ExpeditedWAF

API SECURITY

BEST PRACTICES

MegaGuide

What is API Security, and how can this guide help?

Application Programming Interface (API) Security is the design, processes, and systems that keep a web-based API responding to requests, securely processing data and functioning as intended.

API Security is an incredibly broad topic as the many different API frameworks, and systems all operate differently. This guide will help you build better, more secure APIs and help you assess the security footing of your current API implementations.

Why is API Security Important?

Modern web programming has split the front end (HTML/CSS/JS) of applications from the backend (Server-side processing and storage). Similarly, mobile applications often handle user-facing functions within the app but pass data back to a centralized server. These trends have sparked a massive increase in the number of web APIs that are now in production.

What differentiates API security from general Web security?

Web/HTTP Application Programming Interfaces (API) have unique threat models, security concerns, and authentication modes that are distinct from standard web applications. Much of this difference is because standalone APIs cannot rely on basic browser security features to help limit the scope of actions an attacker could do.

How to Secure an API

Our guide below is a mix of techniques and recommendations that:

- You can directly implement in your application
- Design choices you can make to strengthen your security model
- Features of Expedited WAF that can make the whole process easier

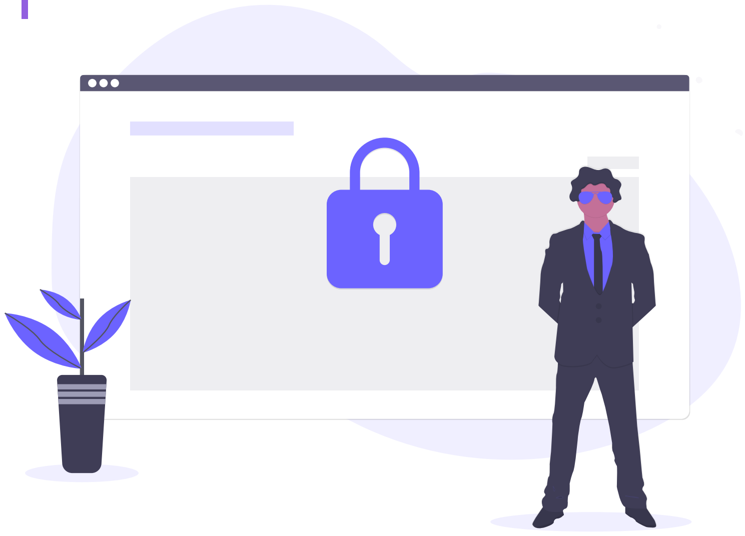
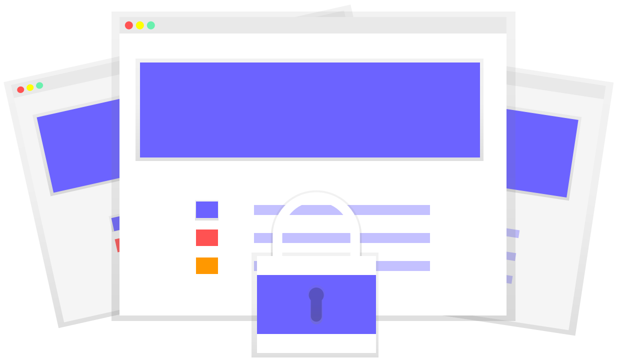


Table of Contents

1 Distributed Denial of Service (DDoS) Attacks on APIs

Any action taken against your API that results in disruption of service to legitimate requests is a denial of service attack.



Defending API's from DDOS attacks is a challenge for two main reasons:

API clients tend to consume both more and more expensive requests than web users.

1. Some of the most common DDoS prevention mechanisms (like CAPTCHAs) don't make sense in an API context.
2. Even worse, external (non-accidental) attacks are typically more serious as they're typically initiated by:

Competitors who are trying to take you offline.

- Disgurntled former employees (with a deep knowledge of your systems).
- Cyberr-criminals that are seeking to extort you to call off the attack.
- No magic bullet will stop all attacks against your API, but the following approaches form a "recipe" that will blunt most attacks.

Which one to apply depends upon how clients authenticate to your API, whether clients operate from within a browser a dedicated client app or a hybrid system and the overall number of clients that you need to support.

Classic Distributed Denial of Service (DDoS) Attacks

Classic Distributed Denial of Service (DDoS) Attacks rely upon low-level network tricks like TCP Amplification Attacks that do things like send one packet but that require the server to send ten packets in response.

Network and cloud operations have become adept at identifying and stopping these types of attacks.

Modern Denial of Service Attacks (HTTP Flood)

Modern denial of service attacks employ full GET/POST HTTP Requests that mirror legitimate traffic (HTTP Floods).

Web API's that work at the HTTP level are vulnerable to these attacks as each API request: Takes a larger slice of server resources to parse and respond. Holds a connection open for longer. It is more difficult to determine if it is legitimate or something deliberately meant to be disruptive.

What's worse is that the complex interaction of API providers and API clients can easily result in an unintentional Denial of Service attack on the API.

CASE STUDY: The Out of Control API Client

An Expedited WAF customer had developed an API used by hundreds of high volume e-commerce sellers, each interacting with the API via a customized client.

A bug in a single seller's client code caused a massive spike in API requests. The requests were legitimate calls, and the client was fully authenticated and authorized to make the requests.

The wave of calls was massive enough to start knocking off other clients and slowing their entire system, despite their efforts to add more resources to deal with the issue.

“We couldn't identify and kill connections fast enough within our app to keep up.”

Blocking the client's API key was ineffective as:

- Blocked HTTP requests still ate up app resources that they were knocking legitimate requests offline.
- The API client retried failed requests and kept hammering the API.

Unable to contact the client and with support requests pouring in, they blocked the client's IP address in Expedited WAF. This action immediately stopped the requests at the network level, and the properly configured clients were able to use the API again successfully.

2 Data Breach Attacks on APIs

Data breach attacks seek to extract information from your API beyond what the user is authorized to access. These attacks range from subtle (manipulating search filters to return normally out of bounds records) to bots brute force trying "unguessable" URLs as they trawl for data.

APIs are particularly vulnerable to data breach attacks as they are designed to be consumed programmatically. An attacker doesn't have to reverse engineer endpoints from your web app or try and work around authenticity tokens in web forms.

Note: we're classifying these attacks as "Data Breach" attacks as from our perspective that's the worst-case scenario (that all the)

SQL Injection (SQLi) in APIs

SQL Injection is the most well-known data exploit. The classic version of the attack features an attacker changing an API URL from requesting a single ID value like:

Standard Request

Request: https://api.example.com/?user_id=1234

Generated SQL: SELECT * FROM users WHERE id = 1234

Injected Request

Request: https://api.example.com/?user_id=1234%20OR%20*

Generated SQL: SELECT * FROM USERS WHERE id = 1234 OR *

This enables them to pull the entire Users table from the application.

These types of attacks are now generally blocked as part of the security mechanisms of web frameworks.

However:

1. Perfect implementations don't exist, web frameworks still rely on developers properly implementing protection mechanisms perfectly every time, even in situations where time and resources are tight.

2. The one area developers are most likely to stray from standard out of the box framework querying features is in reporting. Many applications try to expose a "Report Builder" interface that lets users specify columns, filters, and joins to pull their data.

These queries back to your API deserve special attention for proper escaping as they're more complex and more vulnerable to manipulation.

CASE STUDY: JSON SQL Injection

In the course of an investigation on behalf of a client, we found that a developer had built a custom "REST Query Interface" that they were incredibly proud of where the front end of the application would create a SQL Query from a visual drag and drop query builder.

> **"{query: 'SELECT * FROM any_table WITH no_restrictions'}"**

The resulting SQL statement was then bundled up into a JSON Object and POST'd back to the API for raw execution, completely bypassing their web framework's built-in SQL Injection Protection.

Non-SQL Query Injection (NSQLi)

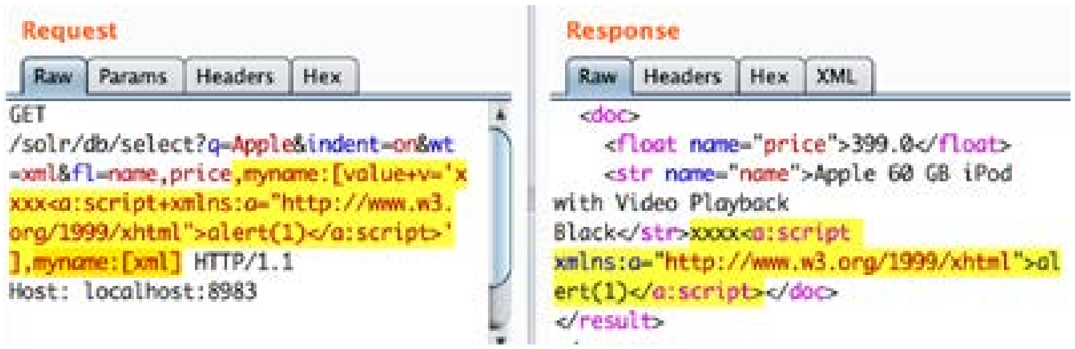
Datastores have gotten more diverse as application needs have changed, and modern web applications often use a mix of full-text search applications, No-SQL data stores, and caching servers to deliver data back to clients.

These datastores, each with their own custom query language, are just as vulnerable to injection attacks as traditional SQL servers (Like Postgres and MySQL).

But the web frameworks that most developers rely on to help protect them don't have equally mature built-in sanitization mechanisms.

This is not theoretical, practical examples of these attacks are currently in the wild. One solid example is of the SOLR search engine and the corresponding Apache Solr Injection project which demonstrates how trivial it is for improperly escaped SOLR query to return out of bounds data or to enable remote code execution (as seen in the screenshot below).

SOLR Query with Injected Script from the SOLR Injection Project



```
GET
/solr/db/select?q=Apple&indent=on&wt
=xml&fl=name,price,myname:[value+v='x
xxx<a:script+xmlns:a="http://www.w3.
org/1999/xhtml">alert(1)</a:script>'
],myname:[xml] HTTP/1.1
Host: localhost:8983
```

```
  <doc>
    <float name="price">399.0</float>
    <str name="name">Apple 60 GB iPod
with Video Playback
Black</str>xxxx<a:script
xmlns:a="http://www.w3.org/1999/xhtml">al
ert(1)</a:script></doc>
</result>
```

CASE STUDY: Social Search Problems

An Expedited WAF customer had an API that was used to provide search functionality via a Javascript client on an otherwise static site.

Each search result called the API once, and there was a relatively small amount of traffic using the search function as they were a boutique seller of multimedia resources.

"Generating a search result page is the most execution heavy action on our site and it looked like half of China was querying simultaneously."

One of their resources went virally popular on a Chinese social network, but what was passed around was the results URL of the search page. Each new preview, click, and share resulted in a new API call that quickly brought down the site.

Using Expedited WAF, the team temporarily blocked access to the site from China and was able to remain online to their business customers.

XSS (Cross Site Scripting) & Code Injection API Attacks

Classically an XSS attack occurred when one user of a site was able to upload some code (typically Javascript) into a field that would later be displayed to other users on a site.

Code Injection is a more encompassing term that better represents the scope of challenges an API faces in keeping scripts and executable content out of their system.

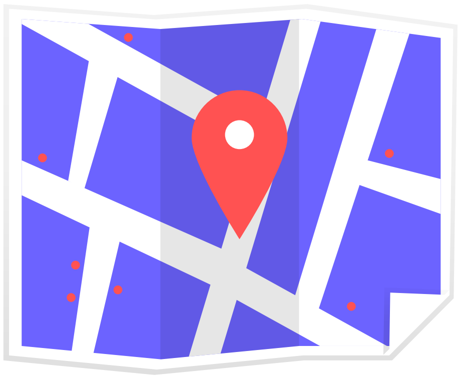
Most app frameworks have some limited ability to filter HTML and Javascript from form inputs; however, this gets complicated when payloads are wrapped in additional layers of abstraction and encoding.

Counters to this will be covered in the Data Validation section.

3 Functionality and Resource Attacks on APIs

APIs are built to expose the underlying functionality of a service. This underlying functionality can be in and of itself valuable to cybercriminals. These resource and functionality attacks may be launched via other vulnerabilities but often are simply unexpected uses of otherwise innocent features.

The following are general areas of functionality that are often abused and deserve special monitoring for abuse.



Email Sending

Spammers are constantly trying to break into legitimate APIs that have some communications mechanisms. Most often, it's email spammers selling shifty pharmaceuticals, but any type of unfettered email access is likely to be exploited. To the greatest extent possible, you should lock down email subjects and content to predefined messages that can't be customized.

Location Tracking

Location tracking (and especially real-time location tracking) can be exploited by stalkers and other criminals. This doesn't need to be a set of Lat/Long coordinates either, like:

- IP Addresses which can be run through services like IP to Earth to determine (rough) location.
- EXIF data in uploaded images which often contains location information unless deliberately stripped, see Pic2Map

IP Addresses and Image Uploads both leak location data unless properly filtered.

Link and File Publishing

Letting users modify content via an API is a dicey proposition (even if everything is functioning in a secure and error free way) as there may be unintended consequences.

Link Publishing

A page's rank in search engine results is, in large part, based upon how many other sites are linking to it. If there's any functionality within your API to add links (or freeform text with links in it), then it's quite likely that you'll be exploited by those seeking to inflate their search engine results.

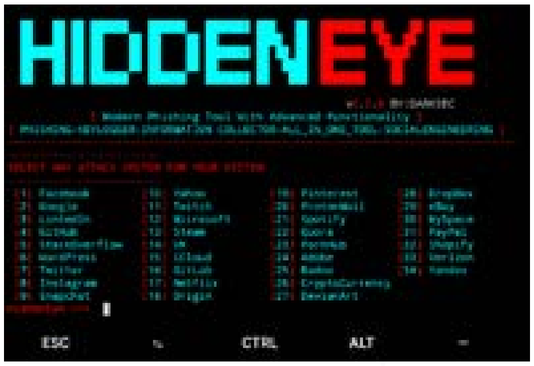
File Uploads

For reasons similar to link publishing, file uploading is often exploited by those seeking to distribute unethical content or just looking to boost their search rankings. For years a popular black hat SEO technique was to attempt to upload HTML files full with links pointing to their clients' sites to any location that failed to filter their uploads properly. Often even a "failed" upload via the API and validation mechanisms still left a file in a web-accessible location.

Phishing

One on one messaging within apps has been abused for Phishing and other scams to the point where if you include this functionality within your API, you should build in some protection mechanisms for your users.

To get a sense of how bad it's gotten, check out the HiddenEye tool from DarkSecDevelopers.



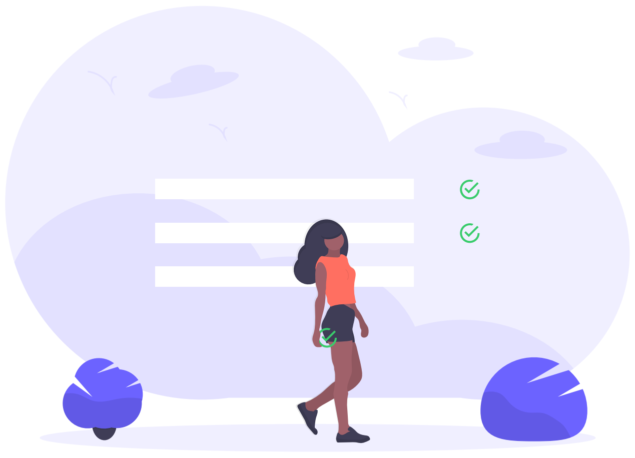
```
                                        v1.1.0 BY:DARKSEC
          [ Modern Phishing Tool With Advanced Functionality ]
[ PHISHING-KEYLOGGER-INFORMATION COLLECTOR-ALL_IN_ONE_TOOL-SOCIALENGINEERING ]
SELECT ANY ATTACK VECTOR FOR YOUR VICTIM:
 [1] Facebook        [10] Yahoo       [19] Pinterest       [28] DropBox
 [2] Google          [11] Twitch      [20] ProtonMail      [29] eBay
 [3] LinkedIn        [12] Microsoft   [21] Spotify         [30] MySpace
 [4] GitHub          [13] Steam       [22] Quora           [31] PayPal
 [5] StackOverflow   [14] VK          [23] PornHub         [32] Shopify
 [6] WordPress       [15] iCloud      [24] Adobe           [33] Verizon
 [7] Twitter         [16] GitLab      [25] Badoo           [34] Yandex
 [8] Instagram       [17] Netflix     [26] CryptoCurrency
 [9] Snapchat        [18] Origin      [27] DevianArt
[illegible]
```

HiddenEye will generate phishing assets for Facebook, Google, Twitch...

4 OWASP Top 10 API Vulnerabilities

The OWASP (Open Web Application Security Project) Top 10 vulnerabilities are best thought of like a real-world survey of the attacks that are most often being seen by sysadmins and security professionals that are working to secure APIs day in and day out.

Released each year, they're a general guide to which threats you should be paying the most attention to countering in your API security processes.



Why care about the OWASP Top 10

We've covered most of this list elsewhere in the guide, but the OWASP Top 10 often comes up in security discussions, and it can be helpful to see it organized in this form as it specifically applies to API security.

> ***The OWASP list is at its heart a method of prioritization. It's your peers saying that they have been affected by these very specific categories of attacks and that you should prepare your systems to defend against them as well.***

There will always be more you could be doing to secure your systems, the OWASP list is at its heart a method of prioritization. It's your peers saying that they have been affected by these very specific categories of attacks and that you should prepare your systems to defend against them as well.

Is the OWASP Top 10 for APIs?

No, but it is still mostly applicable, we've tweaked the items below to better represent the state of the industry with regards to API Security.

1) Code Injection

What is it

Any attack where code is injected to extract data spread malware or otherwise disturb the normal functioning of your API.

What you should do

Filter all inbound inputs. Limit the amount of customizability and separate free-form entry from structured data storage.

2) Broken Authentication

What is it

Insufficiently strong authentication (either in the cryptographic sense of "weak" ciphers) or improperly chosen or applied authenticaiton mechanisms.

What you should do

Choose the Authentication pattern that best matches your API needs, layer additional authentication methods, and encryption on your base authentication scheme until it matches your level of paranoia.

3) Sensitive Data Exposure

What is it

The unintentional revealing of data from the API that could be impactful in some way. Examples of sensitive data are things like physical location, gender, medical history, financial standing, etc.

What you should do

Carefully audit all data returns to make sure that only the proper data fields are exposed. Avoid coding defaults that publish all data fields of an object by default. Deliberately choose which fields are displayed in all cases. 4) XML External Entities (XXE)

CASE STUDY: Devise Auth IP

Devise is an excellent user authentication system for Rails applications that among other things will log the IP address used by the client at login. A typical implementation would add that to the main User model in the application.

A naive API implementation of user profiles might then inadvertendly expose that IP address.

4) XML External Entities (XXE)

What is it

An attack against XML parsing engines used to process XML based requests.

What you should do

Keep your XML parsing libraries up to date.

5) Broken Access Controls

What is it

Insufficient bounds being placed on users with legitimate access. Examples would be ordinary users that can access administrative functions in an API.

What you should do

Audit your system for out of bounds access.

6) Security Misconfiguration

What is it

Today’s systems are complex enough that it’s not always clear what the end result may be and who can access it.

A common example is that many APIs using Amazon Web Services will output data processing jobs to S3, but the S3 bucket itself will be improperly permissioned.

> ***“If the API is secure, but the output is written to S3 where due to a misconfiguration, the data is not secure, then the API is actually not secure.”***

What you should do

Test and document both internal and external systems for for misconfigurations.

7) Cross Site Scripting (XSS)

What is it

Manipulation of your API to further spread a malicious script.

What you should do

Strongly filter all inputs for both corectness and script components.

8) Insecure Deserialization

What is it

Any time that a data object is transformed from one notation system to another, there is a danger that the system doing the deserialization could itself have vulnerabilities that could be exploited.

What you should do

Keep your serialization and deserialization libraries up to date, filter inputs for malicious inputs.

9) Vulnerable Components

What is it

Modern web development is, in many ways, a system of plugging many different libraries and services together to create the desired result. While this is great for programmer productivity, it implicitly requires that all components are tested and updated in perpetuity.

What you should do

Use one of the many services that automatically tracks the packages included in your framework and alerts you to insecure versions.

10) Insufficient Logging and Monitoring

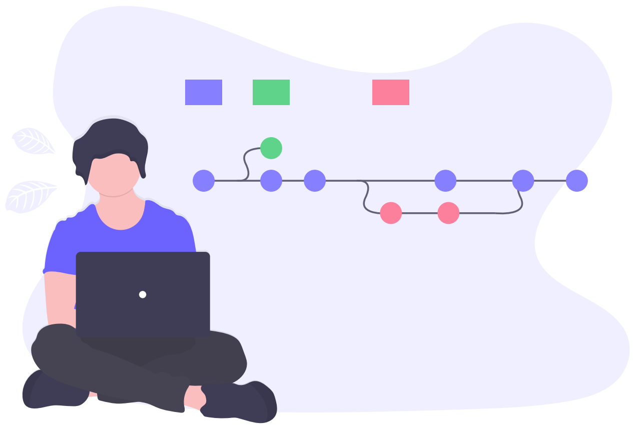
What is it

Most security breaches are only discovered far after they've actually occurred. Determining what data was affected, what systems were touched, and what steps need to be done to remediate the situation all hinge on having good logs of access, security alerts, and user activity.

What you should do

Define a data retention policy that includes how far back logs will be kept. Instrument your API access actions to record key metrics and events. Keep logs indexable and searchable.

5 API Security Controls



With respect to security functions, “controls” are any feature or process that lets you respond to a threat. Data privacy and security regulations will often invoke language like “…application should have controls in place to counter threats such as X, Y, and Z” where XYZ are cybercriminals setting your API on fire.

The following security controls can help to blunt Denial of Service, Data Breach, and other attacks aimed at your API. No control is 100% effective 100% of the time in countering a threat. For this reason, it’s important to layer them along with your application-level security precautions to achieve security in depth.

The following are general techniques and approaches (controls) to improving the overall security of your API.

Most tend to cross multiple different areas of threats. For example: blocking attack probes tends to stop both DDoS attack probes and framework vulnerability probes.

Stop Anonymous Proxy Networks

An attack originating from a handful of IP addresses is trivially easy to stop with a solution like Expedited WAF. Attackers counter to this is to obscure the true origination point of attacks behind what are known as “anonymous proxies.”

While there do exist legitimate anonymous proxy providers, most often “anonymous proxy” is a polite fiction for “a PC in someone’s house that’s been infected with malware and is being remotely manipulated.”

These networks are difficult to counter on your own, and it’s only by continuously combing shared security vendor lists of identified anonymous proxies and our data from over 20,000 sites, that we’re able to block anonymous proxies at the network level.

Blocking anonymous proxies has the dual benefit of blocking both DDoS requests but also the source of many web attacks against applications.

Designate Allowed IP Ranges

Depending upon the usage model of your API, you can further extend the security of your endpoints by only allowing designated IPs and IP ranges. Many popular API’s use explicitly allowed IPs in conjunction with Tokens or API keys as a means of layering an additional level of security on their endpoints. A good example is the Namecheap Domains API, which is used by thousands of organizations.

In a cloud environment like Heroku (where hosts are referenced by name and not IP), clients will need to use an add-on like one of the following to consistently make all API calls from the same IP address. There are a number of services designed to help with situation such as:

- Fixie
- QuotaGuard Static
- Proximo

CASE STUDY: Testing Bots In Production

A newly launched API service had set up Expedited WAF in anticipation of meeting regulatory security requirements but was surprised when they were being hit multiple times a day by a bot that was methodically working through their entire set of API endpoints.

They restricted allowed IP requests to a small set of known values. Immediately they got a call from one of their remote developers who had incorrectly configured their test suite to run against the production server instead of the unprotected WAF staging instance.

Geo Filtering

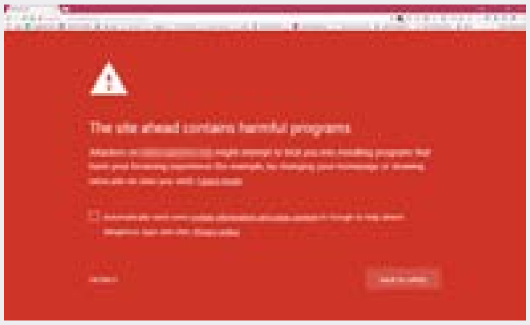
While theoretically, attacks against your API can originate from anyplace on the Internet, practically, they often originate from a single country. The reasons for this vary, but it is a mix of certain countries having laxer cybersecurity regulations, other countries with a computer population more likely to be taken over for botnets or simply that attackers find it easier to break into computers in their native language.

With that in mind, the ability to quickly block a country from GET/POST requests against your API may be the fastest way to stop an attack in progress.

CASE STUDY: South African NSFW Links

An Expedited WAF customer with a landing page creation service for mobile users had an API that was being exploited by unscrupulous people in South Africa to "launder links" to various NSFW sites. They'd sign up for new trial accounts, get fresh API keys, and then script the automatic creation of thousands of landing pages with their links embedded.

As a result, the landing page service itself was classified by Google as an "Unsafe Site" and Chrome users started seeing red error messages the one shown below when browsing to the site.


The site ahead contains harmful programs

[illegible]

Failing to Kill NSFW Links may lead Google to block your site.

As part of the cleanup effort, the customer was able to block the offending customers and regain their safe site status.

Filter Bots from your Single Page Applications and Browser APIs

Single Page Applications often are architected with a front end framework like React, Angular, or Vue calling a backend API on each interaction.

By necessity, the API endpoints are readily discoverable by anyone with the technical ability to right-click on the page and open developer tools.

Attackers will take these endpoints and attempt to brute force or otherwise manipulate them with command-line tools.

Expedited WAF can fingerprint bot activity and block these attempted endpoint enumerations.

CASE STUDY: Auto Scaling for SEO Bots

An Expedited WAF protected application was using an auto-scaling plugin to add additional resources when the performance of the application slowed under load.

Once a week like clockwork, they'd have a spike in load, and for several hours their system would burn 10x their standard workload.

"We didn't realize there were that many dumb bots out there."

However, this traffic spike wasn't accompanied by any of the user signup and sales metrics that they should have been seeing. They tracked it down to a bot from a known SEO spider that was aggressively indexing their site, disregarding their robots.txt directives and costing them thousands in server fees.

Expedited WAF's bot blocking features were enabled, and they discovered that beyond the one bot, many were ripping through their site and that what they considered their "base" load was actually substantially less than they had thought.

6 API Security in Transit (SSL/TLS/HTTPS)

SSL/TLS (HTTPS) is by far the most commonly used in transit encryption mechanism used for web-based HTTP APIs.

Secure API connections are not without challenges as much of the security provided by HTTPS connections is a result of web browser innovations like HSTS, protocol negotiations, and others that help enforce higher levels of security.

In contrast, most HTTP request/response libraries used by API clients either do not have or do not enable similar features by default.

Expedited WAF's HTTPS enforcement features help reduce the chances of an insecure connection by eliminating insecure options and blocking circumvention attempts.

HTTPS Enforcement for APIs

All requests from clients to your API should be encrypted (HTTPS). Unfortunately many client HTTP libraries do not enable HTTPS/secure connections by default so it's necessary to enforce that from the server.

Unfortunately many client HTTP libraries do not enable secure connections by default.

Expedited WAF implements HTTPS enforcement as a standard (status code 301) redirect. Clients who attempt to connect via HTTP are seamlessly redirected to secure HTTPS connections with no opportunity to force an unsecured connection.

TLS Version Enforcement

Most http client libraries are capable of interacting with multiple different version of the SSL/TLS specification. A modern web browser like Chrome will preferentially choose the most secure/latest version of TLS (1.3).

Many HTTP libraries have a different handshake model that instead works UP from the oldest version of TLS to the latest (and the underlying SSL/TLS library in use and hasn't been updated).

The result can be connections that are much more easily spoofed or broken. Expedited WAF enforces v1.2 and higher connections by default preventing downgrade attacks and poorly configured clients from connecting insecurely.

7 API Authentication Methods

There's no one "best" method of securing an API you need to choose a method based on:

1. How many clients you need to support
2. Sensitivity of data
3. Process concerns (revokability, expirations)
4. Platforms to support
5. Developer experience
6. User Enablement
7. Regulatory Guidelines
8. Logging/Tracking Needs

API Key Authentication

Static API keys are one of the classic ways that API's have been accessed since web applications have been brought online.

When to Use API Keys

API Keys support a very simple interaction model that works well for API clients that:

- Directly access resources
- Work on platforms that can keep the key secret (server applications)

Allow for multiple keys

Users of your API should be able to generate multiple keys. This allows for more fine-grained access and tracking across development, testing, staging, and production environments.

Depending on your system it may even make sense to allow for individual API creation for each developer or application consuming the API

API Key Storage

Client code examples should be generated with the assumption that the API keys are being loaded from either a secure key store. a non-source controlled environmental secrets file or in the case of Heroku their environmental variable configuration feature.

Allow self-revocation

Users should have the ability to revoke the current authorization of an API key. Revocation is needed in situations where the key may have been exposed to unauthorized users.

Example: A key is accidentally checked into and then deployed to a public Github repository.

Allow Expiration

In some cases allowing for expiring API Keys is advantageous to limit the total number of keys in use.

API's Using API Keys for Authentication

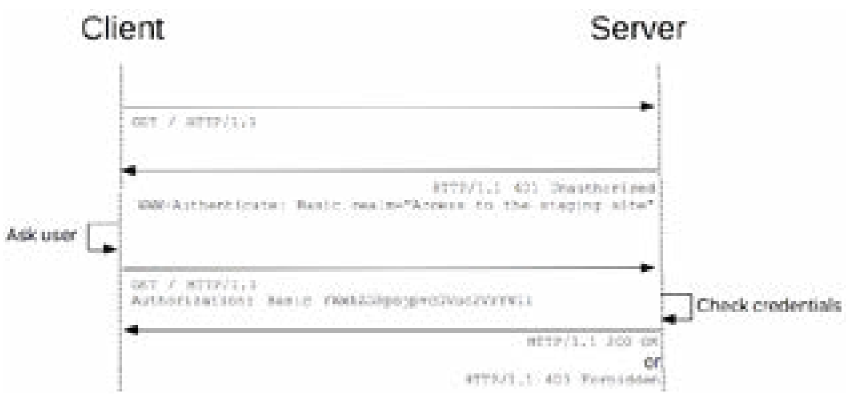
The following prominent services use API Keys for aspects of their Authentication schemes.

- IBM Cloud API
- Data.gov API
- Google Cloud API

Basic Auth API Authentication

Basic Auth passes unencrypted security credentials (username and password) in a standard HTTP header to the API endpoint. This allows it to function with almost every HTTP library and API consumption application that exists.

Visit the Mozilla Developer site for a more in-depth explanation of Web/Basic Authentication.


Client
Server
GET / HTTP/1.1
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to the staging site"
Ask user
GET / HTTP/1.1
Authorization: Basic [illegible]
Check credentials
HTTP/1.1 200 OK
or
HTTP/1.1 403 Forbidden

When to use Basic Auth

In modern web applications, Basic Auth is most often mixed with another API authentication scheme to create a more secure system. It's unlikely that standard basic auth is something that you should depend upon for your standard API Authentication.

Must use with HTTPS

Basic Auth encodes credentials in Base64, but as they are not encrypted in transit unless you are also enforcing HTTPS connections, the credentials can be read by any intermediate in their path to the server. If you're using Heroku, Expedited WAF will enforce Full HTTPS compliance.

Examples of Basic Auth combined with other schemes

There are a number of notable API's that use Basic Auth together with API Keys or generated Tokens to add more security depth to their application.

- Heroku Add-ons API
- Jira Cloud

Client Certificate API Authentication (X.509)

Many developers are familiar with TLS/HTTPS certificates that are used to secure server communications. Lesser known is that you can issue Public Key Infrastructure (PKI) certificates to a client that are used for authentication.

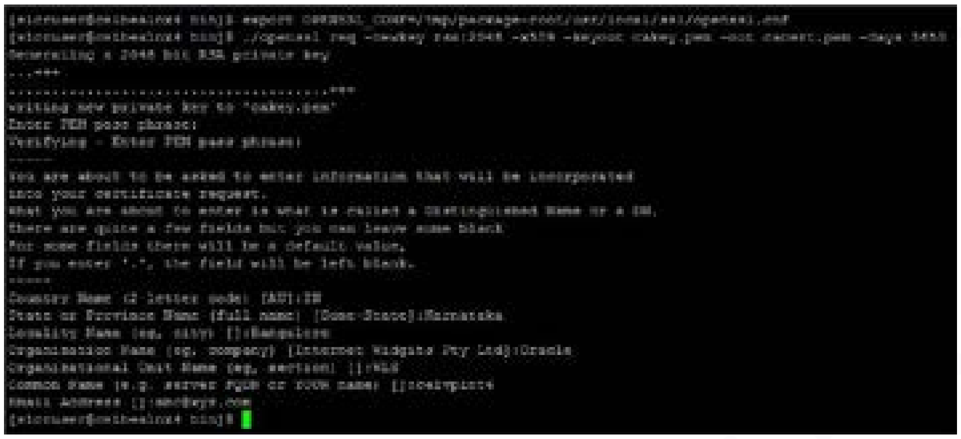
When to use Client Certificates

Client certificate authentication adds another layer of cryptographic complexity to the authentication and transmission of API communications. The overhead of managing, distributing, and supporting client certificates is non-trivial.

This scheme is best used when you have a limited number of very high-value API clients and sufficient support resources to assist them.

How it Works

Client and Server certificates are generated with the same set of keys and need to remain in sync. When a connection is first made, both the client and server certificates are validated to each other, and if either has expired, been invalidated, or have mismatched information, the underlying API connection (HTTP request) won't be established.



```
[illegible]$ export OPENSSL_CONF=/tmp/package-root/usr/local/ssl/openssl.cnf
[illegible]$ ./openssl req -newkey rsa:2048 -x509 -keyout cakey.pem -out cacert.pem -days 3650
Generating a 2048 bit RSA private key
...+++
........................................+++
writing new private key to 'cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Karnataka
Locality Name (eg, city) []:Bangalore
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Oracle
Organizational Unit Name (eg, section) []:[illegible]
Common Name (e.g. server FQDN or YOUR name) []:[illegible]
Email Address []:[illegible]
[illegible]$
```

OpenSSL Generation of a x.509 certificate

Typically, at the same time, another identity service (such as Active Directory) authenticates the veracity of the client certificate.

Examples of APIs with Client Certificate Authentication

- Microsoft Azure API Management (support for)
- Salesforce API

OAuth 2.0 for APIs

Open Auth (OAuth) combines elements of both authentication (who is connecting to your API) with authorization (what the person connecting is allowed to access). OAuth 2 was designed to be an industry-standard protocol for authentication/authorization and shines when used for APIs with third-party access.

OAuth 1 vs OAuth 2

Despite the simple version bump, OAuth 2 is a much-simplified version of the protocol that's fairly easy to implement (even without a client library).

HTTPS Required

One of the main reasons that OAuth 2 is substantially simpler than OAuth 1 is that OAuth 2 relies upon TLS/SSL to secure transmissions. HTTPS connections are required for a secure OAuth 2 implementation.

When to use OAuth

OAuth is ideal for API ecosystems where there is a primary service that developers are building applications for end users.

OAuth vs API Key

Consider a library that exposes an API for their patrons that a single endpoint that returns all books that a patron has ever checked out, and that lets you check out a book.

GET https://library-example.com/books

API KEY: An API Key authentication model would allow a developer to create an application to consume the API and analyze the books for their own use.

OAUTH 2: An OAuth 2 authentication model would allow a developer to create an application "CheckApp" that other patrons of the library could then use, the Authentication/Authorization could include scoped limits on access, could be easily revoked from the main service and would limit reuse and distribution of usernames/ passwords.

OpenID Connect (OIDC)

OpenID Connect is a further simplified subset of Open Auth 2.0 focused on requesting and returning a small set of user-specific information that could be used across services. While this may sound great for your needs:

1. As soon as you step outside the scopes of profile, email, address, and phone you're back to a full OAuth 2 implementation.
2. One of the promises of OIDC was that it would allow for easier implementations between services, but immediately each service using it opted to fragment their implementation.

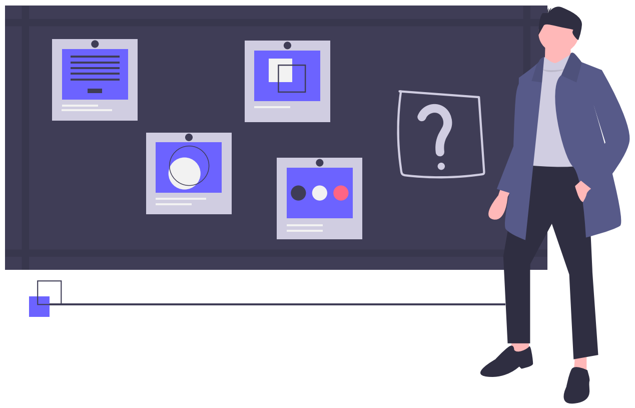
APIs that use OAuth 2 for Authentication

- Github
- Facebook
- Fitbit

8 API Authorization Approaches

Authorization (if a user connected to your API has permissions to perform an action) is an important and often neglected aspect of API creation.

By establishing a consistent model for handling authorizations within your API, you are much less likely to over permission users, allow unsafe defaults to persist, and as a result, you'll deliver an overall safer application.



Role Based Access Control (RBAC)

Roles are collections of authorizations granted to users of your API. As an example, consider Wordpress User Roles. By default, Wordpress creates multiple different roles such as Administrator (complete control of articles and users, Editor (control of all articles, but not users), Author (control of articles they've written but not others), etc.

Wordpress's REST API extends these roles to calls against the API endpoints, so, for example, an Editor is not able to modify a User via the API.

By shipping multiple roles with associated access out of the box with Wordpress, the developers are materially improving the security of the platform.

Attribute Based Access Control (ABAC)

Sometimes called Policy Based Access Control (PBAC) the overall concept of PBAC/ABAC is that rigid access control based on roles is too inflexible and inefficient.

In this authorization model, what is important is both the context of the request and the context of what is being requested.

ABAC Policy Example

Consider an API whose endpoints return a mix of public information (overall system status) and private information (a user's profile information), in an ABAC system two different policies could be applied to those endpoints.

A policy for the public status endpoint might be:

```
IF $client_api_key IS valid
THEN return resource
```

And the private user profile endpoint might be:

```
IF $client_api_key IS valid
AND $client_api_ip_address IS WITHIN $allowed_ip_range
THEN return resource
```

Attributes for Policies

Some common contextual attributes used in creating policies are:

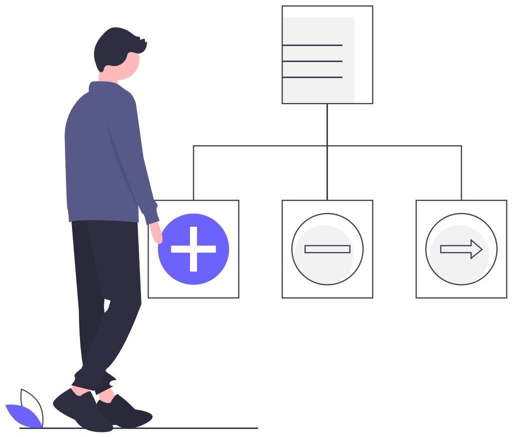
- Client IP
- Time of Day
- Client Request Country (Geo IP)
- Rate Limiting
- User Agent
- VPN/Anonymous Proxy Detected
- Sensitive of Request

Depending upon the resource being requested and the mix of the above factors, additional access requirements might be applied, or access denied.

9 Secure API Design

APIs are by their nature extremely diverse allowing developers to do everything from finding out if it will be sunny tomorrow to destroying 440 million dollars in less than 45 minutes.

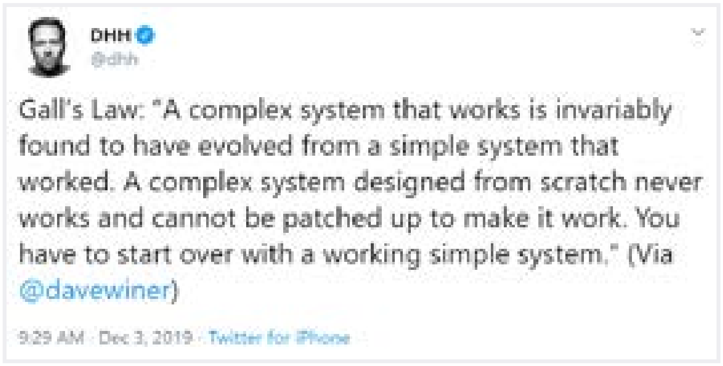
Despite this very disparate nature of APIs, there are still some fundamental approaches that can help you create a secure and solid API.



Simple Means Secure

Static definitions of “simple” and “complicated” never really capture the feelings involved.

Simple is “what I would have done,” and complicated is “what anybody else does,” and on that scale, implementing APIs are always complicated because it’s your thoughts and design that another person is consuming.


DHH
@dhh

Gall’s Law: "A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system." (Via @davewiner)

9:29 AM · Dec 3, 2019 · Twitter for iPhone

Even DHH Agrees

Choose the simplest methods of authentication, authorization, and security controls you can for your API. Leverage industry-standard technologies and don’t implement your own cryptographic solutions. Additionally, prize security over flexibility in your API. Set secure defaults and raise warnings when they are changed.

Allow Least Privilege

Least Privilege as applied to APIs means allowing only enough access to API clients for them to perform what tasks they need to. Additional functionality should only be granted if necessary.

Accomplish this with Role Based Access, separate read/write API Keys, OAuth Scopes, and granular permissions systems.

Be Conservative in Exposing Data

Similar to Least Privilege, you should strive to only include the absolute least amount of data necessary from any API call.

This minimizes the chances that you’ll accidentally expose a sensitive field but also prevents inadvertent disclosures of personal information that can be inferred by correlating data from different datasets (a particular vulnerability of APIs).

A Twitter app bug was used to match 17 million phone numbers to user accounts

17 Million People had their data compromised.

In December of 2019, a design flaw in the Twitter Android API was disclosed. Twitter's API was allowing massive generated lists of phone numbers to be uploaded to their account matching API. Via this method, over 17 million phone numbers were matched to Twitter accounts.

Least Data Collected

Pre Data regulations like GDPR or CCPA and the rash of data breaches destroying people's privacy, there was a collective thought that having data in your system was a good thing.

Currently, holding user data is more like having a toxic asset.

“*Holding user data is like having a toxic asset.*”

Minimize the amount of data you collect. Document why you are collecting each piece of user data. Document their consent in giving you their data.

Versioning

It's inevitable that at some point, you'll need to make potentially breaking changes to your API; at that moment, you'll thank yourself for having the foresight to build request versioning into your API from the start.

Beyond being a nice to have feature for both clients and developers, versioning gives you a clear process to deprecate older API methods that may be less secure than you intended, lets you cleanly swap to stronger authentication methods, and communicates these changes in working code to your API clients.

Stripe's Engineering blog lists some of the steps they've gone through and benefits from implementing versioning as a first-class feature of their APIs.

Logging Requests and Responses

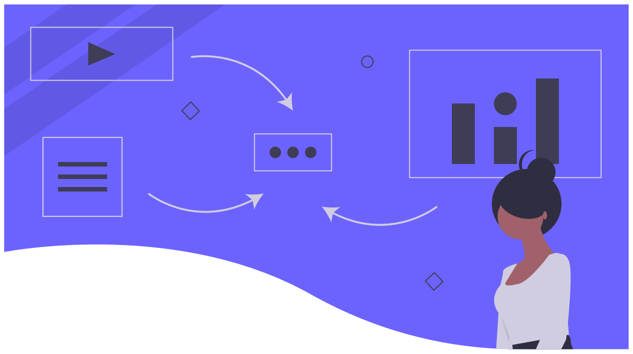
Earlier in this article, we discussed "logging" in the abstract and how many apps don't retain or structure logs sufficiently to investigate a data breach. Now we're focusing specifically on the request/response cycle and a feature you can add to each response to greatly simplify log searching and debugging: adding a request_id.

Typically this is a cryptographic hash of the inputs (request parameters, JSON blog, SOAP Envelope, etc.) that is returned in the response and also logged.

Later if a data dump of the API is discovered, a security researcher reports a problem, or even if you're just debugging, you have a very easy log parameter to search.

10 Security Procedures

Handling Security Reports



Even with the best of intentions, latest frameworks and stacks of security controls your API may still have security issues.

As that's the case, it's very important that you have publicly established a contact point where security issues can be reported. Not having a public contact method risks losing security reports in support or customer service queues. The easiest thing is to add a security@example.com alias to your Contact Us page (here's ours).

"You can't walk into a Starbucks, order a Venti Pike Place and casually report that you have gained access into all of their internal administrative applications."

Alternatively you can add a 'security.txt' file to the root of your website, more information here: securitytxt.org.

As an example of how this plays out in practice: Starbucks API developers accidentally left an API key in a public Github repo which granted access to internal Starbucks authentication and SSO services. An independent vulnerability hunter found the key, reported it to Starbucks via their disclosure service and the developers were able to resolve the issue before systems were affected.

Now, consider the case of not having those communications systems in place. You can't reasonably expect to walk into your corner Starbucks ask for a Venti Pike Place and casually mention that if you were so inclined that you could impersonate Howard Schultz on their corporate network if you felt like it.

Clear lines of security communication:

- Reduce the time that vulnerabilities are open
- Encourage threat researchers from going public with vulnerabilities before they're patched
- Improve your overall security posture

Security communication processes are one of the most effective, speedy and cost effective ways to improve your API security.

Data Breach Handling

Depending on your jurisdiction if a data breach has occurred, you'll need to report it to various authorities such as your US State's Attorney General or a EU Country's Data Protection Authority.

In most cases, there are escalating levels of reporting requirements as the level of sensitivity rises (ex: medical data is treated more seriously in a data breach than local weather reports) and the overall number of people affected.

ABOUT

A team of dedicated security professionals

Application and Security Consulting

We got our start building custom web applications for the US Navy, 3M Health Information Services and other high security environments with significant compliance requirements.

To this day our team is still primarily drawn from the veteran community.

Expedited SSL

In 2014 we launched Expedited SSL - a service to automate the installation, maintenance and renewal of SSL/TLS certificates on cloud platforms.

IP Investigator

In 2016 we released a public API exposing our network investigation tools, letting developers build threat intelligence scoring into their own applications.

Expedited WAF

In 2019 we relased Expedited WAF - a preconfigured and tuned Web Application Firewall that can safely be dropped in front of any web app to instantly improve its security profile.

Today

We're continuing on our mission to deliver security products and services that developers can readily use to protect user data, secure their SAAS and feel confident in their security posture.

Expedited**WAF**

BOOK A DEMO