SEEBURGER

White Paper | SEEBURGER BIS API Capabilities

API Security – A Brief Introduction

Table of Contents

APIs Today and Tomorrow

Based on a comprehensive analysis of the market and insights from many industry analysts, predictions suggest that APIs will remain the most common attack vector for businesses and their data. The increasing use of APIs highlights how urgent it is for businesses to give strong security measures top priority, especially when it comes to API security, in order to protect their systems and confidential data from potential attacks. Proactive measures are essential to strengthen defenses and reduce the risks connected with unauthorized access and cyber attacks as the API landscape keeps growing.

What is API Security, and why has it become so important?

API security, is the term used to describe the procedures and policies put in place to protect APIs from cyberattacks, illegal access, and data breaches. APIs enable smooth communication between applications by facilitating data interchange and interaction between various software systems. Since APIs are now widely used in contemporary digital ecosystems, it is critical to ensure their security.

API security is becoming increasingly important for a number of reasons, including:

Rapid Adoption of APIs: APIs are being used by industries at a rapid pace, which has expanded the attack surface for possible threats. It is becoming more and more important to secure these interfaces as more apps depend on them to exchange functionality and data.

Data Sensitivity: APIs handle a lot of sensitive data, such as financial, user, and business-critical data. Unauthorized access to sensitive data may result in serious repercussions, including invasions of privacy, monetary losses, and reputational harm to a business.

Cybersecurity Threats: Organizations face difficulties as a result of the constantly changing cybersecurity threat landscape. Sophisticated methods for exploiting API vulnerabilities are constantly being developed by malicious actors, so businesses need to stay ahead of the game by putting strong security measures in place.

Regulatory Compliance: Stricter controls over the processing and transfer of sensitive data are required by the emergence of data protection laws like GDPR, HIPAA, and others. Organizations must protect API security in order to comply with these regulations and stay out of trouble financially and legally.

Business Continuity: Because APIs are essential to the operation of business processes and functions, any security breach can cause operations to be disrupted, resulting in lost time, money, and possibly even a tarnished reputation among customers.

Organizations invest in best practices and API security solutions, such as encryption, access controls, authentication methods, and recurring security audits, to address these issues. In a world that is becoming more connected and data-driven, proactive API security measures not only defend against possible threats but also strengthen the digital infrastructure‘s overall resilience.

The Effect of Bad API Security Implementation

Venmo's public API is one of the best-known cases in the field of API security and what can be done with collected data. Venmo is a mobile payment service owned by PayPal. With the help of the mobile app, after creating a user account, money transfers can be made to other users. When the company started, the focus was on the exchange of money between friends who shared bills, such as when they went to the cinema together, or had dinner together. To use Venmo, you must create an account, and both the sender and receiver must live in the U.S.


```
{"payment_id": 2141499719669514525, "permalink": "", "via": "",
"transactions": [{"target": {"username": "JohnD", "picture": "",
"is_business": false, "name": "Doe", "firstname": "John", "lastname": "D",
"cancelled": false, "date_created": "2020-02-02T16:57:40", "external_id":
"2141499719669514525", "id": "2141499719669514525"}}], "story_id":
"2141499719669514525", "comments": [], "updated_time": "2020-02-
02T16:57:40", "audience": "public", "actor": {"username": "JaneD",
"picture": "", "is_business": false, "name": "Doe", "firstname": "D",
"lastname": "D", "cancelled": false, "date_created": "2020-02-02T16:57:40",
"external_id": "2141499719669514525", "id": "2141499719669514525"}, "type":
"payment", "created_time": "2020-02-02T16:57:40", "mentions": [],
"message": "Miss you", "action_links": {}, "likes": {"count": 0, "data":
[]}},
```


When creating the user account, which can be done either via the mobile app or the website, basic information must be entered. This includes bank account information, debit cards, credit card information, a username, phone numbers and/or email. The recipient can then be found via these data.

Venmo uses a public API. Unless it is changed in the default settings, all transactions between two persons are public for all to see. The transaction contains first and last name, profile photo, a corresponding message, and the Facebook ID, the transaction date as well as the status of the transaction could be accessed. Security researcher Hang Do Thi Duc used this opportunity in 2018 and evaluated all data records from 2017. The API provided a wide range of information. By analyzing and evaluating this data, it was possible to create detailed profiles about individuals, including potential health conditions.

She was able to do this because the public API provides private information and was not securely configured, and data filters, authentication and authorization mechanisms for the retrieval of data were not available. There were also other serious security mistakes which can be found in OWASP API Security Top 10 – "massive data exposure and no security by design."

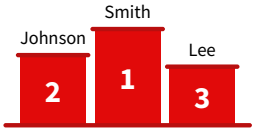
What Can Happen When APIs Are Not Secure

Imagine if this was your critical business data.
The graphic below features the information Hang Do Thi Duc was able to access because of the insecure API used by Venmo.

1,731,783
Facebook IDs

207,984,218
Transactions

18,429,464
Humans



Most frequent
last names

Busiest weekend
December 1-3, 2017
2,342,411 transactions

2,979,616
transactions
for pizza

Profiles

YOLOist

- Female with a Greek name
- Friends live in Texas and Mexico City
- 865 transactions for soda, alcohol, fast food and sweets in 8 months
- Uses 1-2 words or an emoji to describe things
- Follows routines

The All Americans

- Couple
- Own a car and a dog
- Favorite pizza is Shakey's
- Live in Orange County, CA
- Brand name shoppers
- Visit theaters
- Favorite grocery store is Walmart
- Goes shopping more than once a week, but never on Wednesdays or Sundays

Protecting Your Data: Mechanisms and Technology

Auth (n/z)

Auth (n/z) stands for authentication and authorization. Authentication and authorization are broad terms using multiple approaches to secure, identify and enforce rights management. Authentication deals with the identity of the user. For authentication mechanisms are implemented which answer two questions: Is this the person who he/she claims to be, and who is the person? Authorization, on the other hand, is used to answer the question, what is this person allowed to do? For example, it decides what data the requesting instance is allowed to see.

As a practical example, take a journey by train or a flight. During check-in, the identity card is shown, which authenticates the person, as it clearly shows who the person is. The plane or train ticket represents the authorization token. It indicates what the person is allowed to do. In this case, it shows which flight they are on and where the person may sit. Within APIs, auth (n/z) identifies who calls the corresponding API and what rights this person has, i.e. what this person is allowed to do or what data a calling instance is allowed to see.

> “Hello, I am John Doe.
> Here, look at my identity card.”
>
> Authentication

> “I have the permission to sit in this seat.
> Here is my plane ticket.”
>
> Authorization

OAuth2.0

OAuth2.0[1] represents the standard for authorization of a third party to obtain access to resources, which belong to someone else – RFC 6749. OAuth2.0 gives websites or applications access to someone else’s data without forwarding the actual password to them.

OAuth2.0 has four roles:

01 Resource owner:
Owner of the requested resource that allows the client to use the resource.

02 Client:
The requesting instance that wants to use a resource.

03 Resource server:
The resource server hosts the protected resource requested by the client and belonging to the resource owner.

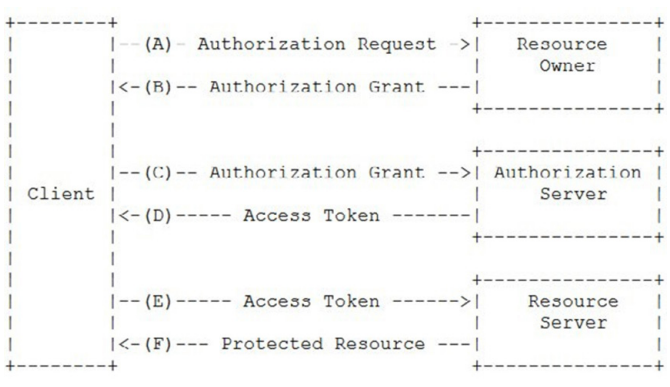
04 Authorization server:
Identifies and authenticates the user and issues access tokens to the client.

A well-known OAuth2.0 example shows the connection between Pinterest and Facebook. Within the online pin board Pinterest, users can add their own Facebook friends as contacts. For this purpose, Pinterest has to access the user’s Facebook information, but without receiving the user’s Facebook log-in data. That’s where OAuth2.0 comes in.

The user (resource owner) is shown a window by Pinterest (client) for the login on Facebook. The user logs in and grants Pinterest access. Pinterest can now request an access token from Facebook’s Authorization Server and with the help of this token, request the resources – in this case the user’s contacts – from the Resource Server. Pinterest now receives the resource as a protected resource. Pinterest only has the rights that it has requested and that the resource owner approves during the entire process and does not know about the user’s login data on Facebook.

Grants include

+ Authorization code
+ Implicit
+ Resource owner password credentials
+ Client credentials

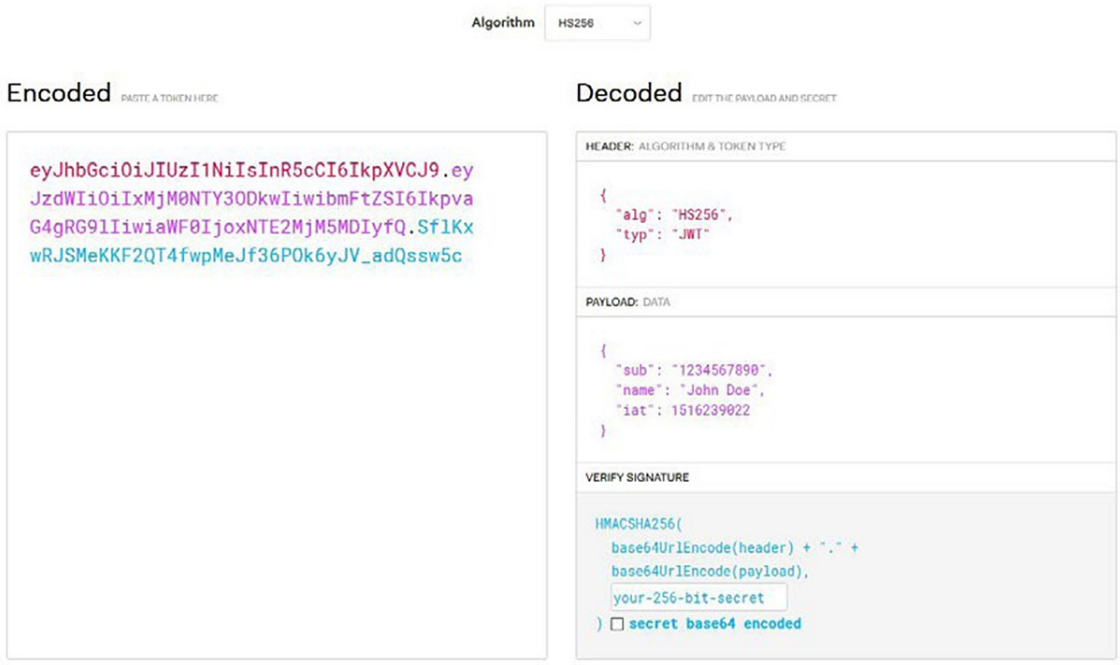


1 https://tools.ietf.org/html/rfc6749

JSON Web Token

A JSON Web Token (JWT, pronounced /dʒɒt/) is an access token using JSON as a basis. It can be used within the OAuth2.0 flows as the access token, which the Authorization Server grants the client. The token is a string that is passed in the header or as a request parameter. Information is exchanged between two parties using the verified claims contained in the JWT. As an open standard (RFC 7519)[2] JWT provides a secure and compact way to exchange this information. Information in the JWT is digitally signed and therefore trustworthy. JWT is a type of container that defines the form of the information that is sent back and forth.

A JSON Web token always consists of three parts, which are marked in the following figure by the three colors red, pink and blue.



Header (red)

The header indicates which algorithm was used to sign the token. In this example, it is HMAC SHA256, which is represented by “alg”: “HS256”. The token type, JSON Web Token, is also specified. The data stored in JSON format is Base64 encoded.

Payload (pink)

The payload represents the actual information in JSON format. The key/value pairs representing the information are called claims. The payload in the example contains the information “sub”, “name” and “iat”. Exactly like the header, the payload is also Base64 encoded.

Signature (blue)

The signature is derived from the header and is defined using the algorithm standardized in RFC 7515. After that, the signature is – exactly like header and payload – Base64-encoded.

2 https://tools.ietf.org/html/rfc7519

The Central Point for API Security Implementation: The API Gateway

An API gateway is the central point for implementation of security aspects and control. Within the gateway, security topics such as Auth (n/z) can be addressed. An API Gateway protects from data security incidents.

The gateway serves as a sort of gatekeeper that guards the entrance to the data. It is located between the clients and the actual service and represents the single point of entry.

Its job is to:

+ Provide transparency in API-related traffic
+ Enable IT to control and fulfill the API demands set by the market and LOB
+ Protect exposed services
+ Secure data
+ Ensure reliable processing of every API call

To fulfill these tasks, various mechanisms are available, including policy enforcement to help with throttling and authentication.

> API Gateways provide transparency in API-related traffic and enable the fulfillment of API demands set by the market and line of business.

The SEEBURGER BIS Platform's API Gateway

Full lifecycle API management such as provided by the SEEBURGER BIS Platform's API capability, helps to capture and control the consumed and provided APIs. One of the essential components of API management is the API gateway, which also plays an important role in the area of API security.

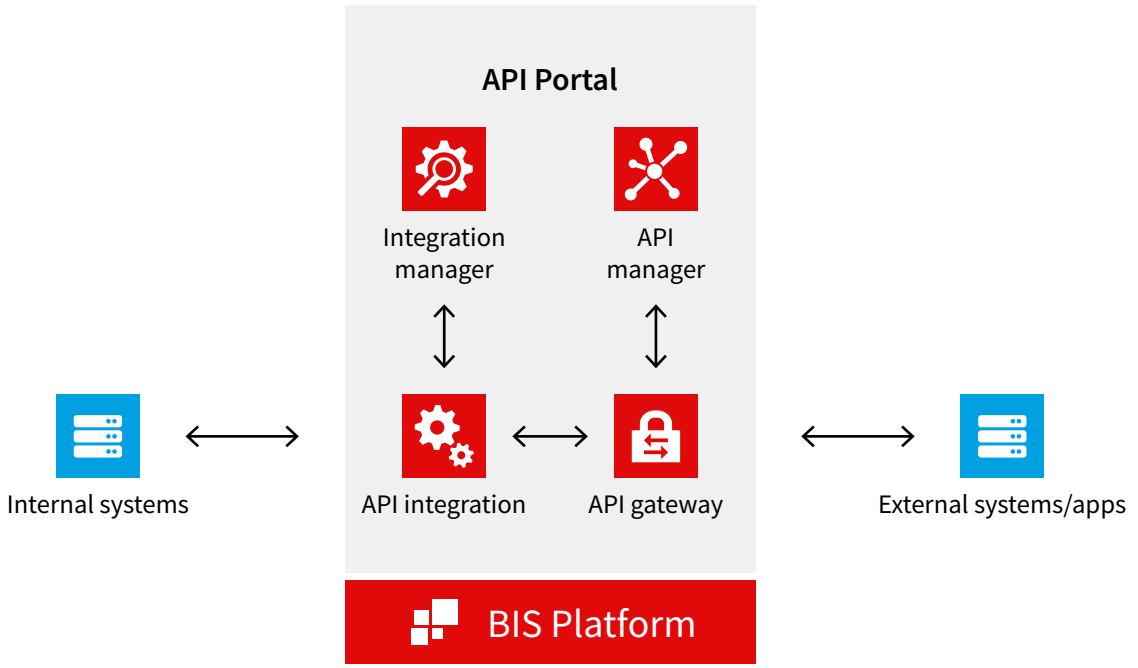
SEEBURGER API Gateway is a preconfigured shipped part of BIS API Management. All data traffic related to APIs goes through the gateway, which is the central point of the solution. Using the API Gateway increases transparency, controls data flows, helps protect exposed services, secures data and provides reliable processing of every API call.

Although multiple gateways can be used in parallel or sequentially, they are not managed individually, but centrally via a single GUI – the BIS API Manager. Install the gateway(s) with the help of the BIS Landscape Manager, and receive manual, partial or fully automated updates, depending on your needs.

The following figure shows the data flows between calling instances, and public and private APIs through two gateways, which are centrally managed by the BIS API Manager.

Supported functionality

+ Supports parallel operations of several instances (Active / Active) and ensures scaling, high availability, and updates without downtime.
+ Provides a user exit, in the event that an individualization deviating from the SEEBURGER standard, is required.
+ Comes with a comprehensive set of predefined policies that can be easily expanded by using Policy Designer capabilities in Developer Studio.
+ Functions of the BIS API Gateway are based on the http adapter layer.
+ Uses in-memory channel processing.
+ Minimal latency.



Policy Architecture

Policies are rules that serve to design and secure the underlying services and therefore determine the behavior of the API. Policies can be assigned to different groups, such as traffic management or security. The structure of a policy is always the same – at least one rule to be checked and the resulting action to be executed. All of this happens inside the API gateway, which is responsible for policy enforcement.

BIS API Gateway checks policies based on the http request information:

+ Caller IP
+ URL, which consists of Protocol, Host, Port, Path, Query Parameters
+ (http://petstore.swagger.io:80/v2/pet/5?name=dog)
+ Header = a set of key/value pairs
+ Content
  + REST: content = payload
  + WebService define SOAP content with additional SOAP Header, payload is inside
+ AS2 uses S/mime with additional headers, payload inside

Rate-limit-by-key

It is important to restrict the number of times an API can be called within a certain time range, for example, to protect the backend systems which can only handle a certain number of requests in a certain time period, or to protect against DoS attacks.

```
rate-limit-by-key {
    @calls := 10
    @renewal-period := "M1"
    @refresh-increment := 60
    @counter-key := "key-to-limit-call"
}
```

Parameter	Description
Calls	The maximum number of calls that can be done before 429 “too many requests” is returned. Checks the counter service to see whether the counter for the given key has exceeded the maximum or not.
Renewal-period	Renewal-period in which the counter service will decrement the counter. Possible values: • “S10” – 10 seconds • “S30” – 30 seconds • “M1” – 1 minute • “M10” – 10 minutes • “M30” – 30 minutes • “H1” – 1 hour
Refresh-increment	In case the number of calls exceeds the accepted limits, all further calls are blocked until the renewal period is over. After this, the counter value is decremented by the value set in refresh-increment.
Counter-key	The name of the key that we are using to limit the number of calls. Each time a call is done, a counter for the given key will be incremented.

Validate-jwt

As mentioned above, JWTs are a common type of access token which contain user information and permissions. It is important to validate any given JWT. If validation fails, a force reply will be given. Otherwise, the request is simply forwarded to the next set of policies.

```
validate-jwt {
    @header-name := "some-header-name"
    @failed-validation-httpcode := 409
    @failed-validation-error-message := "Access Denied"
    @token-value := "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
    @require-expiration-time := "false"
    @require-signed-tokens := "true"
    issuer-signing-keys {
        key := "secret"
    }
    audiences {
        audience := "test-audience"
    }
    issuers {
        issuer := "test-issuer"
    }
}
```

Parameter	Description
Header-name	The name of the header that contains the JWT token. If the attribute “token-value” is already set, this will be ignored.
Failed-validation-httpcode	The http code that the user will obtain as response when the validation fails.
Failed-validation-error-message	The message that the user will obtain as response when the validation fails.
Token-value	The JWT
Require-expiration-time	“true” or “false”. Setting this to “false” means that using an expired JWT token is allowed.
Require-signed-tokens	“true” or “false”. Setting this to “false” means that the JWT token will not be validated against the “signing-key”.
Issuer-signing-keys	Contains all the keys that will be used to validate the JWT token. You can add multiple keys and they will be validated one by one until the correct key is found or all the keys have been tested.
Audience	Lists the expected audiences. The token will be validated to see whether the expected audiences match the token’s target audiences.
Issuers	Lists expected issuers. The token will be validated to see whether the token’s issuer can be found in the list of expected issuers.

Today's API Security Risks

The information in this document about threats and protection mechanisms is by no means exhaustive. The possibilities of attack are at least as varied as the possible protective measures. When it comes to the security of web applications and APIs, Open Web Application Security Project (OWASP) is an authority. OWASP is a community that published its first security document in 2003. Since then, there have been documents, tools, videos and the OWASP forum covering a multitude of topics on security. In 2019 OWASP started focusing on specific topics within API security, and have updated what are considered the top 10 security risks in 2021:

Threat	Description
A01:2021 – Broken Access Control	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.
A02:2021 – Cryptographic Failures	Previously known as Sensitive Data Exposure, which is more of a broad symptom rather than a root cause, the focus now lies on failures related to cryptography (or lack thereof) which often lead to exposure of sensitive data.
A03:2021 – Injection	Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A04:2021 – Insecure Design	Insecure design is a broad category representing different weaknesses, expressed as "missing or ineffective control design." There is a difference between insecure design and insecure implementation because they have different root causes and remediation (i.e., failure to determine required security level).
A05:2021 – Security Misconfiguration	Security misconfiguration is commonly a result of unsecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.
A06:2021 – Vulnerable and Outdated Components	Unknown versions of all used components, unsupported or out of date software, the failure to regularly scan for vulnerabilities and fix or upgrade underlying platforms, frameworks and dependencies, the failure to test the compatibility of updated, upgraded or patched libraries or to secure the components' configurations – all these are examples for vulnerable and outdated components.
A07:2021 – Identification and Authentication Failures	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising system's ability to identify the client/ user, compromises API security overall.

Threat	Description
A08:2021 – Software and Data Integrity Failures	Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations if for example an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.
A09:2021 – Security Logging and Monitoring Failures	Security logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems to tamper with, extract, or destroy data. Most breach studies demonstrate the time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.
A10:2021 – Server Side Request Forgery (SSRF)	SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL). As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

SEEBURGER

www.seeburger.com