DDOS-Cloudflare

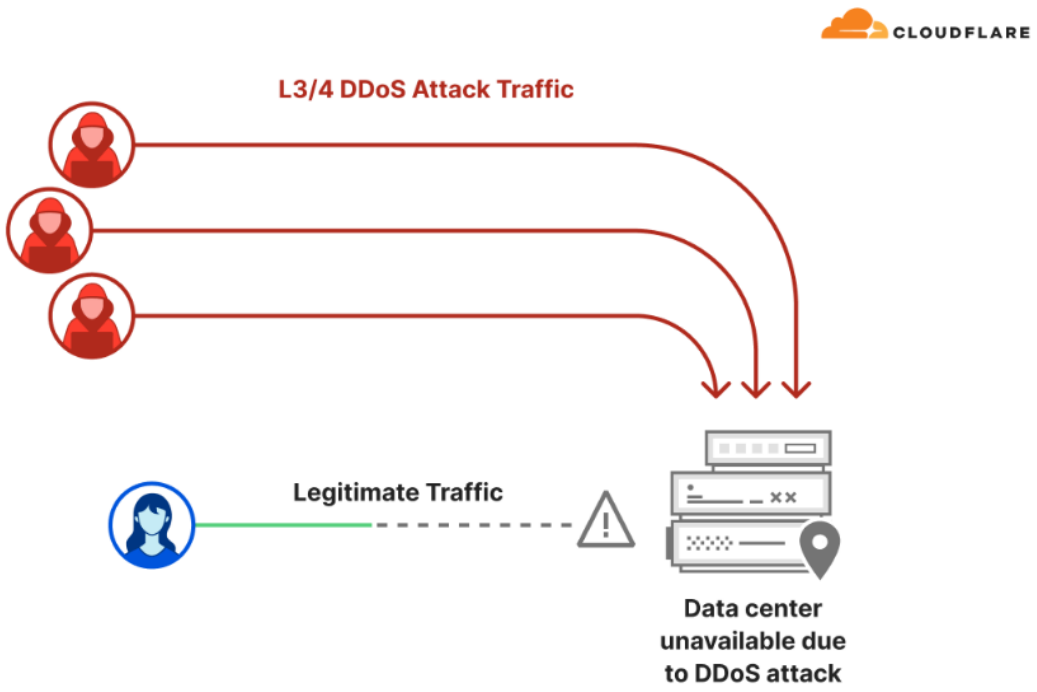
Cloudflare's defenses mitigated over one hundred hyper-volumetric L3/4 DDoS attacks throughout the month, with many exceeding 2 billion packets per second (Bpps) and 3 terabits per second (Tbps). The largest attack peaked 3.8 Tbps — the largest ever disclosed publicly by any organization. Detection and mitigation was fully autonomous.

The scale and frequency of these attacks are unprecedented. Due to their sheer size and bits/packets per second rates, these attacks have the ability to take down unprotected Internet properties, as well as Internet properties that are protected by on-premise equipment or by cloud providers that just don't have sufficient network capacity or global coverage to be able to handle these volumes alongside legitimate traffic without impacting performance.

The attacks predominantly leverage UDP on a fixed port, and originated from across the globe with larger shares coming from Vietnam, Russia, Brazil, Spain, and the US.

The high packet rate attacks appear to originate from multiple types of compromised devices, including MikroTik devices, DVRs, and Web servers, orchestrated to work in tandem and flood the target with exceptionally large volumes of traffic. The high bitrate attacks appear to originate from a large number of compromised ASUS home routers

Before we discuss how Cloudflare automatically detected and mitigated the largest DDoS attacks ever seen, it's important to understand the basics of DDoS attacks.



The goal of a Distributed Denial of Service (DDoS) attack is to deny legitimate users access to a service. This is usually done by exhausting resources needed to provide the service. In the context of these recent Layer 3/4 DDoS attacks, that resource is CPU cycles and network bandwidth.

Layer 3 (Network Layer) DDoS Attacks

Layer 3 attacks target the network protocol, typically IP (Internet Protocol). Common types include:

1. **ICMP Flood**: Sends a massive volume of ICMP echo request (ping) packets to a target, consuming bandwidth and processing resources.
2. **IP Fragmentation (Teardrop)**: Sends malformed or overlapping IP fragments that the target system cannot reassemble, potentially causing crashes.
3. **Smurf Attack**: Sends ICMP packets with a spoofed source address (the victim's) to a network's broadcast address, causing all devices on that network to respond to the victim simultaneously.

Layer 4 (Transport Layer) DDoS Attacks

Layer 4 attacks target transport protocols, primarily TCP and UDP. Major types include:

1. **SYN Flood**: Exploits TCP's three-way handshake by sending many SYN packets without completing the handshake, exhausting the server's connection table.
2. **UDP Flood**: Sends a high volume of UDP packets to random ports, forcing the target to check for applications on those ports and respond with ICMP "Destination Unreachable" messages.
3. **TCP RST/FIN Flood**: Sends TCP packets with RST or FIN flags to disrupt legitimate connections.
4. **TCP Amplification**: Uses techniques like TCP reflection to amplify attack traffic.

Exhausting CPU cycles

Processing a packet consumes CPU cycles. In the case of regular (non-attack) traffic, a *legitimate* packet received by a service will cause that service to perform some action, and different actions require different amounts of CPU processing. However, before a packet is even delivered to a service there is per-packet work that needs to be done. Layer 3 packet headers need to be parsed and processed to deliver the packet to the correct machine and interface. Layer 4 packet headers need to be processed and routed to the correct socket (if any). There may be multiple additional processing steps that inspect every packet. Therefore, if an attacker sends at a high enough packet rate, then they can potentially saturate the available CPU resources, denying service to legitimate users.

To defend against high packet rate attacks, you need to be able to inspect and discard the bad packets using as few CPU cycles as possible, leaving enough CPU to process the good packets. You can additionally acquire more, or faster, CPUs to perform the processing — but that can be a very lengthy process that bears high cost

Exhausting network bandwidth

Network bandwidth is the total amount of data per time that can be delivered to a server. You can think of bandwidth like a pipe to transport water. The amount of water we can deliver through a

drinking straw is less than what we could deliver through a garden hose. If an attacker is able to push more garbage data into the pipe than it can deliver, then both the bad data *and* the good data will be discarded upstream, at the entrance to the pipe, and the DDoS is therefore successful.

Defending against attacks that can saturate network bandwidth can be difficult because there is very little that can be done if you are on the downstream side of the saturated pipe. There are really only a few choices: you can get a bigger pipe, you can potentially find a way to move the good traffic to a new pipe that isn't saturated, or you can hopefully ask the upstream side of the pipe to stop sending some or all of the data into the pipe.

Generating DDoS attacks

Just as it takes CPU cycles to receive a packet, it also takes CPU cycles to create a packet. If, for example, the cost to send and receive a packet were equal, then the attacker would need an equal amount of CPU power to generate the attack as we would need to defend against it. the attacker is able to generate packets using fewer CPU cycles than it takes to receive those packets.

Saturating network bandwidth can be even more difficult for an attacker. Here the attacker needs to be able to output more bandwidth than the target service has allocated. They actually need to be able to exceed the capacity of the receiving service. This is so

difficult that the most common way to achieve a network bandwidth attack is to use a reflection/amplification attack method, for example a DNS Amplification attack. These attacks allow the attacker to send a small packet to an intermediate service, and the intermediate service will send a large packet to the victim.

In both scenarios, the attacker needs to acquire or gain access to many devices to generate the attack.

DNS AMPLIFICATION ATTACK

A DNS amplification can be broken down into four steps:

The attacker uses a compromised endpoint to send UDP packets with spoofed IP addresses to a DNS recursor. The spoofed address on the packets points to the real IP address of the victim.
Each one of the UDP packets makes a request to a DNS resolver, often passing an argument such as "ANY" in order to receive the largest response possible.
After receiving the requests, the DNS resolver, which is trying to be helpful by responding, sends a large response to the spoofed IP address.
The IP address of the target receives the response and the surrounding network infrastructure becomes overwhelmed with the deluge of traffic, resulting in a denial-of-service.

As a result of each bot making requests to open DNS resolvers with a spoofed IP address, which has been changed to the real source IP address of the targeted victim,

the target then receives a response from the DNS resolvers. In order to create a large amount of traffic, the attacker structures the request in a way that generates as large a response from the DNS resolvers as possible. As a result, the target receives an amplification of the attacker's initial traffic, and their network becomes clogged with the spurious traffic, causing a denial-of-service.

How Cloudflare defends against large attacks

Spreading the attack surface using global anycast

In brief, anycast allows a single IP address to be advertised by multiple machines around the world. A packet sent to that IP address will be served by the closest machine. This means when an attacker uses their distributed botnet to launch an attack, the attack will be received in a distributed manner across the Cloudflare network.

Our anycast network additionally allows Cloudflare to allocate compute and bandwidth resources closest to the regions that need them the most. Densely populated regions will generate larger amounts of legitimate traffic, and the data centers placed in those regions will have more bandwidth and CPU resources to meet those needs. Sparsely populated regions will naturally generate less

legitimate traffic, so Cloudflare data centers in those regions can be sized appropriately. Since attack traffic is mainly coming from compromised devices, those devices will tend to be distributed in a manner that matches normal traffic flows sending the attack traffic proportionally to datacenters that can handle it. And similarly, within the datacenter, traffic is distributed across multiple machines.

Additionally, for high bandwidth attacks, Cloudflare's network has another advantage. A large proportion of traffic on the Cloudflare network does not consume bandwidth in a symmetrical manner. For example, an HTTP request to get a webpage from a site behind Cloudflare will be a relatively small incoming packet, but produce a larger amount of outgoing traffic back to the client. This means that the Cloudflare network tends to egress far more legitimate traffic than we receive. However, the network links and bandwidth allocated are symmetrical, meaning there is an abundance of ingress bandwidth available to receive volumetric attack traffic.

Generating real-time signatures

By the time you've reached an individual server inside a datacenter, the bandwidth of the attack has been distributed enough that none of the upstream links are saturated.

That doesn't mean the attack has been fully stopped yet, since we haven't dropped the bad packets.

We need to execute custom code in kernel space and process (drop, forward, or modify) each packet directly at the network interface card (NIC) level. This component helps the system drop packets efficiently without consuming excessive CPU resources on the machine.

We use XDP to sample packets to look for suspicious attributes that indicate an attack. The samples include fields such as the source IP, source port, destination IP, destination port, protocol, TCP flags, sequence number, options, packet rate and more. This analysis is conducted by the *denial of service daemon (dosd)*. *Dosd* holds our secret sauce. It has many *filters* which instruct it, based on our curated heuristics, when to initiate mitigation. To our customers, these filters are logically grouped by attack vectors and exposed as the DDoS Managed Rules. Our customers can customize their behavior to some extent, as needed

The detection and mitigation of attacks by dosd is done at the server level, at the data center level and at the global level — and it's all software defined.

HTTP DDOS Attacks

An HTTP DDoS attack usually involves a flood of HTTP requests towards the target website. The attacker's objective is to bombard the website with more requests than it can handle. Given a sufficiently high amount of requests, the website's server will not be able to process all of the attack requests along with the *legitimate* user requests. Users will experience this as website-load delays, timeouts,

and eventually not being able to connect to their desired websites at all.

To make attacks larger and more complicated, attackers usually leverage a network of bots — a *botnet*. The attacker will orchestrate the botnet to bombard the victim's websites with HTTP requests. A sufficiently large and powerful botnet can generate very large attacks as we've seen in this case.

HTTP attributes commonly used in DDoS attacks

HTTP methods (also called HTTP verbs) define the action to be performed on a resource on a server. They are part of the HTTP protocol and allow communication between clients (such as browsers) and servers.

The GET method is most commonly used. Almost 70% of legitimate HTTP requests made use of the GET method. In second place is the POST method with a share of 27%.

With DDoS attacks, we see a different picture. Almost 14% of HTTP requests using the HEAD method were part of a DDoS attack, despite it hardly being present in legitimate HTTP requests (0.75% of all requests). The DELETE method came in second place, with around 7% of its usage being for DDoS purposes.

HTTP paths

An HTTP path describes a specific server resource. Along with the HTTP method, the server will perform the action on the resource.

How do application layer attacks work?

Even in the absence of a login, many times a server receiving a request from a client must make database queries or other API calls in order to produce a webpage. When this disparity is magnified as a result of many devices targeting a single web property like during a botnet attack, the effect can overwhelm the targeted server, resulting in denial-of-service to legitimate traffic. In many cases simply targeting an API with a L7 attack is enough to take the service offline.

Why is it difficult to stop application layer DDoS attacks?

Distinguishing between attack traffic and normal traffic is difficult, especially in the case of an application layer attack such as a botnet performing an HTTP Flood attack against a victim's server. Because each bot in a botnet makes seemingly legitimate network requests the traffic is not spoofed and may appear "normal" in origin.

With other attacks such as SYN floods or reflection attacks such as NTP amplification, strategies can be used to drop the traffic fairly efficiently provided the network itself has the bandwidth to receive them.

What tactics help mitigate application

layer attacks?

One method is to implement a challenge to the device making the network request in order to test whether or not it is a bot. This is done through a test much like the CAPTCHA test commonly found when creating an account online. By giving a requirement such as a JavaScript computational challenge, many attacks can be mitigated.

Other avenues for stopping HTTP floods include the use of a web application firewall, managing and filtering traffic through an IP reputation database, and through on-the-fly network analysis by engineers.

Volumetric Attacks

These attacks are based on brute force techniques where the target server is flooded with data packets to consume bandwidth and server resources.
Volumetric attacks will frequently rely on *amplification* and *reflection*.
Amplification is where a request in a certain protocol will result in a much larger response (in terms of the number of bytes); the ratio between the request size and response size is called the Amplification Factor.
Reflection is where the attacker will spoof the source of request packets to be the target victim's IP address. Servers will be unable to distinguish legitimate requests from

spoofed ones so they'll send the (much larger) response payload to the targeted victim's servers and unintentionally flood them.

Application Layer Attacks

These DDoS attacks target the "top" layer in the OSI model - the application layer. Attackers might flood the backend with HTTP requests, exploit expensive API endpoints, create many SSL/TLS handshakes, etc.
Database DDoS attacks are quite common, where a hacker will look for requests that are particularly database-intensive and then spam those in an attempt to exhaust the database resources. Scaling your database through read replicas takes time, so this attack can be pretty successful.
HTTP Floods are some of the most widely seen layer 7 DDoS attacks, where hackers will spam a web server with HTTP GET/POST requests. Sophisticated hackers will specifically design these to request resources with low usage in order to maximize the number of cache misses the web server has.

Protocol Layer Attacks

Protocol attacks will rely on weaknesses in how particular protocols are designed. Examples of these kinds of exploits are SYN floods, BGP hijacking, Smurf attacks and more.
A SYN flood attack exploits how TCP is designed, specifically the handshake process. The three-way handshake consists of SYN -> SYN-ACK -> ACK, where the client sends a synchronize (SYN) message to initiate, the server responds with a synchronize-acknowledge (SYN-ACK) message and the client then responds back with an acknowledgement (ACK) message.
In a SYN flood attack, a malicious client will send large volumes of SYN messages to the server, who will then respond back with SYN-ACK. The client will ignore these and never respond back with an ACK message. The server will waste resources (open ports) waiting for the ACK responses from the malicious client. If repeated on a large scale, this can bring the web server down since the server won't know which requests are legitimate.