What Is Rate Limiting?

Rate limiting is a technique to limit network traffic to prevent users from exhausting system resources. Rate limiting makes it harder for malicious actors to overburden the system and cause attacks like Denial of Service (DoS). This involves attackers flooding a target system with requests and consuming too much network capacity, storage, and memory.

APIs that use rate limiting can throttle or temporarily block any client that tries to make too many API calls. It might slow down a throttled user’s requests for a specified time or deny them altogether. Rate limiting ensures that legitimate requests can reach the system and access information without impacting the overall application’s performance.

This is part of a series of articles about website security.

Why Is Rate Limiting Important?

Rate limiting is a crucial part of a modern cybersecurity strategy. It addresses several attack techniques that affect the incoming request rate.

Distributed Denial of Service (DDoS)

A DDoS attack attempts to overwhelm the target system with traffic, making it unavailable to legitimate users. Rate limiting mitigates DDoS threats by preventing any given traffic source from sending too many requests.

However, DDoS attacks have unique challenges because they distribute requests among many different sources, sometimes millions of IP addresses. Distributing the attack allows each source to avoid exceeding the rate limit. The security solution should identify the requests from different locations as part of a single attack, treating them as a single source.

Credential Stuffing

When attackers compromise databases containing user credentials, they can use these credentials to carry out further attacks. Usually, a bot stuffs stolen user credentials into a login form until a credential set works, allowing the bot to access the account.

Bots are often highly successful because they can submit hundreds or thousands of credentials into a login form. Rate limiting helps identify credit stuffing and block the bots before they take over the account.

Brute Force

A brute force attack is similar to a credential stuffing attack but without a list of real user credentials. In this case, the bot systematically submits randomly generated credentials until a credential set works.

A strongly secured web application sets password requirements that help mitigate brute force attacks, but large attacks can still consume many network resources. Rate limiting blocks these attacks to save system resources.

Data Scraping and Theft

Malicious actors often scrape target websites for information they can sell or use to undermine competitors. For example, an attacker might steal pricing information from an eCommerce company. A scraper bot can copy large amounts of data from target applications. Rate limiting detects and blocks data scraping.

Learn more in our detailed guide to web scraping

Inventory Denial

An inventory denial or inventory hoarding attack involves sending bots to a target web application, where they start transactions without finishing them. It hoards the inventory, making it unavailable to legitimate users.

Learn more in our detailed guides to:

•***Bots***
•***Bot management***

How Does Rate Limiting Work?

Rate limiting works within applications, not in the web server. Rate limiting typically involves tracking the IP addresses where requests originate and identifying the time lapsed between requests. IP addresses are the application’s main way to identify who has made each request.

Rate-limiting solutions work by measuring the elapsed time between every request from a given IP address and tracking the number of requests made in a set timeframe. If one IP address makes too many requests within the specified timeframe, the rate-limiting solution throttles the IP address and doesn’t fulfill its requests for the next timeframe.

Rate-limited applications can tell individual users to slow down if they make requests too frequently. It’s akin to police officers pulling over drivers that exceed the speed limit or parents telling their children not to eat too much sugar in a short period.

Types of Rate Limits

Administrators can define different parameters and methods and parameters when setting a rate limit. An organization’s chosen rate-limiting technique depends on the objective and the required level of restriction. Here are three main approaches to rate limiting that an organization might implement:

•**User rate limits**—this is the most popular rate-limiting method. It identifies the number of requests a given user makes, usually by tracking the user’s IP address or API key. Users that exceed the specified rate limit will trigger the application to deny any further requests until the rate-limited timeframe resets. Alternatively, users can contact the developers to increase the rate limit.
•**Geographic rate limits**—developers can further secure applications in a given geographic region by setting a rate limit for each specific region for a specified timeframe. For example, developers might predict that users in a given region will be less active between midnight and 9:00 am and define a lower rate limit for this timeframe. This approach helps prevent suspicious traffic and further reduces the risk of an attack.
•**Server rate limits**—developers can set rate limits at the server level if they define a specific server to handle parts of an application. This approach provides more flexibility, allowing the developers to increase the rate limit on commonly used servers while decreasing the traffic limit on less active servers.

What Are the Algorithms Used for Rate Limiting?

There are several types of rate-limiting algorithms.

Fixed-window Rate Limiting

Fixed-window rate limiting algorithms restrict the number of requests allowed during a given timeframe (window). For instance, a server’s rate-limiting component might implement an algorithm that accepts up to 200 API requests per minute. There is a fixed timeframe starting from a specified time—the server will not serve more than 200 requests between 9:00 and 9:01, but the window will reset at 9:01, allowing another 200 requests until 9:02.

Developers can implement a fixed-window algorithm at the server or user level. Implementing the algorithm at the user level will restrict each user to 200 requests per minute. In contrast, a server-level algorithm will restrict the server, meaning that all users combined can make up to 200 requests per minute.

Leaky Bucket Rate Limiting

Leaky bucket rate limiting algorithms differ from fixed-window algorithms because they don’t rely on specified timeframes. They focus on the fixed length of request queues without considering the time. The server will service requests on a first-come, first-serve basis. New requests join the back of the queue. The server will drop a new request if it arrives when the queue is full.

Sliding-window Rate Limiting

Sliding-window rate limiting algorithms are similar to fixed-window algorithms except for the starting point of each time window. With sliding-window rate limiting, the timeframe only starts when a user makes a new request, not a predetermined time. For instance, if the first request arrives at 9:00:24 am (and the rate limit is 200 per minute), the server will allow up to 200 requests until 9:01:24.

Sliding-window algorithms help solve the issues affecting requests in fixed-window rate limiting. They also mitigate the starvation issue facing leaky bucket rate limiting by providing more flexibility.

Rate Limiting with Imperva

Imperva Advanced Bot Protection enforces rate limiting for websites, mobile apps and APIs, and can prevent business logic attacks from all access points. Gain seamless visibility and control over bot traffic to stop online fraud through account takeover or competitive price scraping.

Beyond bot protection, Imperva provides comprehensive protection for applications, APIs, and microservices:

Web Application Firewall – Prevent attacks with world-class analysis of web traffic to your applications.

Runtime Application Self-Protection (RASP) – Real-time attack detection and prevention from your application runtime environment goes wherever your applications go. Stop external attacks and injections and reduce your vulnerability backlog.

API Security – Automated API protection ensures your API endpoints are protected as they are published, shielding your applications from exploitation.

DDoS Protection – Block attack traffic at the edge to ensure business continuity with guaranteed uptime and no performance impact. Secure your on premises or cloud-based assets – whether you're hosted in AWS, Microsoft Azure, or Google Public Cloud.

Attack Analytics – Ensures complete visibility with machine learning and domain expertise across the application security stack to reveal patterns in the noise and detect application attacks, enabling you to isolate and prevent attack campaigns.

Client-Side Protection – Gain visibility and control over third-party JavaScript code to reduce the risk of supply chain fraud, prevent data breaches, and client-side attacks.