34. Denial-of-Service (DoS)

34. Denial-of-Service (DoS)

34. Denial-of-Service (DoS)

34.1. Introduction

Since the bandwidth in a network is finite, the number of connections a web server can maintain is limited. Each connection to a server needs some minimum amount of network capacity in order to function properly. When a server has used up its bandwidth (the ability of its processors to respond to requests), any additional attempted connections are dropped. Any attack that is designed to cause a machine or a piece of software to be unavailable and unable to perform its basic functionality is known as a denial of service (DoS) attack.

A majority of DoS attacks refer to deliberate attempts to exceed the maximum available bandwidth of a server usually through exploiting program flaws or resource exhaustion. The underlying concept is that since different parts of the system might have different resource limits, the attack only needs to exhaust the part of the system with the least resources (i.e. the bottleneck). Since attackers in a DoS attack are not concerned with receiving responses from the server, they often spoof the source IP address in an attempt to obscure the identity of the attacker and make mitigations of the attack more difficult. Because some servers may stop DoS attacks by dropping all packets from certain blacklisted IP addresses, attackers can generate a unique source IP address for every packet sent, thus preventing the target from successfully identifying and blocking the attacker. This use of IP spoofing therefore makes it more difficult to target the source of a DoS attack.

34.2. Application Level DoS

Application level DoS attacks tend to target the resources that an application uses and exploits features of the application itself. Some attacks rely on asymmetry wherein a small amount of input from the attack results in a large amount of consumed resources. Such attacks could include exhausting the filesystem space by having continuous calls to write, exhausting the RAM by having continuous calls to malloc, exhausting the processing threads by having continuous calls to fork, or exhausting the disk I/O operations. Defense against such attacks usually take on a three-pronged approach:

1. Identification: You must be able to distinguish requests from different users and require some method to identify or authenticate them (though this process might be expensive and itself vulnerable to DoS attacks)
2. Isolation: You must ensure that one user's actions do not affect another user's experience
3. Quotas: You must ensure that users can only access a certain proportion of resources. There are many possible implementations of this. One method to implement this is to place specific limits on each user such as limiting users to only 4 GB of RAM and 2 CPU cores. Another example of is to assign specific roles to users such that only trusted people can execute expensive requests. Another possible "defense" would include proof-of-work (like CAPTCHA) wherein you force users to spend some resources in order to issue a request. The idea here is that the DoS attack becomes more expensive for the attacker as they have to now spend extra resources in order to succeed.

34.3. SYN Flood Attacks

Recall (from Chapter 31) that in order to initiate a TCP session, the client first sends a SYN packet to the server, in response to which the server replies with a SYN/ACK packet. This handshake is concluded with

the client sending a concluding ACK packet to the server, but if the server does not receive the ACK packet, it waits for a certain time-out period before discarding the session.

In a SYN flooding attack, the attacker sends a large number of SYN packets to the server, ignores the SYN/ACK replies, and never sends the ACK response. In fact, an attacker will usually use a spoofed IP source address in the SYN packets, so any SYN/ACK replies are sent to random IP addresses. Therefore, if the attacker sends a large number of SYN packets with no corresponding ACK packets, the server's memory will fill up with sequence numbers that it is forced to remember in order to match up TCP sessions with the expected ACK packets. Since these ACK packets never actually arrive, this wasted memory will ultimately block out other, legitimate TCP session requests.

Essentially, if the attacker sends a large volume of SYN packets to the server, the server is forced to send SYN/ACK packets back to the "client" and has to remember the sequence numbers of each of the packets for when the connections are established. However, if the attacker does not complete the handshake by sending the ACK packet, the server has wasted a lot of memory by being forced to remember all the sequence numbers for connections that will never actually happen, thus using up all of the server's bandwidth and preventing legitimate connections from taking place.

There are a couple of possible defenses for SYN flooding. The first is a process known as overprovisioning wherein we ensure that the server has a lot of memory. However, this can be pretty expensive and usually can still be circumvented depending on your threat model. Perhaps a more stable defense is to filter packets to ensure that only legitimate connections will create state, through the use of SYN cookies. In an ideal scenario, the server generates state for the client but does not save it when it sends the SYN/ACK flag; instead, it sends the state to the client encoded with a secret. It is then up to the client to store the state on behalf of the server and return the state in the corresponding ACK packet. Only when the handshake is complete will the server allocate state for the connection after checking the cookie against the secret. The issue, however, is that TCP does not have the mechanism to store state. Thus, instead, the server generates state for the client when it generates the SYN/ACK flag and does not save it; instead, it encodes the state within the sequence number with a secret. The client remembers the sequence number and returns it in the corresponding ACK number. Only when the handshake is complete will the server allocate state for the connection after checking the cookie against the secret.

Essentially, what is happening here is that the server does not create state until the handshake is completed, so the attacker cannot spoof source addresses.

34.4. Distributed Denial of Service (DDoS)

Today, most standard DoS attacks are impractical to execute from a single machine. Modern server technology allows websites to handle an enormous amount of bandwidth, much greater than the bandwidth that is possible from a single machine. However, DoS conditions can still be created by using multiple attacking machines in what is known as a Distributed Denial of Service (DDoS) attack. Here, malicious user(s) leverage the power of many machines (the number of machines could be in the thousands) to direct traffic against a single website in an attempt to create DoS conditions (i.e. prevent availability). Often, attackers carry out DDoS attacks by using botnets, a series of large networks of machines that have been compromised and are controllable remotely.

Theoretically, there is no way to completely eliminate the possibility of a DDoS attack since the bandwidth that a server is able to provide its users is always going to be limited. However, measures can still be taken to mitigate the risks of DDoS attacks. For example, several servers incorporate DDoS protection mechanisms that analyze incoming traffic and drop packets from sources that are consuming too much bandwidth. Unfortunately, IP spoofing makes this defense extremely difficult by obscuring the identity of the attacker bots and providing inconsistent information on where network traffic is coming from.