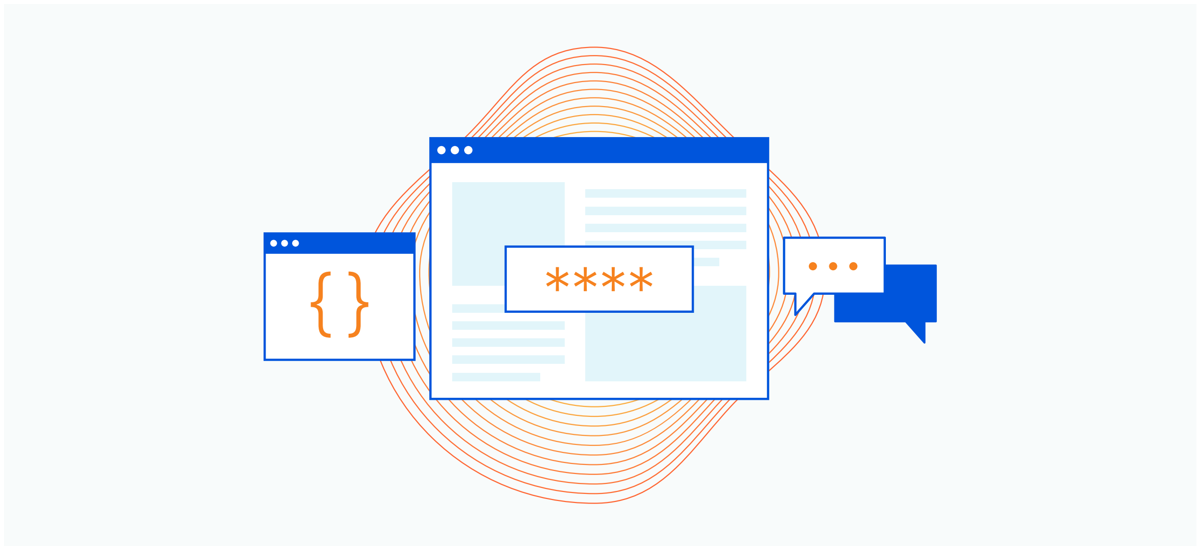
WHITE PAPER

CLOUDFLARE®

A Guide to API Security

APIs make the (app) world go round



We all know by now that APIs, application programming interfaces, make the world go around. More precisely, they let distinct modern applications communicate with each other. Mobile or web applications can access a backend where data is stored and processed. APIs can be public allowing applications from different companies to communicate or private, which is common, where internal applications integrate to meet business goals.

The result? More robust, full-fledged applications, websites and mobile apps with broader functionality and more diverse data.

For instance, rather than creating their own payment services from scratch, ride-sharing companies can add it via the APIs of payment companies. Another example are flight aggregator sites. In order to show us flight times, prices, destinations and everything else we need to know about a plane ticket, they connect via API calls to airline databases to pull the right data and display it to us in our aggregator search results page.

The importance of APIs are growing, with more and more companies describing themselves as "API-first." In some cases, a company's actual product is an API with a business model centered on consuming it. For instance, if a company that provides weather data has made their API their product, other companies who want weather info will pay them a monthly fee for API access. In many other instances, given the expectation an application must interact with other applications, APIs are designed alongside or even before the code that delivers actual product functionality–not bolted on at the end of development.

Companies devote time and effort to deliberately craft their API approach to consider how it can expose the right data, becoming foundational to revenue and business models.

However, building perfect APIs is tough, because just like any piece of software, vulnerabilities will happen, leading to the security challenges we'll discuss in this document.

Suffice it to say, APIs are everywhere and will only gain momentum in the coming years–and they must be protected. For this reason, we'll examine API attacks and aspects of defense-in-depth when thinking through organizational API security.

API momentum in numbers

50%

of the traffic flowing through Cloudflare is API traffic

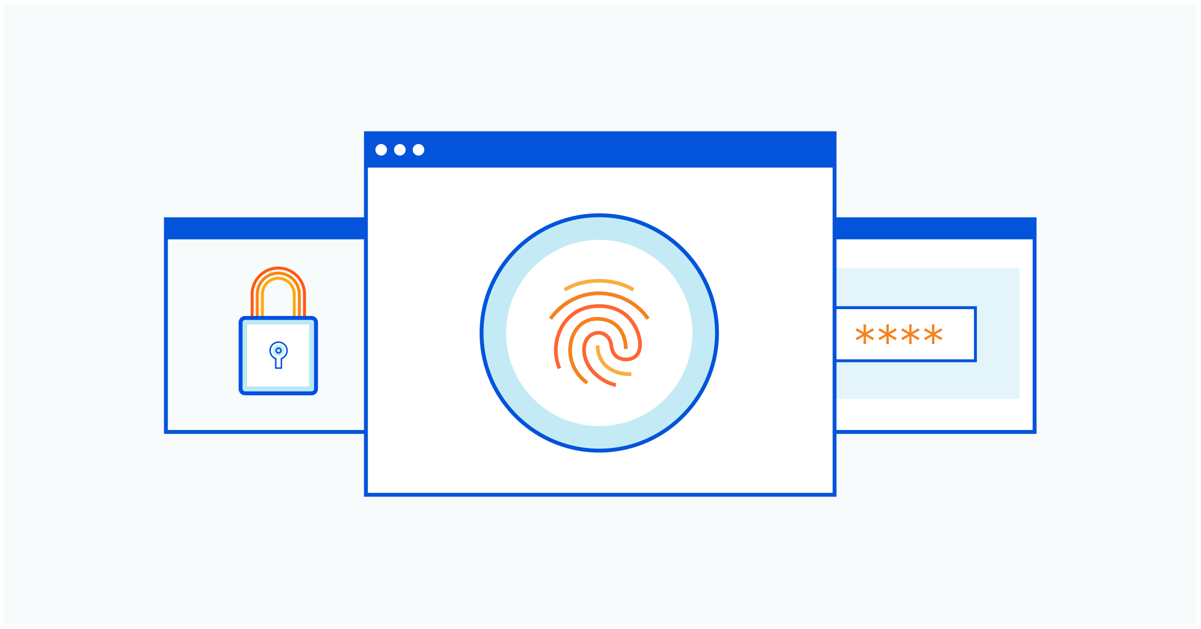
61%

API traffic increased year over year

Programmable Web[1] notes there are more than 24,000 published, well-known APIs. It turns out however most APIs are private, linking internal applications together. Estimates of the number of private APIs are in the millions.

And when we say APIs are gaining momentum, Cloudflare is a first-hand witness to their growth. According to data from Cloudflare Radar from the first half of 2021, roughly half the traffic on the Cloudflare network was API-related. What's more, it increased 61 percent from 2020 to 2021.

Given that they expose important data, we can start to see how they represent an enormous new attack surface we must protect. How do we know this? There have been many prominent attacks in recent years targeting APIs.



[1]https://www.programmableweb.com/apis/directory

An Expanding Attack Surface

We now know that APIs are everywhere and fundamental to the success of modern business. They expose application logic and can share sensitive data with other applications.

It turns out however, in a surprise to no one, that attackers know this and have every intention of exploiting this expanding attack surface in the enterprise.

Maybe Gartner was right when they asserted[2] that by 2022, API abuses will "move from an infrequent to the most-frequent attack vector, resulting in data breaches for enterprise web applications."

Whether APIs become the most frequently targeted attack vector remains to be seen, but it is clear APIs will continue to be in attacker crosshairs.

We say "continue" because the world has already witnessed some high-profile breaches at the hands of weak API security or API development that did not account for security.

T-Mobile fell victim to an API attack in 2017 when 15 million customers who bought new devices or applied for T-Mobile accounts had their information exposed. Data included names, addresses, birth dates, Social Security numbers, drivers license numbers and passport numbers. Vice reported the attack was carried out by adjusting the phone number parameter in the API call. Any user's phone number could be queried and the T-Mobile API would send responses including the sensitive account data of the person whose phone number was queried.

Another API-related breach hit the USPS when an API that supported real-time package tracking was found to lack basic authorization. When a user was logged in, they could query for account information of any other user, via use of wildcard search parameters that would cough up all the records of the data set. This put the data of 60 million USPS account holders at risk.

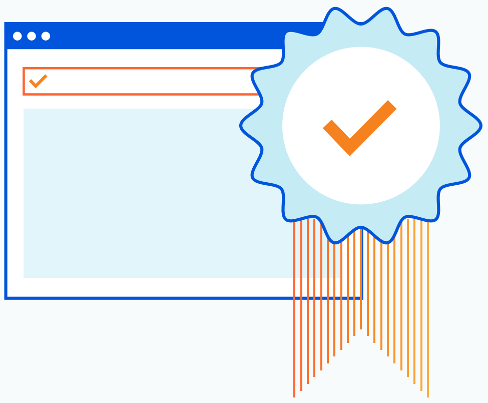
In 2019, JustDial, a large local search engine in India, effectively leaked all customer data when they left older versions of APIs, that had since been replaced by newer versions, exposed. It gets worse–there was effectively no authentication in place, meaning anyone could call the APIs and scrape data from the production server. Put another way, no advanced techniques were required to access user data.

Facebook, despite their leadership with GraphQL, has also suffered multiple breaches at the hands of their APIs. For instance in late 2019, Facebook had a database scraped, putting the names, phone number and user IDs of more than 260 million users at risk.

There is a long list of organizations who have had trouble securing APIs. There are a number of reasons why. First, underlying API vulnerabilities due to insecure design open the door to attacks. What is more, to date, organizations have not had API-first security tools. Perhaps they would use web security tools like WAFs or rate limiting but those were built to protect applications–not APIs. This can lead to challenges like false positives and makes clear the need for API security designed for traffic that is largely automated.

[2]Source: Gartner: "API Security: What You Need to Do to Protect Your APIs", March 2021

Remix! A New OWASP API Top 10



A silver lining in all of this is that the OWASP Foundation, an organization long-focused on improving application security, has gotten involved. OWASP is known for its Top 10 Web Application Security Risks and now they have published the API Security Top 10, a list of top API security risks and vulnerabilities.

The truth is, what has long worried us about application security, also holds true for building and securing APIs.

To begin with, any organization that is API-first, must consider security upfront, as they design their APIs. Let's examine some of the attacks we just mentioned, and the OWASP security risk they exploited.

OWASP API Top 10

1. Broken Object Level Authorization
2. Broken User Authentication
3. Excessive Data Exposure
4. Lack of Resources & Rate Limiting
5. Broken Function Level Authorization
6. Mass Assignment
7. Security Misconfiguration
8. Injection
9. Improper Assets Management
10. Insufficient Logging & Monitoring

Key API security challenges

1. Broken authentication and authorization

Let's take a closer look at a few key OWASP API risks the attacks above exploited, starting with authentication and authorization.

JustDial succumbed to broken authentication on their endpoints, that allowed anyone to make calls to them. When authentication is implemented, only API calls with the right TLS certificate, API keys, web tokens, etc. are permitted to make requests, drastically reducing API security risk.

Moving along to number one on the OWASP list, many API attacks exploit weak, broken or non-existent authorization, like we saw with USPS and T-Mobile. Broken object level authorization is common - when API endpoints are exploited by substituting the ID number of an object they are authorized to access for the ID of something they are not authorized to access. Often by just changing the object ID in a request, they gain unauthorized access to data.

API path and query parameters include the resource ID being accessed:

Authorized call:

`GET api.greatsampleapis.com/v1/users/235` where 235 is the user ID.

Manipulated API calls can gain unauthorized access by changing 235 to 236, that is, adjusting the object identifier, the userID in this case, to access data for user 236.

```
GET api.greatsampleapis.com/v1/users/236
```

If authorization controls are not in place, this can be successful. Developers should threat-model their endpoint to make sure attacks cannot adjust or modify an object's ID value to gain access to other data. Use of unpredictable object ID values can also help so they are not sequential and easy to guess.

2. Mass assignment, data exposure and injection attacks

Another class of attacks expose too much data in responses or permit internal objects to be modified with inputs.

Excessive data exposure happens when an API looks to broadly expose object properties and returns too much data in a response, relying on clients making the request to filter data.

Attackers can use additional details from the response to create an even more powerful attack or phishing email. For instance if a response returns all of the below data, it can then be used for very convincing phishing emails:

```
{
“Id”: 213,
“FirstName”:”Sanjay”,
“LastName”:”Smythe”,
“EmailAddress”:”ssmythe@hacketyhack.com”,
“Occupation”: “Assistant to the Deputy Associate Vice Sub-undersecretary”,
“DOB”:”1986-05-21”,
“Bank”:”Easygo Financial”,
“AccountNumber”: 1362886306,
“PetName”: “Aloysius”,
 }
```

Mass Assignment attacks allow for API calls to alter or exploit internal values when APIs expose internal objects and variables.

OWASP puts it this way:

> “Software frameworks sometime allow developers to automatically bind HTTP request parameters into program code variables or objects to make using that framework easier on developers... Attackers can sometimes use this methodology to create new parameters that the developer never intended which in turn creates or overwrites new variable or objects in program code that was not intended.”

What should developers do? They should understand the potential risks when calling on mass assignment in development and steer clear of exposing any internal objects or variables that can be exploited. Allowlisting properties that can be updated by clients should also be considered.

Web applications have long been susceptible to injection attacks and APIs are no different. Given how well known injection attacks are, we will not dwell on them, but suffice it to say inputs should be validated and sanitized before being passed on. Efforts should be made to use data leak prevention on API responses and limit the number of records that can be returned to prevent a mass disclosure incident.

3. Abuse of resources and shadow/rogue APIs

Other attacks can abuse APIs, so they consume inordinate amounts of compute resources–becoming overwhelmed, succumbing to a DoS-like attack. The door is left open to these attacks If limits are not placed on aspects like number of requests per client/resource, records returned in a single response or request payload size.

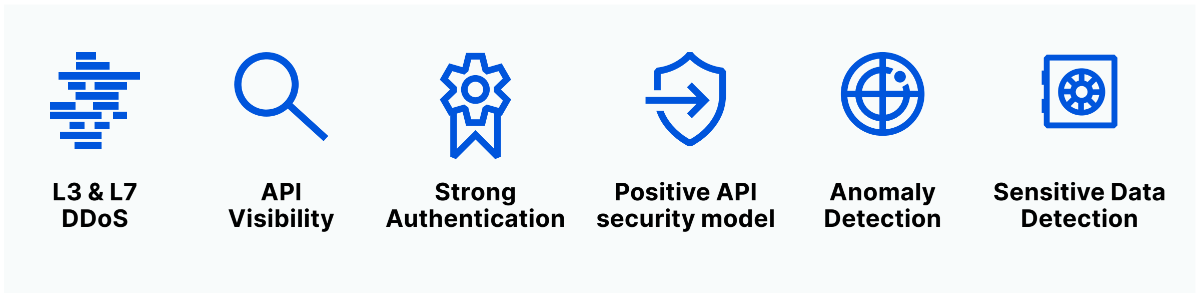
As we saw in the JustDial attacks, production APIs can be forgotten, thus becoming shadow or rogue, since they are likely unprotected and can be exploited. Like in the rest of security, we must have visibility into our IT estate or attack surface to then apply appropriate security controls. Visibility into our entire API endpoint estate is no different.

When developing APIs, teams should have a refined process for tracking API versions to understand what API are in production and what is deprecated.

Considerations for Securing APIs

We have covered what APIs are and why they matter and the general attacks that target APIs. Now let's examine how Cloudflare has built API security to secure APIs from the most common attacks. Effective API security must account for everything from visibility, to positive security models to stopping abuse to data protection.

Cloudflare API Shield



Foundation of visibility

Visibility

Like with other aspects of security, we have to see something in order to protect it. APIs are no different, particularly when companies have hundreds or even thousands of API endpoints.

API discovery and visibility is a key foundational aspect to governing APIs so organizations always have a clear snapshot of their API endpoint estate to prevent shadow or rogue APIs from becoming an issue.

As we saw with JustDial, if organizations lose track of APIs, this can lead to data breaches.

API Defense-in-depth

API L7 protections

We have long deployed web application firewalls to protect applications against layer 7 DDoS attacks. API protection should begin with many of these dependable controls like rate limiting and DDoS protection to ward off denial-of-service attacks and brute-force login attempts and general abuse carried out by specific IP addresses. This will enforce API usage limits and ensure availability combatting OWASP API 4—Lack of Resources & Rate Limiting.

WAF rules should also be used to identify and block common attacks targeting APIs.

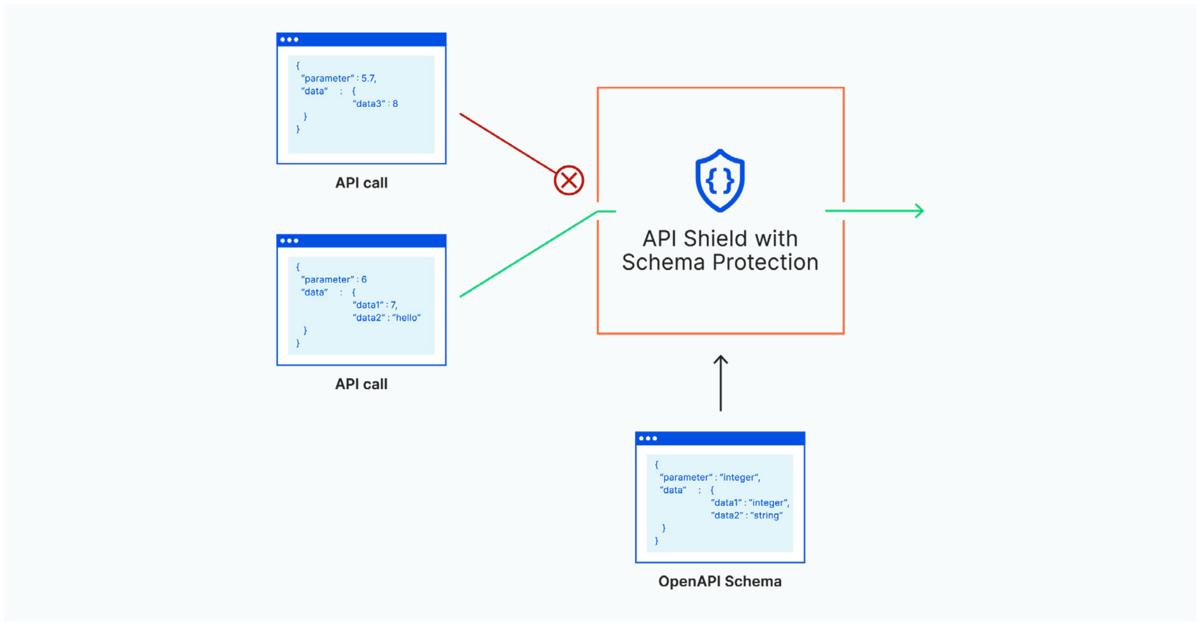
Authentication and Authorization

mTLS authentication

We saw in the API attacks we outlined that lack of authentication can be devastating. Authentication must be built-in from the outset and it should be bolstered with mutual TLS to enforce on certificate-based identity for use cases like mobile or IoT. This approach is a more positive, allowlist model that only allows requests from legitimate clients with valid certificates to connect.

Exposed credential checks

APIs are not immune from credential stuffing attacks, cycling through attempted logins using stolen credentials. These account credentials could get compromised by third-party breaches outside of an organization's control. As part of authentication checks, API security should be able to scan authentication credentials at log-in, against a database of leaked credentials. If credentials do appear to be compromised, API security should trigger additional security measures like password reset or multi-factor authentication and of course, block the attempt.


```
{
  "parameter" : 5.7,
  "data"    :   {
                  "data3" : 8
   }
}
```

```
{
  "parameter" : 6
  "data"    :   {
                  "data1" : 7,
                  "data2" : "hello"
   }
}
```

```
{
  "parameter" : "integer",
  "data"    :   {
                  "data1" : "integer",
                  "data2" : "string"
   }
}
```

Schema Validation evaluates each request against an API Schema logging or blocking requests that do not comply with it.

Positive API protection

API Schema validation

Developers go to great lengths to create an API schema, which is the documentation, or ground rules, for how they expect others to interact with the API. This can establish things like request methods and operations on each endpoint (GET /users, POST /users) or input and output parameters for each operation. OpenAPI v3, also commonly known as the Swagger standard, is the most well-known schema for defining APIs.

Reliable API security should use a positive, zero trust model, that enforces on the schema.

With a schema in place, requests should be automatically validated against it. All API operations are blocked, except those that have been validated as conforming to the schema.

API anomaly detection

A next sophisticated layer of API defense is anomaly detection, to detect and block volumetric anomalies—abuse—that hit individual API endpoints.

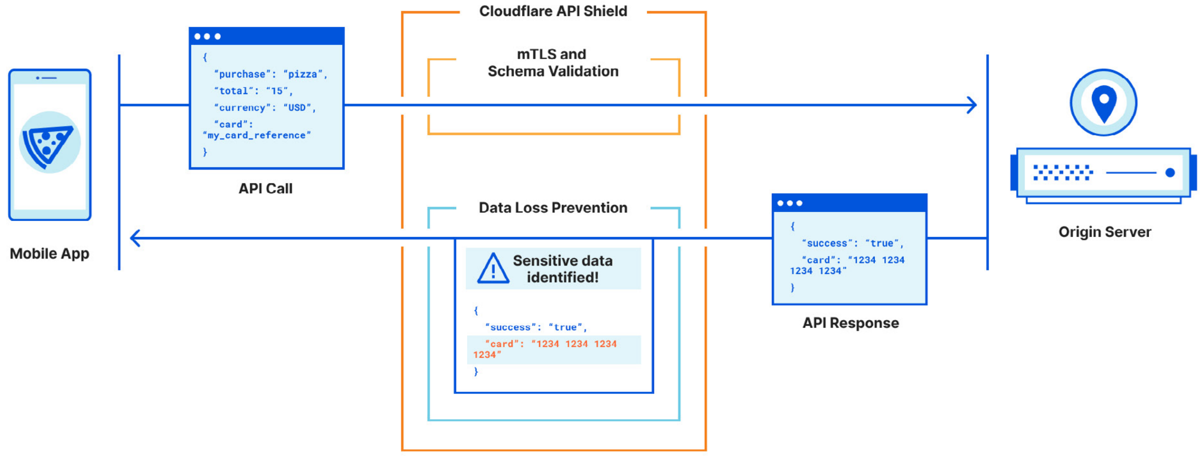
The challenge to getting this right is understanding how often a path is reached and enforcing a threshold particular for that API endpoint. For instance, the API path `/update-score` for a sports website may need to support hundreds of requests per minute.

The same sports website may have a password reset endpoint, `/reset-password` but no sports fan resets their password as much as they check scores, so this path requires a much lower threshold.

Advanced API security, using unsupervised machine learning, is capable of developing separate baselines for each API, and predict the intent of requests as they are made. If 100 sudden attempts are made to reset a password, a session should be blocked as anomalous, as this is a likely account takeover attempt.

In this instance API security should apply request or user-based rate limiting to throttle requests during a session on attributes like user ID, API keys or tokens to prevent abuse.

How Sensitive Data Detection works



```
{
  "purchase": "pizza",
  "total": "15",
  "currency": "USD",
  "card":
"my_card_reference"
}
```

```
{
  "success": "true",
  "card": "1234 1234
1234 1234"
}
```

```
{
  "success": "true",
  "card": "1234 1234 1234
1234"
}
```

Sensitive Data Detection

Not only must security keep the bad out, but also keep the good in. A last, vital layer of API security should be considered: sensitive data detection. To backstop the defense-in-depth already in place, a security layer should detect and block exfiltration of sensitive information in API responses. This includes API keys, social security numbers and PII, credit card numbers or bank information. Many will know that DLP has gained a reputation, fairly or unfairly, as difficult to deploy and tune, so ease-of-use should be a factor in selection.