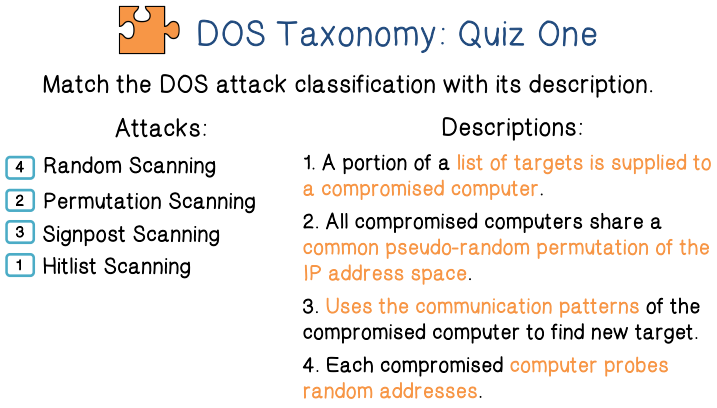
Recommended Reading for this lesson:

i. The DDOS that almost Broke the Internet

You can find the summary of the paper at the end of the document.

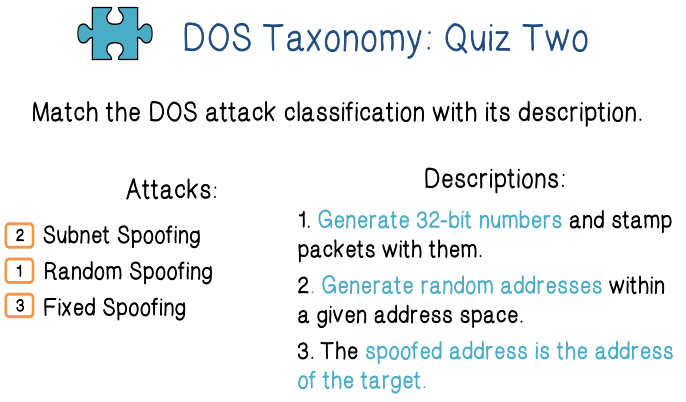
We begin our discussion of network security with an attack that all of us are familiar with. Large scale attacks. Specifically, distributed Denial-of-Service attacks, and malware based attacks. We will discuss the methods used and the security weaknesses that are exploited in these attacks.

Let's do a quick review of the Denial-of-Service attacks. Match the Denial-of-Service attack classification with its description.



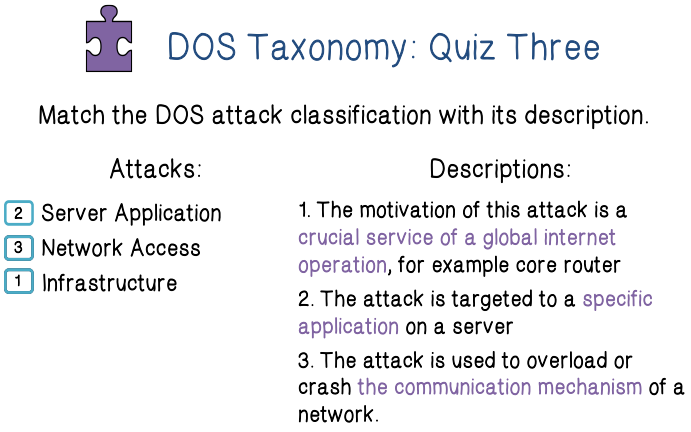
For random scanning, it means that each compromised computer probes random addresses. For permutation scanning, all compromised computers share a common pseudo-random permutation of the IP address space. Signpost scanning uses the communication patterns of the compromised computer to find new targets. In hitlist scanning, a portion of a list of targets is supplied to a compromised computer.

Now let's do another quiz. Again, match the Denial-of-Service attack classification with its description.



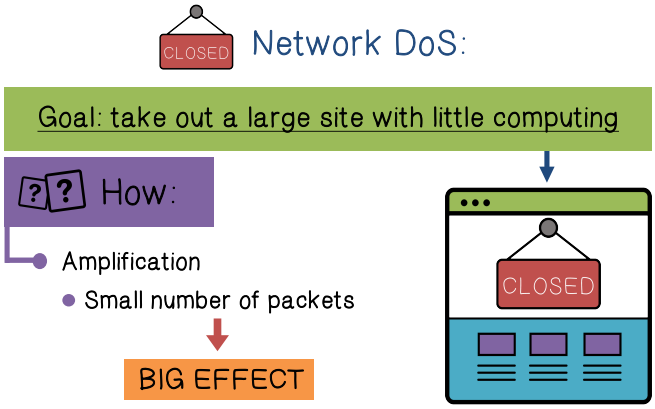
Subnet spoofing means that the spoofed address is within a given address space. Random spoofing means that the spoofed address is a randomly generated 32-bit address. Fixed spoofing means that the spoofed address is the address of a target.

Here's another quiz. Again, let's

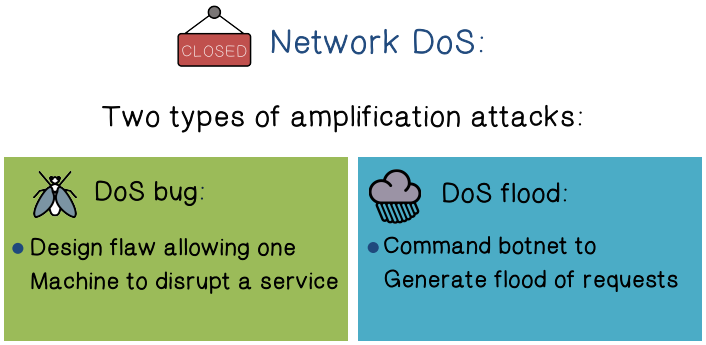


match the Denial-of-Service attack classification with this description.

If the Denial-of-Service attack targets server application, it means that the target is a specific application on a target server. If the target is Network Access, that means the attack is used to overload or crash the communication mechanism of a network, so that network access is now unavailable. If the target is Infrastructure, that means the attack is aimed at the crucial services of the global Internet, for example, the core routers.

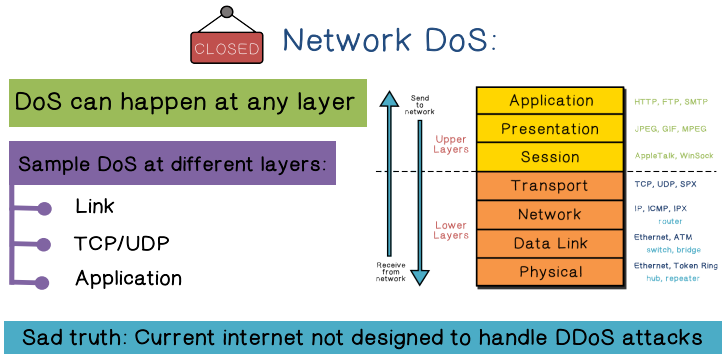


Now let's discuss network Denial-of-Service attacks in some more details. The goal of a network Denial-of-Service attack is to take out a large site such as a web server with as little computing power as possible. So how is network Denial-of-Service accomplished? One of the main approaches is amplification. This means that the attacker only needs to send a small number of packets and can achieve a big effect such as rendering the targeted site unavailable.

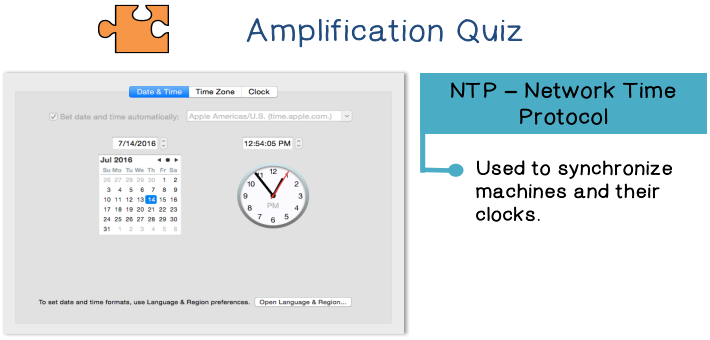


There are two types of amplification attacks. The first type is to exploit a bug or vulnerability on the server. For example, if there's a design flaw or implementation error on the server code, then the attacker machine can send a few packets that contain input that would trigger the bug and then crash the server.

And of course, when the server program is crashed, then the server become unavailable. Another type of Denial-of-Service attack is to send a flood of packets. For example, an attacker can use a large botnet to send a lot of requests to the server.



Network Denial-of-Service attacks can happen at any network layer. As a quick review, there are multiple layers in a network stack. For example, Denial-of-Service attacks can happen at the link layer. This means that the attacker simply sends a lot of traffic to saturate the link. Denial-of-Service attacks can happen at the TCP/UDP layer, or, the transport layer. For example, the server needs to use memory to hold the state of TCP connections, so the attacker can send a lot of TCP packets to exhaust the server's memory. Denial-of-Service attacks can also happen at the application layer. For example, the attacker can request the server application to fetch a large amount of data. And if there are many such requests, the server's resources will be exhausted. The sad truth is that the current Internet design cannot handle Distributed Denial-of-Service (DDoS) attacks.



Now let's go over an application example and then do a quiz. One example of attacks is to use the NTP, the Network Time Protocol. This protocol is used to synchronize computers' clocks. For example, on a Mac, the day and time are set by an NTP server run by Apple. When a computer requests the time from the NTP server, the server responds with the correct time and date. In NTP, the data volume of the request from a machine is much smaller than the response from the server. Now you can imagine how NTP can be used for a Denial-of-Service attack.

So with that background, let's do a quiz. Which of these are reasons why the UDP-based NTP protocol is particularly vulnerable to amplification attacks?

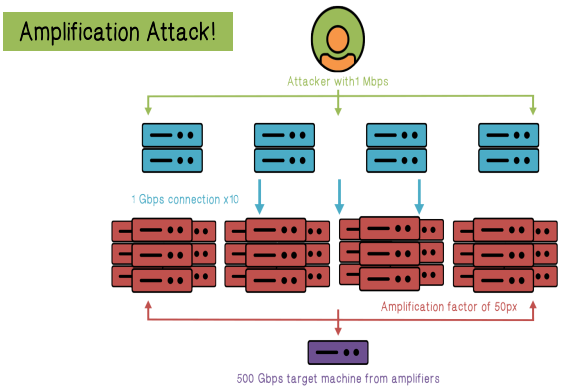
Amplification Quiz

Which of these are reasons why the UDP-based NTP protocol is particularly vulnerable to amplification attacks?
Select all that are true.

- [x] A small command can generate a large response.
- [x] Vulnerable to source IP spoofing.
- [x] It is difficult to ensure computers communicate only with legitimate NTP severs.

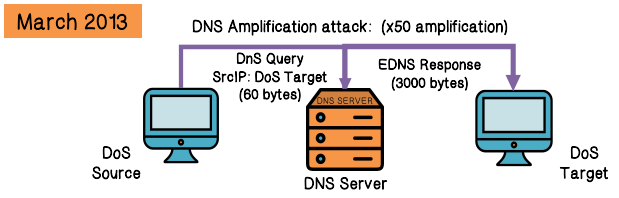
As we already discussed, the volume of request from a computer is smaller than the response from the server. That means a small command can generate a large response. And the attack works because the attacker can send the request to a server by spoofing the IP address of the target. So that the server's response is sent to the target and that's how the attack works. And it is difficult to ensure that the computers only communicate with legitimate NTP servers. This means that it is not easy to figure out which are the legitimate responses from NTP servers.

Amplification Example



Let's take a closer look at amplification attacks. Typically, the attacker uses a machine, and then this attacker machine controls a number of bots, or compromised computers. And each of these bots will send many requests to a server, and the response from the server is much larger than the request. So the amplification is accomplished by two factors. One is the number of bots involved. And second, the server response is much larger than the request.

Amplification Example



2006: 0.58M open resolvers on Internet *(Kaminsky-Shiffman)*

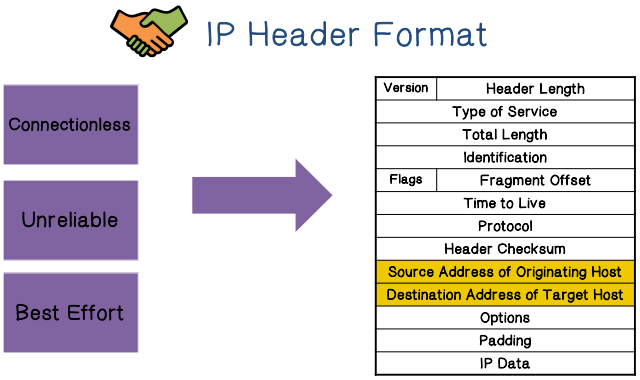
2014: 28M open resolvers *(openresolverproject.org)*

March 2013: DDoS attack generating 309 Gbps for 28 mins

Here's a specific example of amplification attack. This involves DNS, the domain name system. Here's the amplification factor involving the server, the DNS server. So here we have the machine sending a DNS request to the server. And of course, the address, the source of the DNS query is spoofed and the server thinks that the request is from the target. The server response is much larger than the

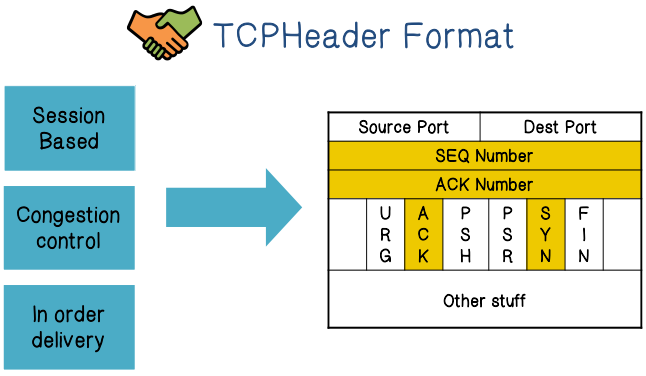
request. In this case, it is 50 times. Here, EDNS means extension mechanism for DNS. It allows for actual flags and response data. Therefore, the response is much larger.

In a DNS-based amplification attack, each of the bots controlled by the attacker will send many requests to any of the DNS resolvers. And there are many of them. And for each request, the response will be sent to the target because in each request, the source IP address is spoofed. The attacker can choose any subset of the DNS servers to use because there are so many open DNS resolvers on the Internet. This attack can generate a huge amount of traffic in a very short period of time. For example, the attacker can easily generate tens or even hundreds of gigabits per second traffic targeted at a victim.

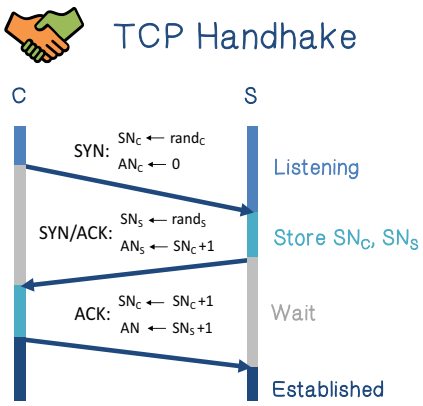


Now let's take a look at the network protocols to understand why the Internet is vulnerable to DoS attacks. So the Internet protocol or IP is connectionless. This means that it is not reliable, meaning that each packet will find its way to destination and there is no mechanism to ensure that all packets will arrive properly and in sequence at least not at the IP layer. Essentially it is best-effort delivery.

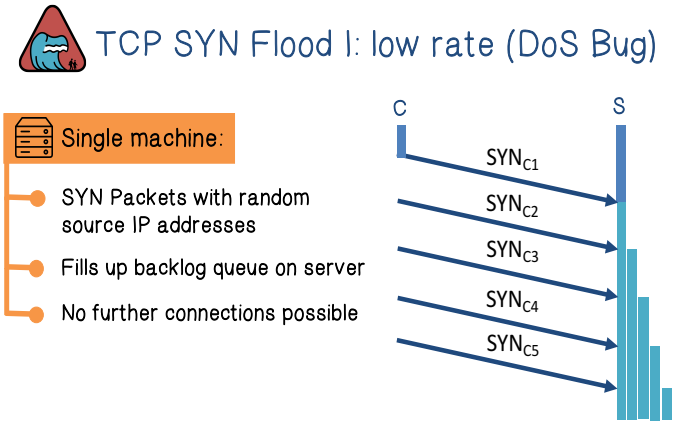
So here is the format of the IP header. For the purpose of our discussion, let's focus on a source IP address and a destination IP address. From the security point of view, the main weakness of IP is that there is no authentication of the source IP address. This means that the attacker can spoof an IP source address.



Now let's take a look at TCP. TCP is session-based, which means the destination is going to make sure that all packets belonging to a same connection will arrive and properly sequenced. And in order to achieve this, there are congestion control and in-order delivery mechanisms. These mechanisms ensure that the data loss or packet loss is minimized and the need to retransmit packets is also minimized. And here is the format of the TCP header. Notice that we use a sequence number for each packet, acknowledgement number to acknowledge a packet as received, and a number flags to actually keep the state of the session.



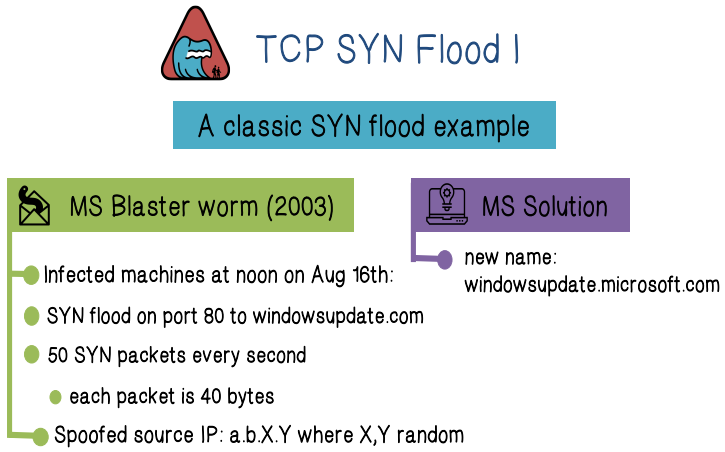
Now let us take a look at the TCP handshake or the steps to establish a TCP connection. Suppose our client wants to connect to a server. It first sends a SYN packet, this packet has a SYN flag set and also a sequence number. The acknowledgement number is 0 because this is the first packet. The server responds with a SYN/ACK packet, which means that both the SYN flag and the ACK flag are set. Its sequence number is a server sequence number and its acknowledgment number is the client sequence number plus one. This means that this SYN/ACK packet is an acknowledgement of the initial SYN packet from the client. And then the client sends a final ACK packet to the server. In this ACK packet, it increments its own sequence number and acknowledge the sequence number from the server. This tells the server that the client has received this SYN/ACK packet. At this point, the TCP connection is established.



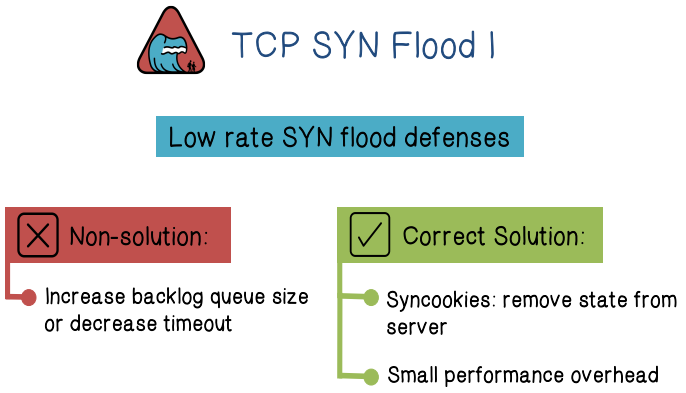
With that background, let us discuss how TCP SYN flood for Denial-of-Service attack can work. Notice that, in TCP handshake, after the server receives a SYN packet from the client, it sends a SYN/ACK packet back to the client, and then waits for the ACK packet from the client. When it receives the ACK packet, it knows that the connection is established. Therefore, the server needs to keep in memory the state of the connection, meaning that it is waiting for the ACK packet that matches the SYN/ACK packet, which matches the initial SYN packet from client. So SYN flood exploits the fact that the server needs to keep in memory such state information.

In particular, the attacker can send a lot of SYN packets to the server, and the source IP address is spoofed to some random target source IP address. The result is that the SYN/ACK packet will be sent to the spoofed or the target address. Since the source IP address of these SYN packets are randomly generated and spoofed, the SYN/ACK packets may get lost, meaning that the ACK packets of the SYN/ACK packets may never arrive at a server. The result is that the server's

memory gets filled up, because the server needs to keep track of the SYN/ACK packets and wait for the ACK packets from the clients. And since many of these ACK packets do not arrive, the server is holding in memory this state information. And as a result, its buffer gets filled up. And when that happens, no further connections can be serviced. In other words, Denial-of-Service is accomplished.

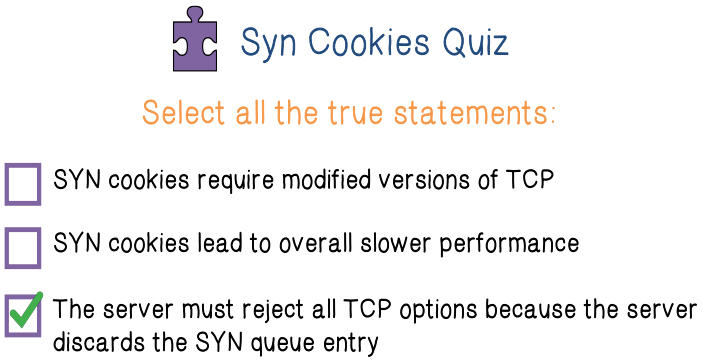


Here is a real example of SYN flood. The Blaster worm in 2003 infected many machines. And these infected machines were instructed to launch a Denial-of-Service attack at noon on August 16th. That is, these machines were instructed to launch a SYN flood on port 80 on the target server windowsupdate.com. In particular, 50 SYN requests were sent every second. And each packet is 40 bytes. And the source IP address of these request packets were randomly generated. As a result, the server windowsupdate.com was rendered unavailable. As a response, Microsoft moved the Windows update service to a new domain, windowsupdate.microsoft.com.

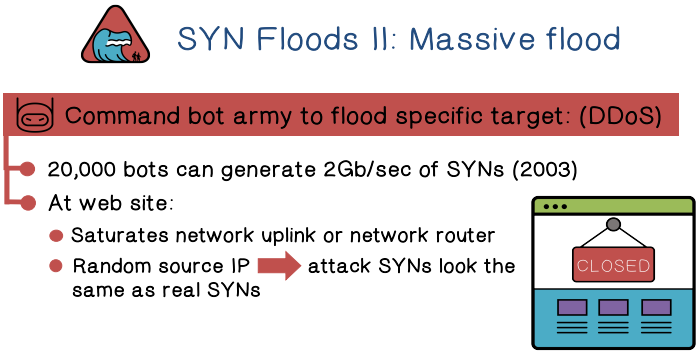


So how do we defend against SYN flood attacks? How about increasing the memory size or decreasing the timeout value so that when a server does not receive an ACK packet, it just clears out the memory. These are not good solutions because an attacker can just send more packets or at a faster pace. A better solution is to remove the need for a server to keep state. And this, of course, comes with a cost.

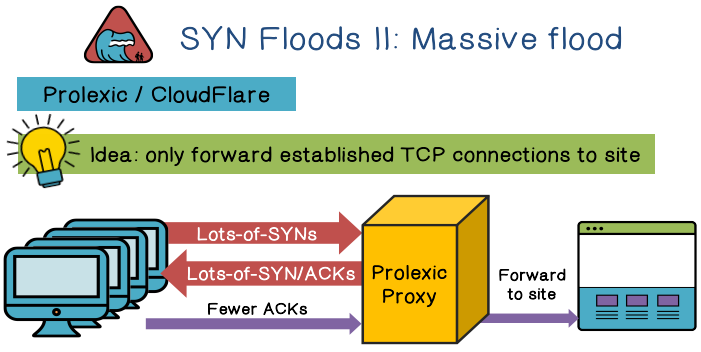
Now let us do a quiz on SYN cookies. Select all true statements.



SYN cookies does not require a modified version of TCP, so the first is false. SYN cookies are only applied when there is a SYN flood attack. That is, during normal operations, or when a server does not experience an overload, it does not require SYN cookies. Therefore, SYN cookies should not lead to overall slower performance, that is, the second statement is false. The third statement is true because during an attack, the server uses SYN cookies and does not keep state information in memory.



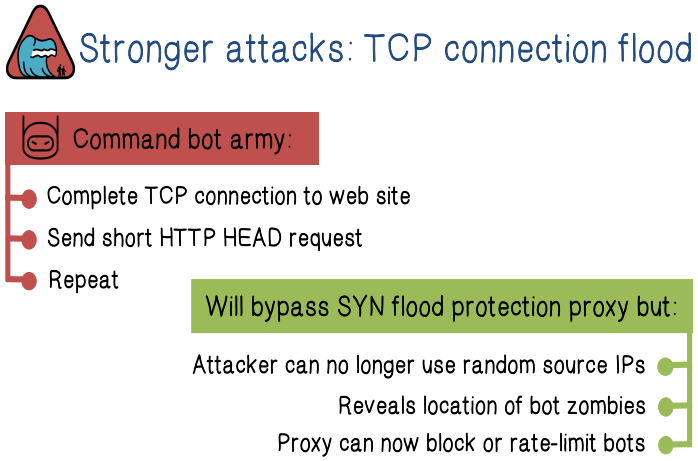
SYN flood attacks can be launched at a massive scale. Typically, for a Distributed Denial-of-Service (DDoS) attack, a large botnet can be used to generate a huge amount of traffic. And the result is that the website, or even its uplink network routers, can be saturated. It is very hard to filter these SYN packets, because they all look legitimate.



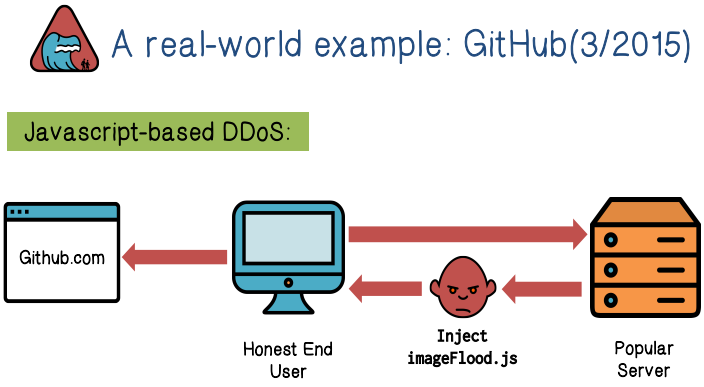
So how do we defend against such massive flooding attack?

One idea is to use a very powerful server, or a group of servers, to protect a website. The idea is that these intermediate servers will only forward established TCP connections to the real website. Suppose many machines or bots send a lot of requests to the website, but they are intercepted by the proxy.

The proxy is very powerful because it can use many servers. And they can be distributed across the Internet. The proxy sends the SYN/ACK packets in response to the initial SYN packets. When a proxy receives the ACK packets from the client, it will then forward to the real website. The idea here is that the attacking machine or the bot will not send actual ACK packets to the proxy. Only the legitimate clients will send the ACK packets to the proxy, and only those will be forwarded to the website to be serviced. In other words, the proxy here stops the flooding attack.



However, the idea of using a proxy to protect a website is not bulletproof. Here's an example of a stronger attack. An attacker can use an army of bots to actually completely finish the TCP handshake. In other words, they use complete TCP connections to the target a website. That is, they can send requests to the website and keep repeating such requests. All of these requests are legitimate from a protocol point of view, but they were designed to overload the web server with a lot of work. And the result is that if an attacker can command a huge army of bots, the attacker can still bring down a website. This is similar to the situation when there are huge number of legitimate users visiting a website at the same time. Of course, such an attack can actually render the proxy protection useless. On the other hand, because the TCP connection is fully established, that means the attacker cannot use any random source IP address. The attacker must use the real IP address of the bots, which means that the bots' IP addresses are now revealed. And then a proxy can actually block or rate limit traffic from these bots. In other words, after the initial attack, there is a chance that the proxy can actually use the information to rate-limit and then reduce the effect of the flooding attack.



Here is a real-world example of such an attack, it is fairly recent. So here, an honest end-user visits a popular website, but this website is compromised and the response will include a malicious JavaScript injected into the response. And the user has no idea that this JavaScript is embedded into the response

HTML page. For example, this JavaScript can be embedded in an invisible iFrame. Once the response HTML page runs on the user's browser, the malicious JavaScript will run, and it will do a Denial-of-Service attack on a server, say, github.com.

A real-world example: GitHub(3/2015)

imageFlood.js

```
Function imgflood() {
 var TARGET = 'victim-website.com/index.php?'
 var rand = Math.floor(Math.random() * 1000)
 var pic = new Image()
Pic.src = 'http://'+TARGET+rand+'=val'
}
setInterval(imgflood,10)
```

Here is how the JavaScript can launch an attack on github.com. It basically asks the victim website, say, github.com, to fetch a random resource on its server. And it sends such a request every ten milliseconds. Therefore, with many users unknowingly running this malicious JavaScript, the victim website, say github.com, can be rendered unavailable.

Flood Attack Quiz

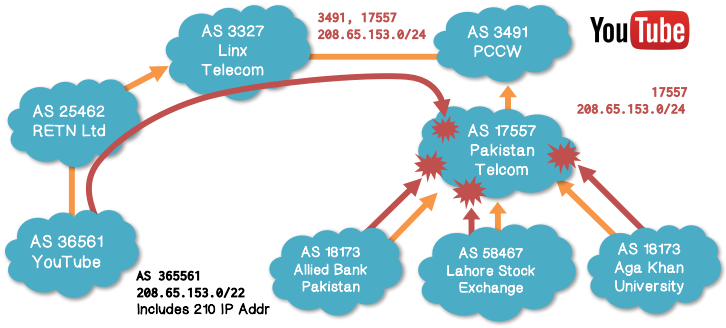
With regards to a UDP flood attack, which of the following statements are true:

- [x] Attackers can spoof the IP address of their UDP packets
- [ ] The attack can be mitigated using firewalls
- [x] Firewalls cannot stop a flood because the firewall is susceptible to flooding.

Now let us do a quiz on flooding attack. With regards to a UDP-based flooding attack, which of the following statements are true?

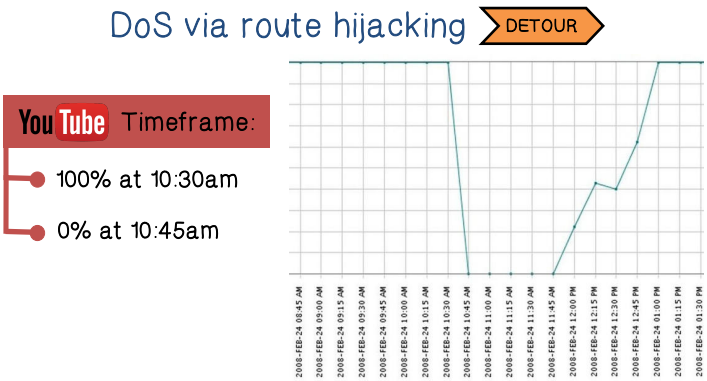
Attackers can obviously spoof the IP address, so the first is true. The attack cannot be mitigated using firewalls because the packets involved in the attack all look legitimate, so the second is false. In addition, even if the firewall is attempting to do filtering, it itself is susceptible to flooding. The reason is that the firewall now needs to examine many, many packets. So, the third is true.

DoS via route hijacking

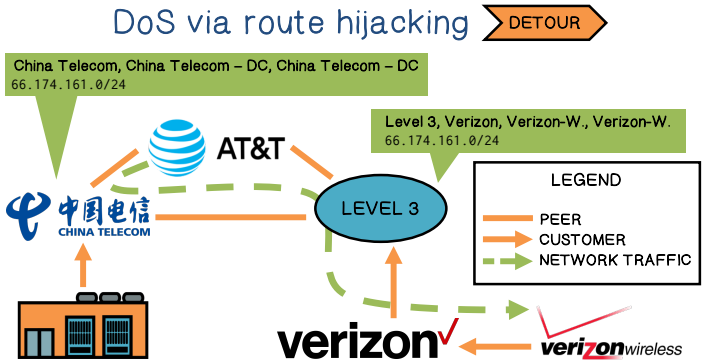


The Internet routing protocol can also be exploited to launch Denial-of-Service attacks. In particular, there have been several incidents of route hijacking that resulted in Denial-of-Service. Here is one example involving Pakistan and YouTube. The Internet is divided into a large number of so-called autonomous systems. Each autonomous system, or, AS, is responsible for routing packets in and out of a subset of the Internet defined by the prefix.

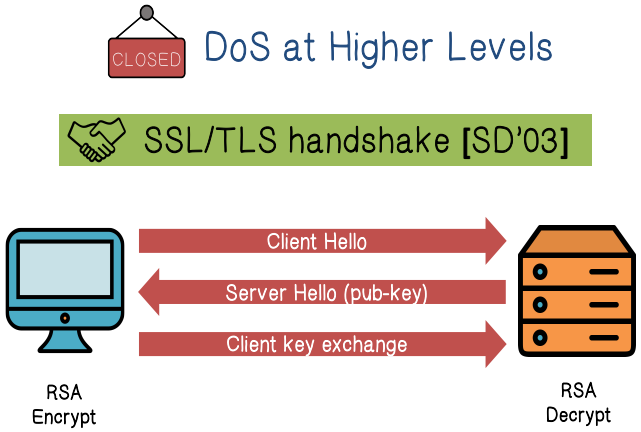
For example, the YouTube service address is within this prefix. It is actually 208.65.103.238. In February 2008, Pakistan Telecom advertised that, it is actually responsible for the subset of Internet defined by this particular prefix. And this prefix is more specific than a segment that includes the YouTube server. And since the routing decisions for a specific IP address, such as the YouTube server, use the more specific prefix, the whole Internet thought that Pakistan Telecom is responsible for routing traffic to YouTube. The result of this route hijacking is that all traffic to YouTube was instantly routed to Pakistan.



As you can see, the traffic volume at the YouTube server fell to zero until the route hijacking mistake was corrected.

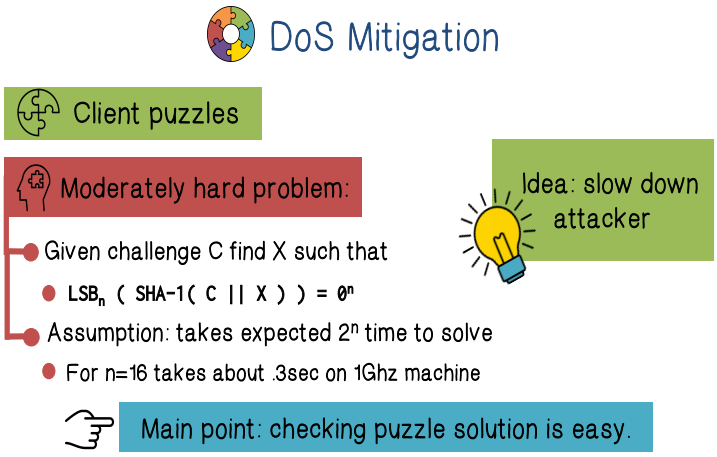


In a more recent example, China Telecom published BGP routes that caused .mil and .gov traffic to route through China Telecom.



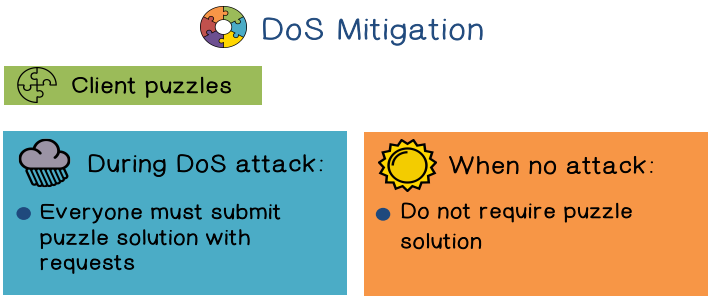
So far, we have discussed Denial-of-Service attacks that exploit weaknesses in network protocols. Denial-of-Service attacks can also happen at a higher level. For example, let's look at a typical handshake protocol. Here's a protocol that use public-key based authentication. The client sends a hello message to the server, and a server sends its public key to the client. And then, the client will use that public key to perform key exchange. For example, the client can generate a shared secret key between the client and the server, and encrypt that using the server's public key. And when the server receives this encrypted key, it will use its private key to decrypt, i.e., to extract this shared secret key. The point is that the client encrypts the shared secret key using the server's public key, and then the server decrypts that using its private key. It's all good from a crypto point of view.

However, RSA decryption is ten times costlier than RSA encryption because the server has to do so much more work. The attacker can send many such handshake requests to the server to bring it down. Similarly, at the application level, a client can send a simple HTTP request to your server asking for a very large PDF file. And obviously, the server needs to spend far more resources than the client. Therefore, an attacker can send many such HTTP requests to your server, causing the server to fetch a large number of very large PDF files, and this will actually bring down the web server.

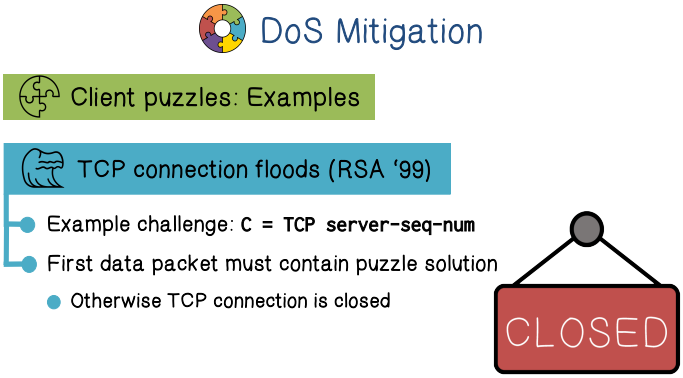


So how do we mitigate such denial of service attacks. One solution is to use client puzzles. The main idea is to slow down the attacker. For example, we can ask the client to solve a problem. For example, the server can send challenge C to the client and ask the client to find or compute X such that the least significant bits of the SHA-1 hash are all 0s. The assumption here is that it would take the client 2^n times of hashing to solve this challenge.

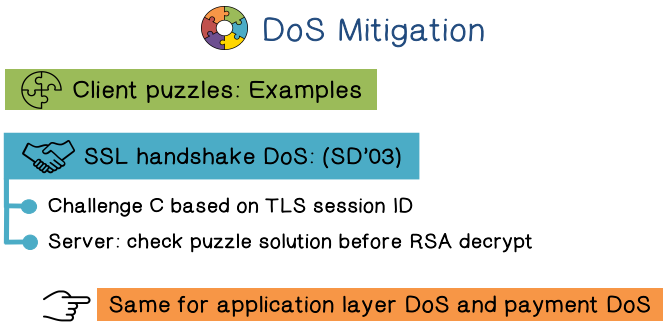
For n=16, it would take 0.3 seconds on a 1 gigahertz machine. And of course, the client needs to present X back to the servers, and it is very easy for server to check that the solution is correct. This is because the server needs to only compute the hash one time, whereas the client has to compute to the 2^n times.



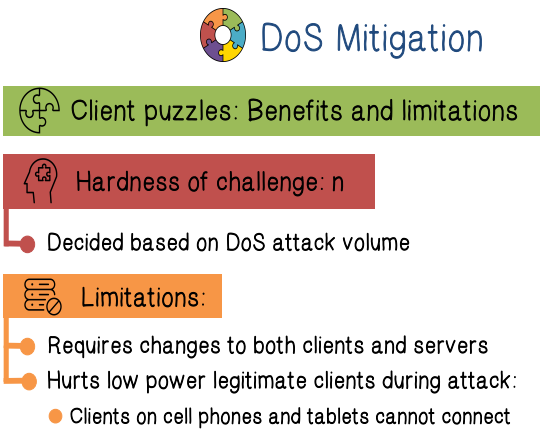
During a Denial-of-Service attack, everyone, that is, including legitimate clients or possible attackers, must submit a puzzle solution to the server. And of course, when there is no attack, no one needs to solve the puzzle.



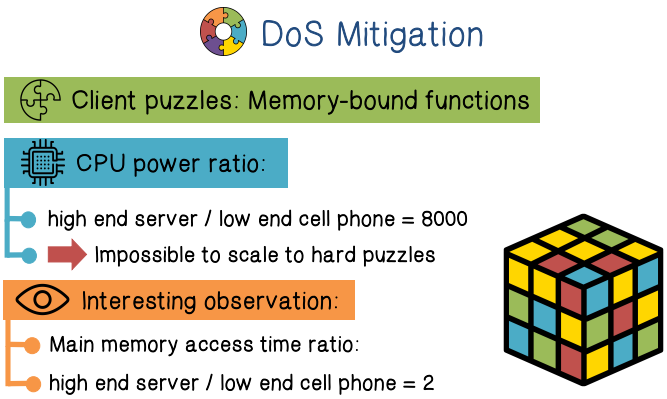
Here are some examples of how client puzzles can be deployed. For TCP connection flooding, the challenge C can be the TCP's server sequence number, and the first data packet from the client must contain the puzzle solution. Otherwise, the server will terminate the TCP connection.



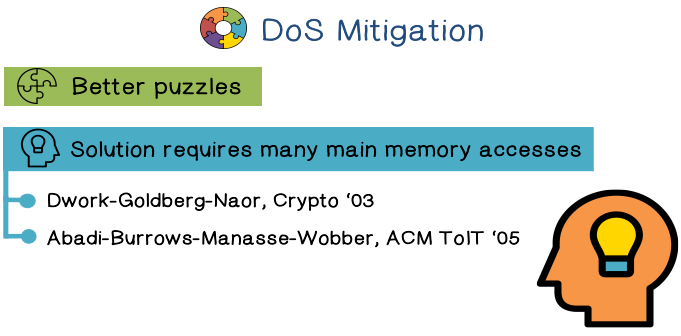
For SSL handshake DoS attack, the challenge C can be based on a TLS session ID. And the server will check the puzzle solution before even attempting to do RSA decryption, because RSA decryption is very expensive. And similar ideas can be applied to application layer DoS attacks.



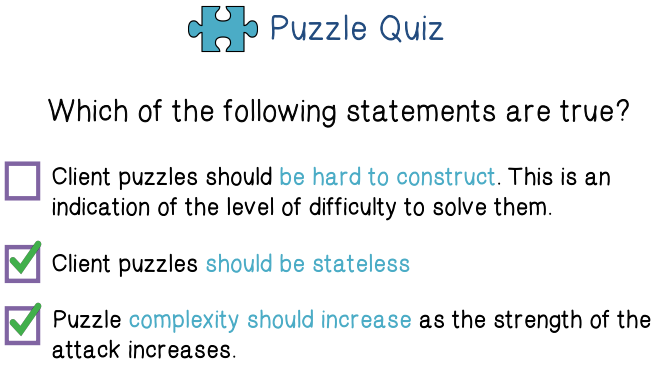
One advantage of the client puzzle is that the hardness of the challenge, or in particular, the parameter n, can be decided based on the DoS attack volume. For example, if the volume is high, you can set n to be higher so that it takes more time for the client to find a solution. In other words, this will reduce the volume of traffic to the server. The limitation is that this requires changes to both the client code and the server code. It also hurts legitimate clients, in particular, clients that use low power computing devices such as cellphones.



Another variant of client puzzle is to use memory-bound functions. This is because CPU-bound functions such as the one we just discussed, cannot be scaled to very hard puzzles for low end machines such as cellphones; whereas memory-bound functions can be easily scaled, even for low-end machines.

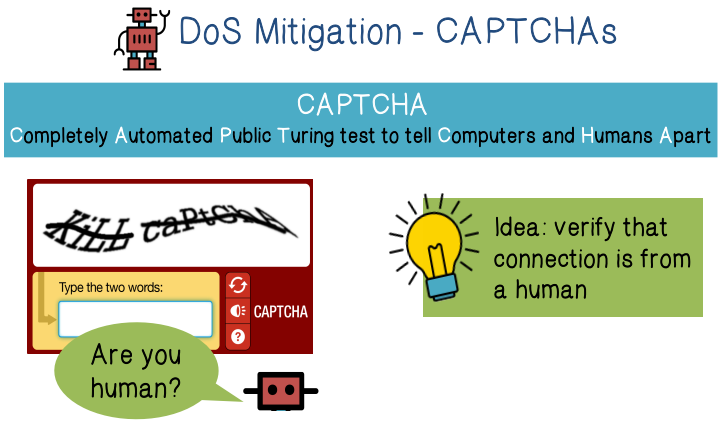


There are several proposals to use memory-bound functions as puzzles. You are encouraged to study these papers.

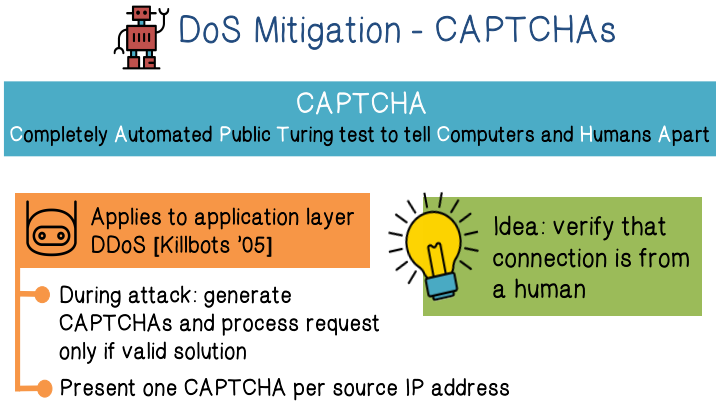


Now let us do a quiz on client puzzles. Which of the following statements are true?

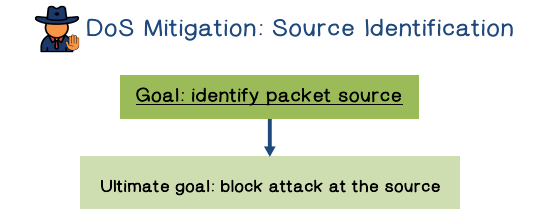
Client puzzles should not be hard for a server to construct, so the first is false. Client puzzles should be stateless, this will keep a client from being able to guess the puzzle and find a solution before even being asked. Puzzle complexity should increase as the attack volume increases. So both the second and third statements are true.



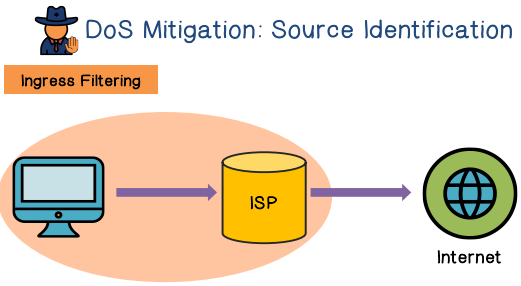
Let us discuss another mitigation technique. You may be already familiar with this. It is called CAPTCHA, which stands for Completely Automated Public Turing test to tell Computers and Humans Apart. The idea is that the server should verify that the connection is from a human instead of, for example, a bot or a malware. So I'm sure many of you are familiar with this. The idea is that only human can interpret this figure and then type in the actual words.



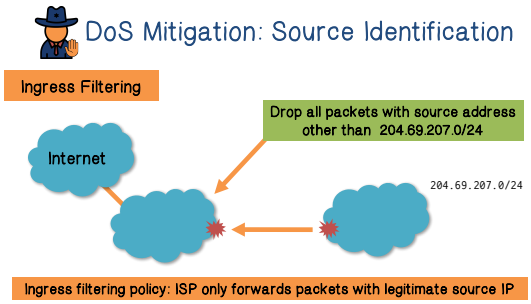
During a DoS attack, the server can generate CAPTCHAs and process a request only if the client presents the actual valid solution to the CAPTCHA challenge, because that will prove that there is actual human behind the request.



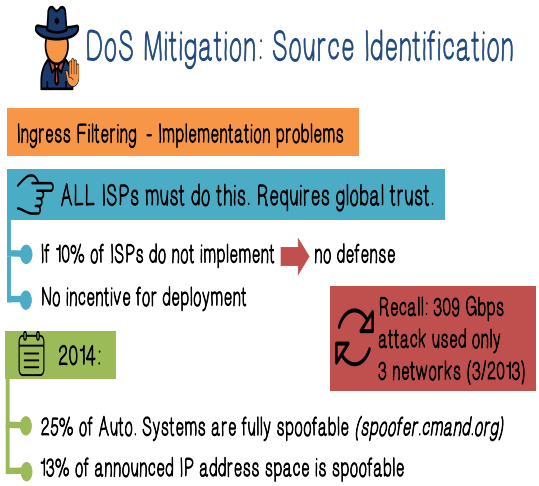
Another important mitigation technique is source identification. The goal is to identify the source of attack packets so that ultimately, we can block the attack at each source.



You may think that this problem should be easy because, for example, we discussed that many of the packets involved in a DoS attack have spoofed or random source IP addresses. So the question is, why don't we just ask the ISPs to filter out source IP addresses that are not legitimate or valid for the ISP ?



For example, if the router expects that all traffic is from this particular prefix, then it can drop all packets with a source IP address other than from this prefix. That way spoofed packets can be dropped.



The biggest problem for this proposal is that it requires all ISPs to do this. Because, as we will show, if only 10% of ISPs do not implement this, then there is actually no defense against denial-of-service for the whole Internet. Then the problem becomes that every ISP is waiting for everyone else to implement this first. As we have shown in the previous example, if only 3 ISPs do not implement ingress filtering, the attackers can already launch a big DoS attack. As of 2014, a quarter of the autonomous systems are mostly ISPs or big enterprises. They do not implement ingress filtering. In total, that means 13% of the IP addresses can be spoofed.

DoS Mitigation: Traceback

Traceback [Savage et al. '00]

Goal:
- Given set of attack packets
- Determine path to source

How: change routers to record info in packets

Assumptions:
- Most routers remain uncompromised
- Attacker sends many packets
- Route from attacker to victim remains relatively stable

Now let us discuss another source identification technique called Traceback. The goal is that given a set of attack packets, we want to determine the paths of these packets and use a path to determine the source of these packets. And the way to do this is to change the Internet routers to record some path information into the packet. There are a few assumptions here. First, most the routers remain uncompromised, meaning that these routers can record information faithfully. Second, the attackers send many packets, and the route from the attackers source to the victim remains relatively stable.

DoS Mitigation: Traceback

Simple Method:

Write path into network packet:
- Each router adds its own IP address to packet
- Victim reads path from packet

Problems:
- Requires space in packet
- Path can be long
- No extra fields in current IP format
  - Changes to packet format too much to expect

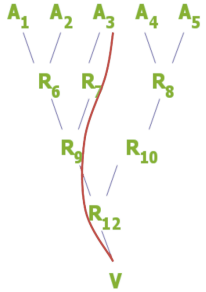
Here is a naive strawman method. We can have each router write its own IP address in the packet. So, at the end, the victim can read a path from the packet because each router has written its own IP address. The problem with this is that it requires space in packet and this can be a problem when a path is long. There is no extra fields in the current IP format to record this whole path information. If we expect the packet format to be changed to include this path information it would take years, if ever, to get this implemented.

DoS Mitigation: Traceback

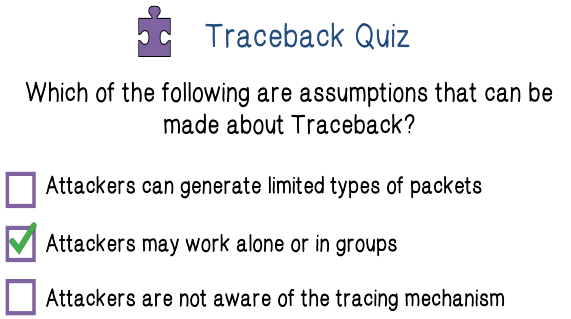
Better Idea

DDoS involves many packets on same path

Store one link in each packet
- Each router probabilistically stores own address
- Fixed space regardless of path length

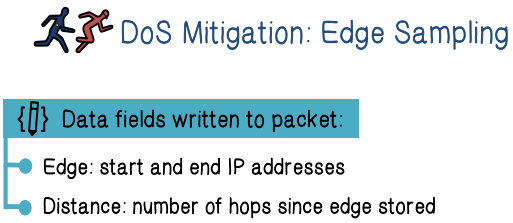


So here is a better idea. We observe that if DoS attacks involve many, many packets on the same paths, we can have each router take a probability to store its own address in a packet. This required only a fixed amount of space regardless of the path length.

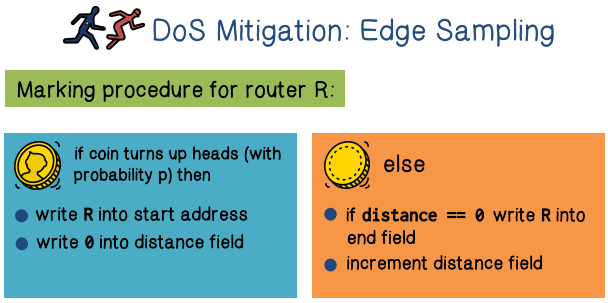


Let us do a quiz on Traceback. Which of the following are assumptions that can be made about Traceback?

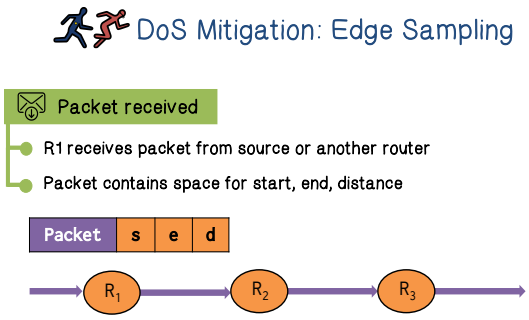
Attackers can generate unlimited types of packets, so the first statement is false. Attackers can indeed work alone or in groups, so the second statement is true. And the Traceback will work regardless of whether the attackers are aware of the tracing mechanisms or not, so the third statement is false.



So now, let us go into the detail of the traceback mechanism. The main component is the edge sampling algorithm. An edge includes the start and end IP addresses. It also includes distance, which is the number of hops since the last edge stored.

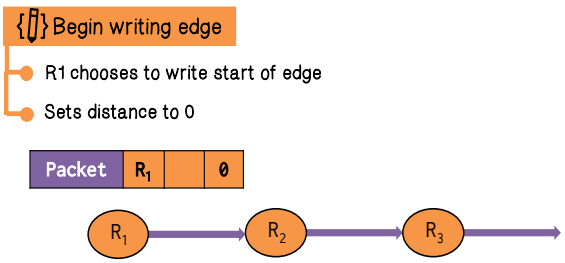


Here is the procedure for a router to decide how to record the edge information. When a packet arrives, it throws a coin. If it is head, it will write its address into the start address and then write 0 into the distance field. If it is tail, then if distance is 0, it writes its IP address into the end address field. And regardless, it will increment the distance field.



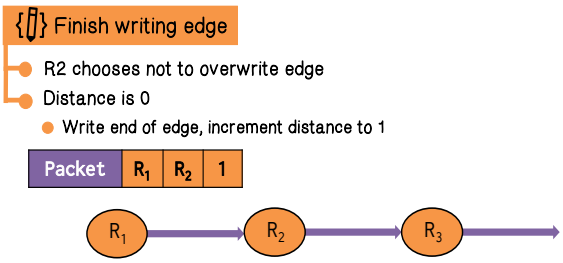
So here is an example. Again, a packet would include edge information, which includes the start address, end address, and distance. Suppose the packet travels through three routers, R_1, R_2, and R_3.

DoS Mitigation: Edge Sampling



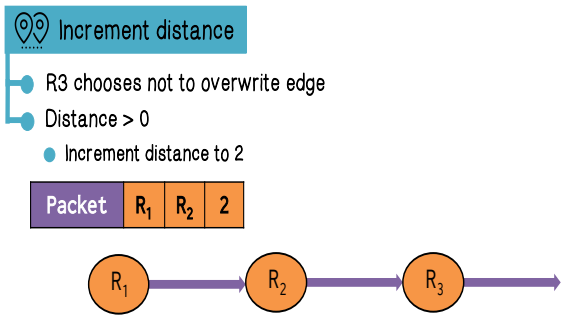
R_1 tosses a coin and it is head. So R_1 writes its address to the start field and 0 in distance.

DoS Mitigation: Edge Sampling



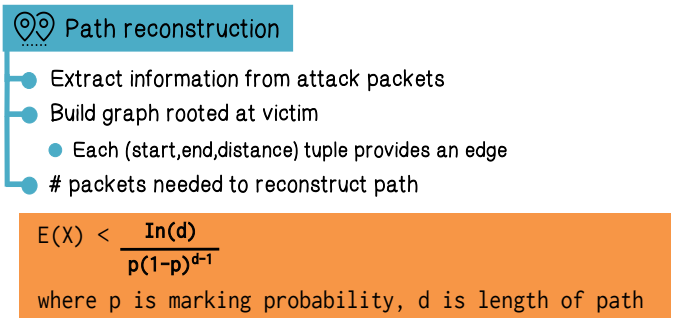
Now R_2, it tosses a coin and it is tail. The distance was 0. So according to the process, it should write itself to the end and then increment distance to 1.

DoS Mitigation: Edge Sampling



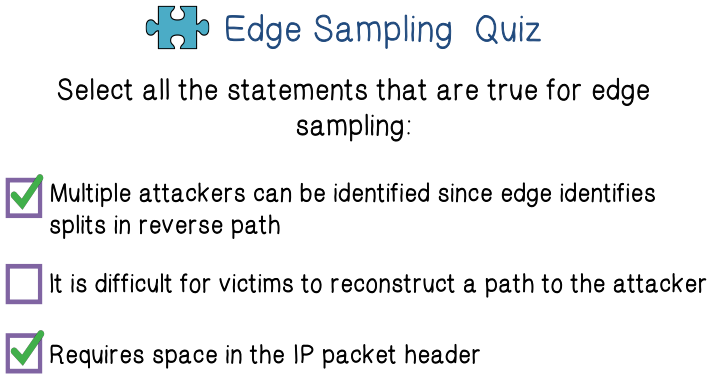
Now for R_3, it tosses the coin and it is tail again. But the distance was not 0. So, it does not write itself to the end. It simply increments the distance. Now, as you can see, the edge information includes the starting of the edge, which is R_1, the end, which is R_2, the distance is 2. That means from R_3's perspective, the distance is 2 from the beginning of the edge.

DoS Mitigation: Edge Sampling



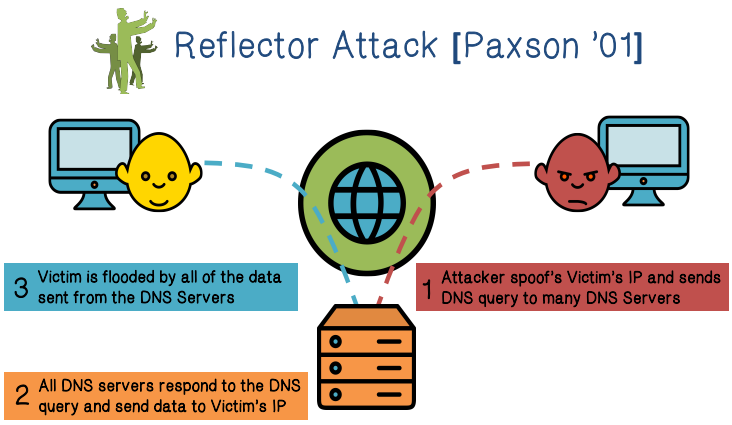
With the edge information, now we can talk about how we reconstruct the path. The packet arriving at the victim contains edge information. And this information can be extracted to reconstruct the path that starts from the victim all the way to

the source of the attack packets. Again, the edge information contains the starting router and the end router of the edge and the distance from the starting router. And the number of packets needed to reconstruct the path is given by this formula. This is the expected number of packets. And p is the probability of head versus tail, and d is the length of the path.

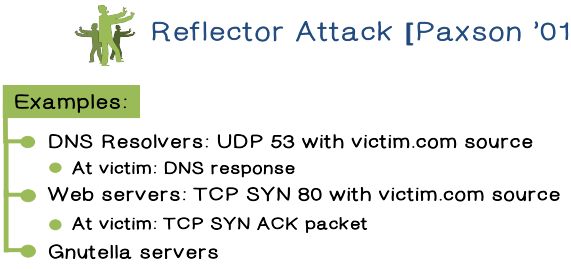


Let us do a quiz on edge sampling. Select all the statements that are true for edge sampling.

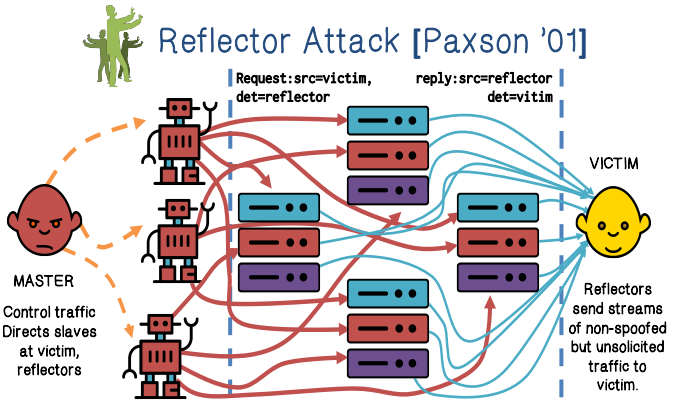
With edge sampling, multiple sources can be identified. Therefore, multiple attackers can be identified. As we can see, it is relatively easy for a victim to reconstruct a path given the edge information. So this second statement is false. The edge information is stored in the IP packet header, so therefore, the third statement is true.



Let us discuss a more recent type of DoS attacks called reflection attack. So here, the attacker spoofed the victim's source IP address and sends DNS query to many DNS servers. And all DNS servers will respond to this query and send their response to the victim machine. And, of course, the result is that the victim is flooded.



In addition to the DNS example, there are other examples that use web servers and Gnutella servers.



A reflection attack is typically launched by a bot master commanding many bots, each of them can send a lot of requests to many reflectors, such as DNS servers, web servers and the Gnutella servers. And these requests will spoof the victim IP address and as a result, the reflectors will send the response to the victim.

Since the actual flooding traffic is from the reflectors to the victim, a traceback scheme will trace the attack packets back to the reflectors. And the reflectors may not do any marking or keep any state, so there is no traceback from the reflector back to the bots or bot master.

Reflector Attack Quiz

Self defense against reflector attacks should incorporate:

- [ ] Filtering - filter DNS traffic as close to the victim as possible.
- [x] Server redundancy - servers should be located in multiple networks and locations.
- [x] Traffic limiting - traffic from a name server should be limited to reasonable thresholds.

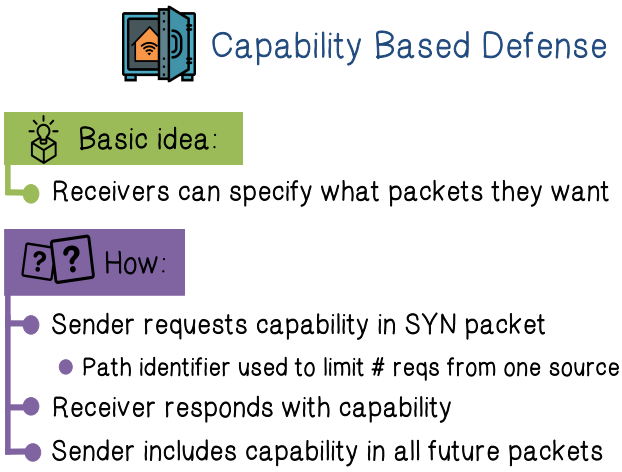
Now, let us do a quiz on reflection attack. Self-defense against reflection attacks should incorporate which of the following?

Filtering should take place as far from the victim as possible, so the first statement is false. Server redundancy is always helpful and rate limiting is always helpful.

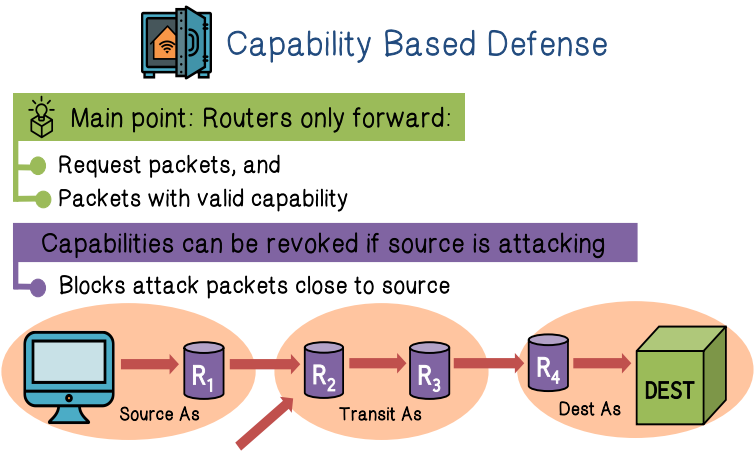
Capability Based Defense

Anderson, Roscoe, Wetherall	Preventing internet denial-of-service with capabilities. SIGCOMM '04.
Yaar, Perrig, and Song	Siff: A stateless internet flow filter to mitigate DDoS flooding attacks. IEEE S&P '04.
Yang, Wetherall, Anderson	A DoS-limiting network architecture. SIGCOMM '05

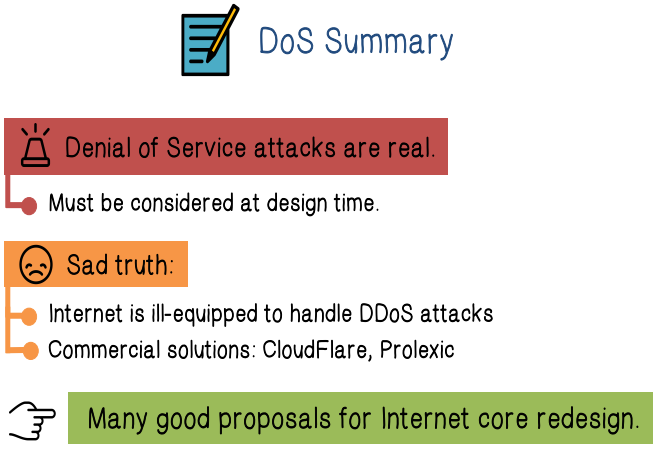
Now let us discuss some novel idea to defend against denial-of-service attacks. There are a number of examples for capability-based defenses. You are welcome to study these papers.



Here is a brief overview of these defenses. The basic idea is that the receivers, such as a server, can specify what packets they want, and this is called the capability. When the sender sends a request to the receiver, he must include capability in his SYN packet, meaning that he needs to tell the server that, hey, I am the packet that you want. When a client wants to connect to a server, it needs to first request capability. And such a request should be very limited. And the server can respond with a capability that the sender can later include in his packets.



Furthermore, all the routers will only forward packets with valid capability. If a source is attacking, then its capability can be revoked, and as a result, the routers will drop or block such packets. And this can take place as close to the source as possible.



So, in summary, DoS attack is a real and present danger on the Internet and to mitigate such attacks, security should have been considered at Internet design time. Therefore, the sad truth is that the current Internet is ill-equipped to handle DoS attacks. There are some commercial solutions. There are many good proposals for Internet core redesign such as capability based routing.

Further Readings

The DDOS that Almost Broke the Internet

Resource link: https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/

The goal of this reading is to understand the impact of a DNS amplification attack, which made use of the vulnerability of open DNS recursors (recursive servers).

Summary of the attack:

In March 2013 a large DDoS attack was launched by attackers against CloudFlare and their network providers. The attack was first started on spamhaus.org and later targeted CloudFlare's network providers, affecting primarily Europe. The volume of data peaked at 120 Gbps of traffic.

Structure of the Internet:

In the Internet, a number of networks (run by CloudFlare, Google, AT&T, etc.) interconnect through peering relationships, which can be observed by tracing packet routes with the traceroute command. CloudFlare connects to peers directly through routers as well as through Internet Exchanges, or IXs. These are central points where a number of major networks meet and can then pass bandwidth. Apart from peering relationships, providers contract with higher providers to purchase paid bandwidth. CloudFlare purchases bandwidth from autonomous systems known as Tier 2 providers, which in turn peer with other Tier 2 providers and also buy bandwidth from Tier 1 providers. Tier 1 providers don't buy bandwidth from anyone, and only engage in settlement-free pairing with other Tier 1 providers.

At the core of the Internet, Tier 1 providers ensure that networks are interconnected and would break the Internet if they failed. The DDoS attack was pushed up the stream to Tier 1 providers. One major provider experienced more than 300Gbps of attack traffic related to this attack, making it one of the largest ever reported. CloudFlare uses Anycast technology, which spreads the load of a distributed attack against all its data centers, which allowed it to mitigate the attack. An attack at one server would be spread across the network instead of being concentrated at one point.

Attacks on the IXs:

The attackers also attacked the core IX infrastructure on the London Internet Exchange (LINX)

and impacted many others. As problems were detected on the IX, traffic was routed by CloudFlare around them. The attacks exposed some vulnerabilities in the architecture of some IXs. Two suggestions to limit attacks like this involve making it more difficult to attack IP addresses that members of the IX use to interchange traffic. The IP addresses should not be announced as routable across the public internet, and packets destined to these IP addresses should only be permitted from other IX IP addresses.

Open Recursor Problem:

This DDoS attack demonstrates the problem of open DNS recursors, whose misconfiguration threatens the stability of the internet itself. For more information refer to the linked document from the first paragraph ("Deep inside a DNS amplification DDoS attack") and refer to the section "Open DNS resolvers: Bane of the Internet". The Open Resolver Project is attempting to solve this problem by making the full list available to be shut down.