from a client, it sends backs a SYN-ACK packet and creates an entry in the TCB. The connection is in a TIME_WAIT status, for as long as the server waits for the final ACK packet from the client. If this final ACK packet is not received, another SYN-ACK is sent to the client. Finally, if after multiple retries none of the SYN-ACK packets are acknowledged by the client, the session is closed and flushed from the TCB. The period of time between the transmission of the first SYN-ACK and the closure of the session is usually around 30 seconds.

During this period of time it is possible to send hundred of thousands of SYN packets to an open port and never acknowledge the SYN-ACK packet of the server. The TCB is quickly overloaded and the stack does not accept any new connections and existing ones are dropped. As there is no use for the attacker to receive the SYN-ACK packet from the server the source address of the initial SYN packet can be spoofed. This makes it more difficult to trace the real source of the attack. Also, as SYN-ACK packets are not sent to the attacker, this saves bandwidth on the attacker's side.

Generating this kind of attack is trivial, as a single command line is enough.

```
# hping3 --rand-source -S -L 0 -p <target port> <target IP>
```

Few variants exist, usually adding some anomalies to the SYN packet in order to increase CPU usage. These will be *legitimate* anomalies such as sequence number or source port 0.

SYN-ACK Flood

SYN-ACK Floods rely on CPU resource exhaustion. Theoretically this kind of packet is the second step of the TCP 3-way handshake and there should be a corresponding entry in the TCB. Browsing the TCB uses CPU resources, especially when the TCB appears to be large. Then, under heavy load, this resource usage can affect the performance of the system.

This is what SYN-ACK attacks take advantage of. Sending a huge load of SYN-ACK packets to a system dramatically increases its CPU usage. Efficiency of the attack is then affected by the choice the hash algorithm and the size of the hash table used to organize the TCB (see applications concepts and logical weaknesses).Furthermore, as those SYN-ACK packets do not belong to an existing connection the target has to send RST packets to the source, increasing bandwidth usage on the link. As for SYNFloods, the source IP can of course be spoofed in order to prevent the attacker from receiving RST, thus increasing its available bandwidth for attacks.

Once again a simple command line will be enough to generate the attack.

```
# hping3 --rand-source –SA –p <open port> <target IP>
```

An interesting factor is the capacity to have SYN-ACK packets generated by third-party servers thanks to a *reflection* mechanism. When a SYN packet is sent to an open port of a server, this server sends back a SYN-ACK packet to the source. Then any server can become a relay for such attacks. A simple SYN packet sent to a server, with a source spoofed to that of the target generates a SYN-ACK back to the target. This technique makes tracing harder. Also, in some specific cases, it may be possible to bypass some anti-spoofing mechanisms, especially when the target and the attacker belong to the same backbone and uRPF (see anti-spoofing) is deployed far enough away from them.

It is also possible to increase the power of this attack by coupling it with a SYNFlood. SYNFloods create entries in the TCB which becomes larger and larger. Then a SYN-ACK flood becomes, far more efficient as the browsing of the TCB gets longer.

UDP Flood

UDP is also naturally bound to be a vector of Denial of Service attacks. As it is specified, a server receiving a UDP packet on a closed port sends back an ICMP Port Unreachable packet to the source. The data part of the ICMP packet is filled with at least the first 64 bytes of the original UDP packet. As no limit or quota is specified as a standard, it is then possible to send huge amount of packets on closed ports. At very high load, operations necessary to generate ICMP error packets consume a lot of CPU, eventually leading to CPU resource exhaustion.

Generating such attacks is once again possible from a simple command line.

```
# hping3 --rand-source --udp <target IP> --flood
```

Once again spoofing can be used so that ICMP packets don’t lower the bandwidth of the attacker.

Anomalies

Anomalies are specific cases that may lead the IP stack to misbehave with various consequences such as crash, freeze etc. Two major families of anomalies can be distinguished: illegitimate data and off-limit exceptions.

Illegitimate data are values or content that doesn’t respect or are explicitly denied by standards. Packets longer than the size specified, overlapping TCP flag

combinations, SYN packets with not null acknowledgement sequence numbers or even bad option types are example of illegitimate data based anomaly attacks.

Off-limit exceptions are based on exceptional cases not properly handled by the stack, even if they are perfectly legitimate from the standards point of view. The famous *ping of death* was just about huge (but still legitimate) ICMP echo request packets. Packets with identical source and target addresses and ports are also legitimate but are harmful to IP stacks.

Few anomaly attacks are still able to down systems with a single packet. Most stacks have been patched and probably most exceptions have been tested and exploited. However handling such packets is still costly in terms of CPU. When anomaly attacks appeared and got patched, 5 years ago or more, attackers' power was limited in terms of CPU as well as bandwidth. Nowadays, the gap between workstations and servers is getting smaller and broadband is available to anybody. With such power, huge load of anomalies can be launched, generating CPU exhaustion on targets.

Again such attacks can be launched from a single command line.

```
# hping3 --rand-source -SAFRU -L 0 -M 0 -p <port> <target> --flood
```

Again spoofing is still a valid, and "recommended", option.

Connectivity attacks

Networks rely on different layer of connections. From the physical cable to the application layer multiple mechanisms are involved in communication setup, handling and termination.

Layer 2 connectivity attacks

Wired networks connectivity attacks

The first and most obvious, solution to generate a denial of service on layer 2 connectivity is to target internetworking devices, usually switches. *ARP Flooding* attacks, once used to have switches behave like hubs, are still efficient as some switches do crash when their switching table is filled up. Then sending hundred of thousands of frames with random source MAC address will usually be enough to prevent any communication from and/or to hosts connected to the switch. If a target valid MAC address is provided it is then possible that all the switches between the attacker and the system with the target MAC address will be downed.

On wired networks ARP mechanisms are also an excellent vector. This resolution protocol makes *blackholing* attacks, which consist in redirecting the traffic to *nowhere* or to a silent node, possible in different ways.

ARP poisoning is the easiest to perform such attack. It simply consists in sending unsolicited ARP announcement. This type of ARP packets have been originally designed to clear ARP entries in cache when a new system appears on the network. They contain the IP and MAC addresses that should be entered in ARP tables of systems belonging to the same physical network. Preventing systems on such network to get out can be simply performed by continuously sending ARP announcement for

the default gateway pointing to an invalid MAC address. Any system within the physical network will then update its ARP table. Any communication attempt to another logical (layer 3) network will then be blocked as frames will be sent to an invalid physical address. Larger denial of service can be performed by sending fake ARP announcement concerning any system on the physical network.

SNMP can also be a good vector to corrupt ARP table. ARP entries are stored in the *ipNetToMediaPhysAddress* leaf of the MIB. With write access to the MIB, usually easy to guess on workstations, it is possible to change entries into the ARP cache which at least contains the default gateway.

Wireless networks connectivity attacks

On wireless networks *blackholing* can be performed by setting up fake access points. If such access point can be positioned close to the targeted network, with exactly the characteristics such as ESSID and WEP key and providing a better signal, most wireless clients will automatically associate with this fake one. As it provides no connectivity to any network, no communication will be made possible. In many cases encryption and authentication characteristics are not even necessary, depending on the specific stack of each operating system or driver.

Another efficient techniques abuses of deauthenticate and disassociate wireless control messages. Sent continuously to the victim MAC address with source spoofed to the MAC address of the access point, those frames will disconnect the target each time it gets connected. Any communication then becomes impossible. As wireless control messages are always sent in clear, even if WEP or WPA is used, this attack appears to be one of the most efficient to be performed on wireless networks. Large scale deauthenticate/disassociate floods are also possible thanks to broadcast messages. However they are rarely supported.

Layer 2 protocol attacks

Layer 2 complex architectures and functionalities obviously need specific protocols for management and auto-configuration. The oldest one is spanning tree, designed to provide high availability at layer 2 level. Switches elect a *root* device and communicate with each other thanks to UDP multicast communication: BPDU (*Bridge Protocol Data Unit*). These packets contain cleartext information, command and data. It is then easy either to fake answer from a better, and non-existing, candidate during next *root* election or to endlessly generate new elections by sending BPDU with a priority higher (inferior id) than this of the actual *root*.

Cisco proprietary protocol CDP (Cisco Discovery Protocol) is used to get networking neighborhood information. Communication between devices is made through the layer 2 multicast address 01:00:0C:CC:CC:CC. No authentication is either required or implemented. In case of a flood switches memory gets filled up and the devices starts to misbehave (exact effect depends on the IOS version). These typical weaknesses based on cleartext messages and lack of proper authentication can be exploited in other protocols such as VTP and DTP (*VLAN/Dynamic Trunking Protocol*) to easily create layer 2 *blackholes*.

Layer 3 connectivity attacks

Network layer local blackholing

Blackholing techniques can also be performed at the network layer. A basic but efficient one consists in using ICMP redirect messages to victims, providing them with a non-routing new gateway for any communication out of its logical network. As no check is performed at the stack level, this attack is very efficient and easy to generate.

```
# sing -red -S <default gateway IP> -gw <fake gateway IP> -dest
0.0.0.0 -x net <target host IP>
```

Dynamic IP configuration is another good vector for network *blackholing*. DHCP relies on Ethernet multicast, UDP and blind trust into unencrypted and unauthenticated data provided by unknown systems, both on client and the server side. Spoofing from layer 2 to the application is then trivial and once the original server is down or unable to provide addresses anymore (see application layer attacks), taking over the original DHCP server is quite straightforward. It is then possible to provide erroneous IP information to any client trying to connect to the network. This kind of attack is very simple to implement and deadly efficient on local networks. Moreover it is difficult to analyze and mitigate as there is no information in frames that could help discover which the fake server is.

Router discovery is another dynamic configuration protocol that can be exploited to create layer 3 *blackholes*. When the stack implements this protocol systems are automatically affected to a multicast group, listening for routers announcement. Cleartext and non-authenticated messages are then easy to forge in order to provide erroneous information about the default gateway.

```
# irdp -i eth0 -S <fake gateway IP>  -D <victim IP>
```

The HSRP (*Hot Standby Router Protocol*) can be exploited almost in the same way as it makes use of multicast and cleartext authentication. It becomes trivial to send fake announces coming from non-existing IP address and claiming to be the active router. This appears to be a layer 3 equivalent of the spanning-tree attack described before.

Sessions interferences

An old technique that recently resurged makes use of ICMP destination unreachable (port/protocol unreachable) messages to perform blind TCP session dropping between two hosts. Necessary information is source and target address and ports. This information is checked thanks to the TCP header provided by the source of the ICMP message. The key point in this attack is the fact that stacks do not have to check for sequence numbers, making spoofing easy. Efficiency of this attack has been essentially proved against protocol such as BGP which relies on persistent sessions.

Additional effect can be obtained on hosts compliant with RFC 1191, in using ICMP code 4 (Fragmentation needed and don't fragment specified) and providing a new next hop MTU with a low value, down to 68 bytes for IPv4. Next connections will then be established with very low performances. ICMP source quench message

can also be exploited to dramatically lower performances of networks. Standards specify that TCP must slow down transmission rate when receiving such packets. Once again performances of the network will be dramatically reduced.

At TCP level, blind reset attacks technique is made possible thanks to many factors that lower complexity of sequence number and source port guessing. First TCP specifies that any packet with a sequence number between expected sequence number and expected plus window size is to be considered as valid. This can lower the number of possibilities by a factor up to 2^16. Source port guess takes advantage of the fact that some port ranges are reserved as they are either privileged (1-1024) or often considered as private by stacks. Also source ports are not randomly chosen, but use linear increments, this can exploited in some situations.

Application level attacks

Networks proved to be vulnerable. However they are only the transport part of the global system and a good means to interrupt communication. However, applications are usually the actual target and they are also exposed to numerous denial of service issues.

Session based attacks

Application connectivity is mostly handled by sessions, usually identified by TCP mechanisms. The number of simultaneous sessions is an important factor that affects the performance of a given application and, as such, has to be limited. When this limitation is simply based on network and transport information (IP+TCP), it appears that generating a denial of service is trivial. A simple attack opening TCP sessions and leaving them open will quickly fill-up all available session slots, preventing any new session from beeing established. This *pending session* attack is a layer 7 equivalent of the SYNFlood. But when gigabits of traffic may be necessary at layer 4, only a few thousands of packets sent within a few seconds are necessary to block any new session establishment.

As an example such attack can be easily performed on a webserver.

```
# webdevil <target ip> <target port> <number of sessions>
```

Full and legitimate sessions can also damage applications, starting with the *F5 attack*, which consisted in keeping the F5 key pressed to force a complete refresh of the web page loaded on Internet Explorer. With this old and trivial attack, server resources could get exhausted simply because of the number of web pages to serve. This kind of *session flooding* attack can also damage other critical paths of the communication channel.

- Telecom link: the volume of data transferred from the server to the client can fill-up the link to the Internet. In this case no communication is possible through this link; the focused denial of service attack becomes global;
- Server application: high number of connections can reach the limit of authorized simultaneous session, the attack becomes similar to a *pending*

session attack. If the limit is not set handling of a high volume of sessions may consume most of the system resources.

- Third-party application: Most applications are linked to middleware and databases. In each case, possible bottlenecks can appear as those third party applications may encounter internal problems handling the huge amount of requests sent by the original application. These effects can also be the consequence of internals weaknesses (see below).

Session flooding techniques did not evolve since the F5 one. However leveraging factors have been found and widely used, making such attacks still one of the worst possibles case of denial of service.

Concepts and logical weaknesses

Applications are developed to provide a specific service in normal conditions whereas an attacks purpose is to have applications behave in an exceptional manner. Some internals of such attacks are generic, but most of them are specific to each application concept and logic, making it impossible to provide an exhaustive list of vulnerable applications and the way to exploit them.

Internals

The first obvious internal mechanism that could lead to a denial of service is memory handling. A simple *buffer overflow* will make it possible to rewrite the stack, leaving the application, and maybe the whole system, unstable. Lack of input checking at different levels of the application channel will also propagate the attack along third party applications, increasing the possibilities of impact.

However, internal weaknesses can be more subtle, and harder to fix. Regular expressions relying on NFA engines can be very dangerous. NFA engines analyze all possible paths to the expression. When a character invalidates the search the engine goes back to the previous matching point and retries all combinations of the expression. A combination of greedy operators such as the * metacharacter and *OR* conditions such as *(int|integer)* can be deadly when launched against a carefully crafted input.

SQL is also an obvious vector for logical attacks as SQL queries on large and complex tables can generate different misbehavior at the application level. The first encountered weakness is the lack of indexes and the general use of string columns in database structures. Without proper indexes, SQL requests involving filters and sorting operations on multiple columns of the same table generate an exponential number of operations, consuming huge CPU resources and dramatically increasing application response time. These operations are made even greedier if the fields to be searched and sorted are strings. The second weakness is memory related. On huge tables, "*SELECT **" like requests can generate millions of results. Temporary database structures as well as applications structures used to store the result consume noticeable amounts of memory that may lead to various results from excessive swapping and increased response time to system-wide denial of service.

Another interesting denial of service attack can be performed on hash functions used to organize and search data in memory. Hash table entries are pointers to chained

lists of object. Two steps are necessary for search operations. First the hash function is calculated to find the matching entry in the hash table. Then, objects in the list are compared one by one. Collisions are the first weakness of the mechanism. If the hash table is not large enough and/or the hashing algorithm makes collision simple to create it becomes possible to generate entries that will fit the same entry into the hash table. This way any search into the table will need numerous operations and consume CPU, potentially leading to a denial of service by itself or leveraging the effect of an attack targeted at an application which relies on such mechanism.

Operating modes

Each application has specific functionalities and operating modes. It is therefore impossible to describe all vulnerabilities. However, some typical and efficient examples may provide clues on commonly exposed operations.

DHCP servers are exposed to a trivial attack that can quickly exhaust their pool of available IP addresses. Vulnerabilities in the protocol itself have already been analyzed (see layer 3 connectivity attacks). In addition to this DHCP servers lock IP addresses during the first stage of DHCP exchange without any acknowledgement from the client. Having a DHCP server run out of available IP addresses is then just a question of sending multiple DHCPDISCOVER requests with different MAC addresses specified in the *chaddr* data field. MAC spoofing is not even necessary. The combination of broadcast, lack of authentication and "early stage processing" is the key factor for a denial of service attack.

Another "state of the art" attack, based on badly designed operation mode, is the flooding of DNS servers. When a DNS server receives a request for the resolution of a name that is not in its cache, or when the request specifies that the answer must be authoritative, the server sends a request to the TLD server (*Top Level Domain*) in order to get the address of the SOA (*Start Of Authority*) of the domain. Once provided with this address the target DNS server issues another request to the SOA server in order to have the requested name resolved. Once it gets the answer it sends it back to the client. As no authentication is needed from the client and the communication between the client and the server is based on UDP protocol, it is trivial to send thousands of requests from a spoofed source to the server. Different level of efficiency can be distinguished.

- Requests for hosts on non-existent domains: in this case the TLD sends an error to the target DNS server which forwards it to the client. The impact is relatively low and requires a lot of requests from the attacker. However, creating a tool to generate such attacks is trivial as any field (host, domain), can be randomized and doesn't need to be real.
- Requests for hosts on few existing domains: here the target server issues one request per domain to the TLD to get the SOA. Then, each request from the attacker is forwarded to the SOA. Non-existent hosts generate errors. On the other hand, requests concerning existing hosts generate answers from the SOA, which are larger packets to forward to the source. In this case, higher traffic and more important processing improve the efficiency of the attack. Such an attack is still relatively easy to perform, as finding a few existing domains is simple. However the efficiency highly relies on the capacity of

SOAs of those domains to handle the high volume of requests sent to them by the target server and the availability of numerous hostnames on such domains.

- Requests for existing hosts on multiple existing domains. In this last case, almost every request sent to the target server causes it to generate a request to the TLD, followed by a request to the SOA and the processing of the answer. This attack is hard to implement as it needs to define a list of thousands of existing domains and hosts. However it is particularly efficient and should bring down most DNS servers with only a few thousand requests per second.

This DNS case is mostly made possible because of the use of stateless protocol, making it possible for an attacker to generate high load of traffic and processing on the server side without being impacted.

Context and session handling is another interesting weakness often found in applications. In the case of an SMTP server implementing some checks on the validity of HELO command's arguments, this effect is leveraged by the lack of immediate processing of the result. According to standards a mail can only be rejected once the RCPT TO command is issued. Therefore, establishing one session and issuing thousands of HELO requests will have the target server repeatedly process the arguments, usually against regular expressions. The worst case is when name resolution is performed as this adds some more processing and traffic. As a single HELO command is around (100 bytes) it is possible to launch very effective attacks on mail servers from a standard PC connected to the Internet through broadband access. If a session were to be interrupted as soon as an anomaly was detected this attack would not be possible.

Improving efficiency

Most Denial of Service attacks are quite simple to implement. However, in some cases a standard PC and broadband access may not be sufficient to be effective. This is particularly true when mechanisms such as clustering and load balancing are implemented on the target infrastructure. A single target IP address will then be physically linked to many servers, creating the need for an attack of larger scale.

There are two ways to make an attack efficient in such context. The first is to find some leveraging factors that will increase the power (number of packets or sessions, bandwidth etc.) of the attack vector. The second is to rely on some side effects that impact devices or resources on the communication path.

Side-effects

Small packets

The most common side effect is the impact of huge numbers of small packets to be handled by inline network or security devices. Depending on functionalities implemented on such devices, forwarding a packet from one interface to another may need several operations. A simple router must make a decision according to its routing table. In case of packet filtering operations, layer 4 headers must be checked to validate filters. Stateful inspection adds complexity as it is needed to check session tables. Last application layer devices such as reverse proxy must handle the packet at layer 7. Some other operations such as packet mangling or NAT requires even more processing as the packet has to be rewritten and several checksums recalculated.

The effect of small packets can be easily described with figures. On average, a device specified to handle a specific bandwidth will hardly provide a throughput of more than 10 percent of announced performances when dealing with 64 byte packets. Any attack relying on packets smaller that 100 bytes may first bring router and firewalls down instead of only impacting the target server.

Uplink flooding

Another common side-effect is the flooding of links on the path to the target. Requests and packets sent by the attacker rarely generate enough traffic to fill up the links. However, responses from the target usually add considerably the traffic. A basic example is that of a SYNFlood. A SYN packet of 64 bytes causes the target to generate 3 SYN-ACK (due to retransmission attempts). This kind of effect is even more efficient in the case of application-based attacks. A simple HTTP request is usually around 200 bytes when the data sent back to the client is usually around 10.000 bytes.

In “one-to-one” attacks it is essential that the response from the server doesn’t make its way to the attacker, as this would eventually also flood the attacker’s link. In

such case, it is necessary for spoofing to be possible and easy to perform. UDP based applications are perfectly suitable for generating such effects. This is particularly simple to demonstrate in the case of DNS floods (see logical weaknesses / operation mode). Without any consideration of requests to TLD and SOA (which again, increases the traffic through target Internet link), a request for MX records on an important domain will provide an interesting multiplication factor to the original request volume.

```
# dig radware.com mx
```

- Request: 71 bytes
- Answer: 551 bytes
- Multiplication factor = 7¾

These side effects are to be considered as a major concern, especially in a shared infrastructure where an attack targeted at a single server may impact dozens of systems relying on the same infrastructure as the original target.

Leveraging effects

The Push flag

The PSH flag is a protocol-based leveraging factor for any TCP based attacks. When a packet is received by the stack it is stored in a specific buffer with other received packets. Once the buffer is full, packets are sent for further processing. This transfer of packets generates a considerable overhead and buffers are there to optimize the "cost" of this change of context. The PUSH flag forces the stack to free the buffer. In the case of an attack based on TCP such as SYN, SYN-ACK or anomaly floods or even application layer attacks, the cost for an attacker to add a PUSH flag in the TCP header is null. On the other hand the server dramatically suffers from the overhead introduced by permanent context change.

Malicious traffic replay

In some cases the power of the attacking system may reach limits. This is mainly true for attacks performed over a LAN where gigabit throughput may be hard to reach with a standard PC. In order to bypass such difficulty, it is possible to take advantage of packets replay capacity offered by several tools. The principle is relatively simple. In order to get rid of CPU usage needed to create packets, get random source address/port, calculate length, checksum etc. Traffic is sniffed when the attack is launched. Then the capture file is replayed in loop with a high multiplication factor, to make sure the limits of attacking resources are reached. The sample of original traffic must be relatively important to make sure that it is difficult for the target to identify the recurrent schema and simply filter out sources.

Reflection

Reflection is a mechanism that uses other systems as relays and / or amplifiers for the attack. As for most attack types, reflection can be performed at network or application level. In the first case the purpose is to get a multiplying factor of the number of packets sent to the target. In the second case reflection is used to get the highest possible ratio of (data sent by attacker)/(data received by the target).

The *smurfing* attack is an excellent example of what a network reflection attack is. Basically it consists of sending ICMP ECHO REQUEST packets to a broadcast address, and with a spoofed source aiming at the target of the attack. Loosely protected networks may allow such broadcast pings, leading every system of the network to send ICMP ECHO REPLY packets back to the target. The choice of the relay network is important. It must obviously allow broadcast pings. Furthermore, it should provide high bandwidth in order to be able to transmit huge amount of ICMP ECHO REPLY packets at a very high rate. The efficiency of the attack is very easy to evaluate. On a 256kbps ADSL uplink it is possible to send about 1100 ICMP packets per second. With a relay network of "only" 50 systems the target will be flooded with more than 50.000 ICMP packets per second for a total bandwidth of around 10 Mbps. This is a very easy attack that anybody can perform with a standard PC, a broadband Internet access and a single command line.

```
# hping3 --icmp --spoof <target address> <broadcast address> --flood
```

At the application layer, the objective of the attack is to either flood the target application or fill up the Internet link with data sent by a relay host. The basic DNS example described above (see Side effects / Uplink flooding) is a perfect example of application-based reflection that can be used against another DNS server. As the application relies on a stateless protocol, it is easy to have huge responses (to small requests) sent back to a spoofed DNS server to which UDP/53 traffic is authorized, thanks to a single command-line.

dnsreflect <target IP address>

Another, and more recent application refection attack relies on SIP (*Session Initialization Protocol*), used for signaling purposes by RTP (*Real Time Transport Protocol*). The need for proxies between SIP servers in order to interconnect SIP UA (*User Agents*) implies that extremities of the communication channel must be specified in the header of the SIP INVITE request. Particularly the Via: field identifies the system with which communication must be established. As no authentication is performed, and as SIP relies on UDP, *spoofing* is trivial. The understanding of the possible leveraging effect is then quite straightforward. A typical SIP INVITE request is around 1 KB. With such request it is possible to have a server continuously send several dozens of kbps of streaming media to the spoofed source. On a 256 kbps ADSL uplink, it is then possible to create around 64 such connections per second, quickly filling up the uplink of the target.

Third-parties

Packet generation and bandwidth are not the bottleneck an attacker may face when launching denial of service attacks. When dealing with an application that relies on TCP sessions handling by the attacking stack may be a limitation, especially in the case of pending sessions. If such operation is performed by the standard stack

functions then it is highly possible that the total number of sessions accepted by a web server farm will be higher than the maximum number of session a system can establish. In such case pending session attacks will not be efficient and the only system to be impacted by the attack will be the attacking system itself.

One possibility, in order to be able to establish a higher number of sessions than the expected limit, is to have a third-party server sending crafted RST packets to the attacking station. This third party is setup on the network of the attacking system and sniffs the traffic between the attacker and the target. While the session is established, the third-party host keeps track of session information, IP addresses, ports and sequence numbers. When the 3-way handshake is completed, it sends a fully legitimate RST packet to the attacking system, based on information it kept in memory. This way the attacking system releases the connection while it is still considered as open by the target server.

Botnets

The last, but the most famous leveraging effect is the use of bots. The purpose is simply to have huge numbers of third party systems perform the attack. This obviously increases the power of a single attack performed by a lonely system and makes it possible to severely damage high performance networks and systems. Several parameters must be considered when dealing with bots. The first is the way they are installed on third-party systems. Second is the command channel through which orders are transmitted, and last is the operation performed.

The first characteristic of bots is that they behave on behalf of legitimate owners of the systems. This means that they are silently installed and are supposed to run more or less stealthily. As bots are efficient when they group hundreds or thousands of compromised hosts, it necessarily means that they rely on worms or mass-hacking mechanisms for their installation. Mass hacking is the basis of automated installation. It is performed by a script that tries to exploit several vulnerabilities. In case of success, the payload installs the agent. Propagation speed is then relatively slow and the source of the botnet can easily be traced back. This technique has been improved thanks to worms. In this second stage, the payload embedded in the attack installs the agent, and then automatically tries to compromise other systems from this new starting point. It provides faster propagation and makes it difficult to identify the source of the botnet at this stage, as it makes it necessary to trace back infections step by step up the original sender of the worm.

The architecture supporting the command channel is now usually a 3-tier architecture involving one or more masters, one or more agent handlers and several agents. Masters are the points from where the order to launch the attack comes from. Agent handlers are the most important part of the architecture. Their objective is to watch for available agents and to transfer orders from the master. Agents, also known as zombies, are the processes running on compromised hosts. They are responsible for performing the attack itself.

Agent handlers are the most variable part of the network and usually define protocols used for communication between each part of the architecture. The key points are the direction and the nature of the communication between agents and agent handlers. Communications established by agents to agent handlers are usually more efficient, as firewalls are more likely to let traffic going out of the internal

network pass. Legitimate protocols as a transport layer are also more efficient. Agents getting orders through HTTP will rarely be blocked as communication to external web servers are usually authorized by corporate security policies. If the protocol used is not HTTP but the agent handler relies on TCP port 80 for communication, protocol anomaly detection is needed to block such traffic. Therefore, communication over non-standard (HTTP, SMTP etc.) ports are usually blocked, but by loosely configured firewalls.

The most common agent handlers are bots running on IRC channels. When a host is compromised it connects to a specific IRC channel. Agent handlers then only have to check for active members to establish the list of available agents. Agents perform operations based on the orders sent on the IRC by the master. This type of agent handler is reliable, flexible and easy to set up. However, IRC communications are often blocked by firewalls and the overall architecture lacks stealth. It is then easier to trace the source back to the master. The use of covert channels should be considered as the near future of botnets management.

Operations performed by bots are definitely floods. This is logical as the main interest of bots is to take advantage of the huge number of third party zombies. Almost all flood techniques have been implemented on botnets, from the basic SYNFlood to custom, application oriented floods. As a consequence, the very specificity of a good botnet is the capacity of agents to adapt their behavior according to orders sent by the master. As mitigation mechanisms differ, an agent able to perform SYNFlood, anomaly flood and to establish thousands of legitimate HTTP sessions makes possible for the attacker to dynamically change the type of attack according to effects on the target. Another advantage is the possibility to manage one single botnet from which multiple operations can be sent.

Flashmobs

The main drawback of botnets is that they rely on static sources. Over time it is possible to blacklist compromised IP addresses and to notify the network administrator that some of the systems have been compromised. It is to be considered that large botnets have a limited time-to-live. This is not the case of flashmobs.

The objective of such mechanisms is to have innocent users perform the attack without being aware of it. The mechanism is quite simple. The attacker sets up a web site. On this web site some malicious content forces the innocent user's browser to launch an attack on a specific target. Usually it is either hundreds of hidden frames linking to the target web site or a dynamic code, such as Java applet or ActiveX, that make it possible to perform some more advanced operations. Increased power of home computers and generic broadband access make it possible to generate important traffic without the user noticing it. On the target side, blacklisting becomes impossible as each time a new user connects to the web site its computer becomes a new source of the attack. The drawback is the difficulty to have a web site with enough traffic to ensure continuous and important traffic to the target.

Conclusion

The real difficulty in denial of service mitigation is linked to the multiple techniques involved in such attacks. An efficient SYNFlood protection will be of no help against session pending attacks and useless in the corporate network if the access router is brought down because of the huge packet rate.

Attackers are aware of this and know how to take advantage of the situation. When an attack is not efficient enough they quickly move to other techniques. This has been commonly observed and raises the need for rich, flexible and high performance security technologies.